



DANIEL J. DENIS

# Multivariate Statistics and Machine Learning

*Multivariate Statistics and Machine Learning* is a hands-on textbook providing an in-depth guide to multivariate statistics and select machine learning topics using R and Python software.

The book offers a theoretical orientation to the concepts required to introduce or review statistical and machine learning topics, and in addition to teaching the techniques, instructs readers on how to perform, implement, and interpret code and analyses in R and Python in multivariate, data science, and machine learning domains. For readers wishing for additional theory, numerous references throughout the textbook are provided where deeper and less “hands-on” works can be pursued.

With its unique breadth of topics covering a wide range of modern quantitative techniques, user-friendliness, and quality of expository writing, *Multivariate Statistics and Machine Learning* will serve as a key and unifying introductory textbook for students in the social, natural, statistical, and computational sciences for years to come.

**Daniel J. Denis, Ph.D.**, is Professor of Quantitative Psychology at the University of Montana, U.S.A, where he has taught applied statistics courses since 2004. He is author of *Applied Univariate, Bivariate, and Multivariate Statistics* and *Applied Univariate, Bivariate, and Multivariate Statistics Using Python*.



**Taylor & Francis**

Taylor & Francis Group

<http://taylorandfrancis.com>

# **Multivariate Statistics and Machine Learning**

An Introduction to Applied Data Science  
Using R and Python

**Daniel J. Denis**



Designed cover image: Getty Images via edge69

First published 2026

by Routledge

605 Third Avenue, New York, NY 10158

and by Routledge

4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

*Routledge is an imprint of the Taylor & Francis Group, an informa business*

© 2026 Daniel J. Denis

The right of Daniel J. Denis to be identified as author of this work has been asserted in accordance with sections 77 and 78 of the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this book may be reprinted or reproduced or utilised in any form or by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying and recording, or in any information storage or retrieval system, without permission in writing from the publishers.

*Trademark notice:* Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

*Library of Congress Cataloging-in-Publication Data*

Names: Denis, Daniel J., 1974- author.

Title: Multivariate statistics and machine learning: an introduction to applied data science using R and python / Daniel J. Denis.

Description: New York, NY: Routledge, 2026. | Includes bibliographical references and index.

Identifiers: LCCN 2025008558 | ISBN 9781032454276 (hardback) |

ISBN 9781032454283 (paperback) | ISBN 9781003376941 (ebook)

Subjects: LCSH: Machine learning. | Statistics. | Multivariate analysis.

Classification: LCC Q325.5 .D46 2026 | DDC 006.3/1—dc23/eng/20250324

LC record available at <https://lcn.loc.gov/2025008558>

ISBN: 9781032454276 (hbk)

ISBN: 9781032454283 (pbk)

ISBN: 9781003376941 (ebk)

DOI: 10.4324/9781003376941

Typeset in Times New Roman

by Deanta Global Publishing Services, Chennai, India

**Only when humanity can look at animals and see its reflection as it peers into the eyes of kinship, will it become not a complete farce to claim any understanding of them ... and of ourselves.**

**Daniel J. Denis**



**Taylor & Francis**

Taylor & Francis Group

<http://taylorandfrancis.com>

# Contents

|                        |       |
|------------------------|-------|
| <i>Preface</i>         | xxiii |
| <i>Acknowledgments</i> | xxiv  |

## PART I

|                                      |          |
|--------------------------------------|----------|
| <b>Preliminaries and Foundations</b> | <b>1</b> |
|--------------------------------------|----------|

|                                                                     |          |
|---------------------------------------------------------------------|----------|
| <b>Introduction, Motivation, Pedagogy, and Ideas About Learning</b> | <b>3</b> |
|---------------------------------------------------------------------|----------|

|         |                                                                                                                                    |    |
|---------|------------------------------------------------------------------------------------------------------------------------------------|----|
| 0.1     | <i>The Paradigm Shift: What Has Changed</i>                                                                                        | 3  |
| 0.1.1   | <i>A Wide Divide</i>                                                                                                               | 4  |
| 0.2     | <i>A Unified Vision: The Bridge</i>                                                                                                | 4  |
| 0.3     | <i>The Data Science and Machine Learning Invasion: Questions and Answers</i>                                                       | 5  |
| 0.4     | <i>Who Should Read This Book?</i>                                                                                                  | 6  |
| 0.4.1   | <i>Textbook Limbo</i>                                                                                                              | 6  |
| 0.4.2   | <i>Theoretical vs. Applied vs. Software vs. “Cookbooks”</i>                                                                        | 7  |
| 0.4.2.1 | <i>Watered-Down Statistics</i>                                                                                                     | 7  |
| 0.4.3   | <i>Prerequisites to Reading This Book</i>                                                                                          | 8  |
| 0.5     | <i>Pedagogical Approach and the Trade-Offs of Top-Down, Bottom-Up Learning</i>                                                     | 8  |
| 0.5.1   | <i>Top-Down, Bottom-Up Learning</i>                                                                                                | 9  |
| 0.5.2   | <i>Ways of Writing a Book: Making It Pedagogical Instead of Cryptic</i>                                                            | 10 |
| 0.5.3   | <i>Standing on the Shoulders of Giants: A Companion to Advanced Texts</i>                                                          | 10 |
| 0.5.4   | <i>Making Equations “Speak”</i>                                                                                                    | 10 |
| 0.5.5   | <i>The Power of Problems</i>                                                                                                       | 11 |
| 0.5.6   | <i>Computing Languages</i>                                                                                                         | 11 |
| 0.5.7   | <i>Notation Used in the Book</i>                                                                                                   | 12 |
| 0.6     | <i>Nobody Learns a Million Things: The Importance of Foundations and Learning How to Learn</i>                                     | 12 |
| 0.6.1   | <i>Essential Philosophy of Science and History</i>                                                                                 | 13 |
| 0.6.2   | <i>Beyond the Jargon, Beyond the Hype</i>                                                                                          | 14 |
| 0.7     | <i>The Power and Dangers of Analogy and Metaphor: Ways of Understanding</i>                                                        | 15 |
| 0.7.1   | <i>The Infinite Regress of Knowledge: A Venture into What It Means to “Understand” Something and Why Epistemology Is Important</i> | 16 |
| 0.7.1.1 | <i>Epistemological Maturity</i>                                                                                                    | 17 |
| 0.8     | <i>Format and Organization of Chapters</i>                                                                                         | 18 |

|          |                                                                              |           |
|----------|------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>First Principles and Philosophical Foundations</b>                        | <b>20</b> |
| 1.1      | <i>Science, Statistics, Machine Learning, and Artificial Intelligence</i>    | 20        |
| 1.1.1    | <i>Mathematics, Statistics, and Computation</i>                              | 22        |
| 1.1.2    | <i>Mathematical Systems as Narratives to Understanding</i>                   | 22        |
| 1.2      | <i>The Scope of Data Analysis and Data Science: Expertise in Everything!</i> | 23        |
| 1.2.1    | <i>Theoretical vs. Applied Statistics and Specialization</i>                 | 23        |
| 1.3      | <i>The Role of Computers</i>                                                 | 24        |
| 1.3.1    | <i>The Nature of Algorithms</i>                                              | 25        |
| 1.3.2    | <i>Algorithmic Stability</i>                                                 | 25        |
| 1.4      | <i>The Importance of Design, Experimental or Otherwise</i>                   | 25        |
| 1.5      | <i>Inductive, Deductive, and Other Logics</i>                                | 26        |
| 1.5.1    | <i>Consistency and Gödel's Incompleteness Theorems</i>                       | 27        |
| 1.5.1.1  | <i>What Is the Relevance of Gödel?</i>                                       | 29        |
| 1.6      | <i>Supervised vs. Unsupervised Learning</i>                                  | 29        |
| 1.6.1    | <i>Fuzzy Distinctions</i>                                                    | 29        |
| 1.7      | <i>Theoretical vs. Empirical Justification</i>                               | 30        |
| 1.7.1    | <i>Airplanes and Oceanic Submersibles</i>                                    | 30        |
| 1.7.2    | <i>Will the Bridge Stay Up If the Mathematics Fail?</i>                      | 31        |
| 1.8      | <i>Level of Analysis Problem</i>                                             | 31        |
| 1.9      | <i>Base Rates, Common Denominators, and Degrees</i>                          | 32        |
| 1.9.1    | <i>Base Rates and Splenic Masses</i>                                         | 32        |
| 1.9.2    | <i>Probability Neglect</i>                                                   | 34        |
| 1.9.3    | <i>The "Zero Group"</i>                                                      | 34        |
| 1.10     | <i>Statistical Regularities and Perceptions of Risk</i>                      | 35        |
| 1.10.1   | <i>Beck Depression Inventory: How Depressed Are You?</i>                     | 35        |
| 1.11     | <i>Decision, Risk Analysis, and Optimization</i>                             | 35        |
| 1.11.1   | <i>The Risk of Making a Wrong Decision</i>                                   | 36        |
| 1.11.2   | <i>Statistical Lives and Optimization</i>                                    | 36        |
| 1.11.3   | <i>Medical Decision-Making and Dominating Criteria</i>                       | 37        |
| 1.12     | <i>All Knowledge, Scientific and Other, Is Tentative</i>                     | 37        |
| 1.13     | <i>Occam's Razor</i>                                                         | 38        |
| 1.13.1   | <i>Parsimony vs. Complexity Trade-Off</i>                                    | 38        |
| 1.14     | <i>Overfitting vs. Underfitting</i>                                          | 39        |
| 1.14.1   | <i>Solutions to Overfitting</i>                                              | 41        |
| 1.14.2   | <i>The Idea of Regularization</i>                                            | 41        |
| 1.15     | <i>The Measurement Problem</i>                                               | 41        |
| 1.15.1   | <i>What Is Data?</i>                                                         | 42        |
| 1.15.2   | <i>The Philosophy and Scales of Measurement</i>                              | 42        |
| 1.15.3   | <i>Reliability</i>                                                           | 42        |
| 1.15.3.1 | <i>Coefficient Alpha</i>                                                     | 43        |
| 1.15.3.2 | <i>Test-Retest Reliability</i>                                               | 44        |
| 1.15.4   | <i>Validity</i>                                                              | 44        |
| 1.15.5   | <i>Scales of Measurement</i>                                                 | 45        |
| 1.15.6   | <i>Likert Scales</i>                                                         | 45        |

|          |                                                                                                              |           |
|----------|--------------------------------------------------------------------------------------------------------------|-----------|
| 1.15.6.1 | Statistical Models for Likert Data                                                                           | 46        |
| 1.15.6.2 | Models for Ordinal and Monotonically Increasing/<br>Decreasing Data                                          | 46        |
| 1.16     | Overview of Statistical and Machine Learning Concepts                                                        | 47        |
| 1.16.1   | Probably Approximately Correct                                                                               | 47        |
| 1.16.2   | No-Free-Lunch Theorem                                                                                        | 48        |
| 1.17     | V-C Dimension and Complexity                                                                                 | 49        |
| 1.18     | Parametric vs. Nonparametric Learning Methods                                                                | 49        |
| 1.18.1   | Flexibility and Number of Parameters                                                                         | 50        |
| 1.18.1.1 | Concept of Degrees of Freedom                                                                                | 51        |
| 1.18.2   | Instance or Memory-Based Learning                                                                            | 51        |
| 1.18.3   | Revisiting Classical Nonparametric Tests                                                                     | 51        |
| 1.19     | Dimension Reduction, Distance, and Error Functions:<br>Commonalities in Modeling                             | 52        |
| 1.19.1   | Dimension Reduction: What's the Big Idea?                                                                    | 52        |
| 1.19.2   | The Curse of Dimensionality                                                                                  | 53        |
| 1.20     | Distance                                                                                                     | 54        |
| 1.21     | Error Minimization                                                                                           | 54        |
| 1.22     | Training vs. Test Error                                                                                      | 55        |
| 1.23     | Cross-Validation and Model Selection                                                                         | 55        |
| 1.24     | Monte Carlo Methods                                                                                          | 57        |
| 1.25     | Missing Data                                                                                                 | 57        |
| 1.26     | Qualitative Approaches to Data Analysis                                                                      | 58        |
| 1.27     | Chapter Review Exercises                                                                                     | 59        |
| <b>2</b> | <b>Mathematical and Statistical Foundations</b>                                                              | <b>60</b> |
| 2.1      | Mathematical Previews vs. the "Appendix" Approach                                                            | 61        |
| 2.1.1    | Why Previews Are Better                                                                                      | 61        |
| 2.1.2    | About Proofs                                                                                                 | 62        |
| 2.2      | Elementary Probability and Fundamental Statistics                                                            | 63        |
| 2.3      | Interpretations of Probability                                                                               | 63        |
| 2.4      | Mathematical Probability                                                                                     | 64        |
| 2.4.1    | Unions and Intersections of Events                                                                           | 65        |
| 2.5      | Conditional Probability                                                                                      | 66        |
| 2.5.1    | Unconditional vs. Conditional Statistical Models                                                             | 67        |
| 2.6      | Probabilistic Independence                                                                                   | 68        |
| 2.6.1    | Everything Is about Independence vs. Dependence!                                                             | 68        |
| 2.7      | Marginal vs. Conditional Distributions                                                                       | 69        |
| 2.8      | Independence Implies Covariance of Zero, but Covariance of Zero Does Not<br>(Necessarily) Imply Independence | 70        |
| 2.9      | Sensitivity and Specificity: More Conditional Probabilities                                                  | 71        |
| 2.10     | Bayes' Theorem and Conditional Probabilities                                                                 | 72        |
| 2.10.1   | Bayes Factor                                                                                                 | 73        |
| 2.10.2   | Bayesian Model Selection                                                                                     | 73        |

|          |                                                                                 |     |
|----------|---------------------------------------------------------------------------------|-----|
| 2.10.3   | <i>Bayes' Theorem as Rational Belief or Theorizing</i>                          | 74  |
| 2.11     | <i>Law of Large Numbers</i>                                                     | 75  |
| 2.11.1   | <i>Law of Large Numbers and the Idea of Committee Machines</i>                  | 76  |
| 2.12     | <i>Random Variables and Probability Density Functions</i>                       | 76  |
| 2.13     | <i>Convergence of Random Variables</i>                                          | 77  |
| 2.14     | <i>Probability Density Functions</i>                                            | 77  |
| 2.15     | <i>Normal (Gaussian) Distributions</i>                                          | 79  |
| 2.15.1   | <i>Univariate Gaussian</i>                                                      | 79  |
| 2.15.2   | <i>Mixtures of Gaussians</i>                                                    | 80  |
| 2.15.3   | <i>Evaluating Univariate Normality</i>                                          | 80  |
| 2.15.4   | <i>Multivariate Gaussian</i>                                                    | 81  |
| 2.15.5   | <i>Evaluating Multivariate Normality</i>                                        | 82  |
| 2.16     | <i>Binomial Distributions</i>                                                   | 83  |
| 2.16.1   | <i>Approximation to the Normal Distribution</i>                                 | 84  |
| 2.17     | <i>Multinomial Distributions</i>                                                | 84  |
| 2.18     | <i>Poisson Distributions</i>                                                    | 85  |
| 2.19     | <i>Chi-Square Distributions</i>                                                 | 85  |
| 2.20     | <i>Expectation and Expected Value</i>                                           | 85  |
| 2.21     | <i>Measures of Central Tendency</i>                                             | 86  |
| 2.21.1   | <i>The Arithmetic Mean (Average)</i>                                            | 87  |
| 2.21.1.1 | <i>Averaging Over Cases: Why Thinking in Terms of Averages Can Be Dangerous</i> | 89  |
| 2.21.2   | <i>The Median</i>                                                               | 90  |
| 2.22     | <i>Measures of Variability</i>                                                  | 90  |
| 2.22.1   | <i>Variance and Standard Deviation</i>                                          | 91  |
| 2.22.2   | <i>Mean Absolute Deviation</i>                                                  | 92  |
| 2.23     | <i>Skewness and Kurtosis</i>                                                    | 92  |
| 2.24     | <i>Coefficient of Variation</i>                                                 | 94  |
| 2.25     | <i>Statistical Estimation</i>                                                   | 94  |
| 2.26     | <i>Bias-Variance Trade-Off</i>                                                  | 95  |
| 2.26.1   | <i>Is Irreducible Error Really Irreducible?</i>                                 | 96  |
| 2.27     | <i>Maximum Likelihood Estimation</i>                                            | 96  |
| 2.27.1   | <i>Why ML Is So Popular and Alternatives</i>                                    | 98  |
| 2.27.2   | <i>Estimation and Confidence Intervals</i>                                      | 98  |
| 2.28     | <i>The Bootstrap: A Way of Estimating Nonparametrically</i>                     | 99  |
| 2.28.1   | <i>Simple Examples of the Bootstrap</i>                                         | 100 |
| 2.28.2   | <i>Why Not Bootstrap Everything?</i>                                            | 101 |
| 2.28.3   | <i>Variations and Extensions of the Bootstrap</i>                               | 102 |
| 2.29     | <i>Elements of Classic Null Hypothesis Significance Testing</i>                 | 102 |
| 2.29.1   | <i>One-Tailed vs. Two-Tailed Tests</i>                                          | 104 |
| 2.29.2   | <i>Effect Size</i>                                                              | 104 |
| 2.29.3   | <i>Cohen's <math>d</math> Measure of Effect Size</i>                            | 104 |
| 2.29.4   | <i>Are P-values That Evil?</i>                                                  | 105 |
| 2.29.5   | <i>Absolute vs. Relative Size of Effect Size: Context Matters</i>               | 106 |

|          |                                                           |     |
|----------|-----------------------------------------------------------|-----|
| 2.29.6   | Comparability of Effect Sizes Across Studies              | 107 |
| 2.29.7   | Operationalizing Predictors                               | 108 |
| 2.30     | Central Limit Theorem                                     | 109 |
| 2.31     | Covariance and Correlation                                | 109 |
| 2.31.1   | Why Does $r_{xy}$ Have Limits $-1$ to $+1$ ?              | 112 |
| 2.31.2   | Covariance and Correlation in R and Python                | 112 |
| 2.31.3   | Correlating Linear Combinations                           | 113 |
| 2.31.4   | Covariance and Correlation Matrices                       | 114 |
| 2.32     | Z-Scores and Z-Tests                                      | 115 |
| 2.32.1   | Z-Tests and T-Tests for the Mean                          | 116 |
| 2.33     | Unequal Variances: Welch-Satterthwaite Approximation      | 118 |
| 2.34     | Paired Data                                               | 118 |
| 2.35     | Review Exercises                                          | 119 |
| 2.36     | Linear Algebra and Matrices                               | 119 |
| 2.36.1   | Vectors                                                   | 120 |
| 2.36.1.1 | Vector Spaces and Fields                                  | 121 |
| 2.36.1.2 | Zero, Unit Vectors, and One-Hot Vectors                   | 121 |
| 2.36.1.3 | Transpose of a Vector                                     | 121 |
| 2.36.1.4 | Vector Addition and Length                                | 122 |
| 2.36.1.5 | Eigen Analysis and Decomposition                          | 124 |
| 2.36.1.6 | Points vs. Vectors                                        | 125 |
| 2.37     | Matrices                                                  | 126 |
| 2.37.1   | Identity Matrix                                           | 127 |
| 2.37.2   | Transpose of a Matrix                                     | 128 |
| 2.37.3   | Symmetric Matrices                                        | 129 |
| 2.37.4   | Matrix Addition and Multiplication                        | 129 |
| 2.37.5   | Meaning of Matrices: Matrices as Data and Transformations | 130 |
| 2.37.6   | Kernel (Null Space)                                       | 131 |
| 2.37.7   | Trace of a Matrix                                         | 132 |
| 2.38     | Linear Combinations                                       | 132 |
| 2.39     | Determinants                                              | 134 |
| 2.40     | Means and Variances of Matrices                           | 136 |
| 2.41     | Determinant as a Generalized Variance                     | 137 |
| 2.42     | Matrix Inverse                                            | 137 |
| 2.42.1   | Nonexistence of an Inverse and Singularity                | 138 |
| 2.43     | Quadratic Forms                                           | 139 |
| 2.44     | Positive Definite Matrices                                | 139 |
| 2.45     | Inner Products                                            | 139 |
| 2.46     | Linear Independence                                       | 140 |
| 2.47     | Rank of a Matrix                                          | 141 |
| 2.48     | Orthogonal Matrices                                       | 142 |
| 2.49     | Kernels, the Kernel Trick, and Dual Representations       | 142 |
| 2.49.1   | When Are Kernel Methods Useful?                           | 144 |
| 2.50     | Systems of Equations                                      | 144 |



|        |                                                 |     |
|--------|-------------------------------------------------|-----|
| 2.51   | <i>Distance</i>                                 | 147 |
| 2.52   | <i>Projections and Basis</i>                    | 147 |
| 2.53   | <i>The Meaning of Linearity</i>                 | 148 |
| 2.54   | <i>Basis and Dimension</i>                      | 149 |
| 2.54.1 | <i>Orthogonal Basis</i>                         | 150 |
| 2.55   | <i>Review Exercises</i>                         | 150 |
| 2.56   | <i>Calculus and Optimization</i>                | 150 |
| 2.57   | <i>Functions, Approximation, and Continuity</i> | 151 |
| 2.57.1 | <i>Definition of Continuity</i>                 | 151 |
| 2.58   | <i>The Derivative</i>                           | 152 |
| 2.58.1 | <i>Local Behavior and Approximation</i>         | 154 |
| 2.58.2 | <i>Composite Functions and Basis Expansions</i> | 155 |
| 2.59   | <i>The Partial Derivative</i>                   | 156 |
| 2.60   | <i>Optimization and Gradients</i>               | 156 |
| 2.60.1 | <i>What Does “Optimal” Mean?</i>                | 157 |
| 2.60.2 | <i>Minima and Maxima via Calculus</i>           | 158 |
| 2.60.3 | <i>Convex vs. Nonconvex Functions and Sets</i>  | 159 |
| 2.61   | <i>Gradient Descent</i>                         | 160 |
| 2.61.1 | <i>How Does Gradient Descent Find Minima?</i>   | 160 |
| 2.62   | <i>Integral Calculus</i>                        | 162 |
| 2.62.1 | <i>Double and Triple Integrals</i>              | 164 |
| 2.63   | <i>Review Exercises</i>                         | 164 |

### 3 R and Python Software

165

|         |                                                            |     |
|---------|------------------------------------------------------------|-----|
| 3.1.    | <i>The Dominance of R and Python</i>                       | 165 |
| 3.2     | <i>The R Project</i>                                       | 166 |
| 3.2.1   | <i>Installing R</i>                                        | 167 |
| 3.2.2   | <i>Working with Data</i>                                   | 168 |
| 3.2.2.1 | <i>Building a Data Frame</i>                               | 170 |
| 3.2.3   | <i>Installing Packages in R</i>                            | 170 |
| 3.2.4   | <i>Writing Functions in R</i>                              | 171 |
| 3.2.5   | <i>Mathematics and Statistics Using R</i>                  | 171 |
| 3.2.5.1 | <i>Addition, Subtraction, Multiplication, and Division</i> | 171 |
| 3.2.5.2 | <i>Logarithms and Exponentials</i>                         | 171 |
| 3.2.5.3 | <i>Vectors and Matrices</i>                                | 172 |
| 3.2.5.4 | <i>Means</i>                                               | 173 |
| 3.2.5.5 | <i>Covariance and Correlation</i>                          | 173 |
| 3.2.5.6 | <i>Sampling with Replacement in R</i>                      | 174 |
| 3.2.5.7 | <i>Visualization and Plots</i>                             | 174 |
| 3.2.6   | <i>Further Readings and Resources in R</i>                 | 175 |
| 3.3     | <i>Python</i>                                              | 176 |
| 3.3.1   | <i>Installing Python</i>                                   | 176 |
| 3.3.2   | <i>Elements of Python</i>                                  | 177 |

|         |                                                 |     |
|---------|-------------------------------------------------|-----|
| 3.3.3   | <i>Working with Data</i>                        | 177 |
| 3.3.4   | <i>Python Functions for Data Analysis</i>       | 178 |
| 3.3.4.1 | <i>Mathematics Using Python</i>                 | 178 |
| 3.3.4.2 | <i>Splitting Data into Train and Test Sets</i>  | 180 |
| 3.3.5   | <i>Further Readings and Resources in Python</i> | 181 |
| 3.4     | <i>Chapter Review Exercises</i>                 | 181 |

## PART II

### MODELS AND METHODS

183

## 4 Univariate and Multivariate Analysis of Variance Models

185

|       |                                                                            |     |
|-------|----------------------------------------------------------------------------|-----|
| 4.1   | <i>The Classic ANOVA Model</i>                                             | 185 |
| 4.1.1 | <i>Mean Squares</i>                                                        | 188 |
| 4.1.2 | <i>Expected Mean Squares of ANOVA</i>                                      | 189 |
| 4.1.3 | <i>Effect Sizes for ANOVA</i>                                              | 190 |
| 4.1.4 | <i>Contrasts and Post-Hoc Tests for ANOVA</i>                              | 191 |
| 4.1.5 | <i>ANOVA in Python</i>                                                     | 194 |
| 4.1.6 | <i>ANOVA in R</i>                                                          | 196 |
| 4.2   | <i>Factorial ANOVA and Higher-Order Models</i>                             | 197 |
| 4.2.1 | <i>Factorial ANOVA in Python</i>                                           | 198 |
| 4.3   | <i>Random-Effects and Mixed Models</i>                                     | 200 |
| 4.3.1 | <i>The Meaning of a Fixed vs. Random Effect</i>                            | 201 |
| 4.3.2 | <i>Is the Fixed-Effects Model Actually Fixed? A Look at the Error Term</i> | 201 |
| 4.3.3 | <i>Mixed Models in Python</i>                                              | 202 |
| 4.3.4 | <i>Mixed Models in R</i>                                                   | 203 |
| 4.4   | <i>Multilevel Modeling</i>                                                 | 204 |
| 4.4.1 | <i>A Garbled Mess of Jargon</i>                                            | 204 |
| 4.4.2 | <i>Why Do Multilevel Models Often Include Random Effects?</i>              | 205 |
| 4.4.3 | <i>A Priori vs. Post-Hoc “Nesting”</i>                                     | 206 |
| 4.4.4 | <i>Blocking as an Example of Hierarchical/Multilevel Structure</i>         | 207 |
| 4.4.5 | <i>Nonparametric Random-Effects Model</i>                                  | 208 |
| 4.5   | <i>Repeated-Measures and Longitudinal Models</i>                           | 209 |
| 4.5.1 | <i>Classic Repeated-Measures Models</i>                                    | 209 |
| 4.6   | <i>Multivariate Analysis of Variance (MANOVA)</i>                          | 210 |
| 4.6.1 | <i>Suitability of MANOVA</i>                                               | 211 |
| 4.6.2 | <i>Extending the Univariate Model: Hotelling’s <math>T^2</math></i>        | 211 |
| 4.6.3 | <i>Multivariate Test Statistics</i>                                        | 212 |
| 4.6.4 | <i>Evaluating Equality of Covariance Matrices: The Box-M Test</i>          | 213 |
| 4.6.5 | <i>MANOVA in Python</i>                                                    | 214 |
| 4.6.6 | <i>MANOVA in R</i>                                                         | 215 |
| 4.7   | <i>Linear Discriminant Analysis (as the “Reverse” of MANOVA)</i>           | 217 |
| 4.8   | <i>Chapter Review Exercises</i>                                            | 218 |

|          |                                                                                             |            |
|----------|---------------------------------------------------------------------------------------------|------------|
| <b>5</b> | <b>Simple Linear and Multiple Regression Models (and Extensions)</b>                        | <b>219</b> |
| 5.1      | <i>Simple Linear Regression – Fixed Predictor Case</i>                                      | 219        |
| 5.1.1    | <i>Parameter Estimates</i>                                                                  | 220        |
| 5.1.2    | <i>Simple Linear Regression in R</i>                                                        | 222        |
| 5.1.3    | <i>Simple Linear Regression in Python</i>                                                   | 224        |
| 5.2      | <i>Multiple Linear Regression</i>                                                           | 226        |
| 5.2.1    | <i>Minimizing Squared vs. Absolute Deviations</i>                                           | 228        |
| 5.2.2    | <i>Hypothesis-Testing in Multiple Regression</i>                                            | 228        |
| 5.2.3    | <i>Multiple Linear Regression in Python</i>                                                 | 229        |
| 5.2.4    | <i>Multiple Linear Regression in R</i>                                                      | 230        |
| 5.3      | <i>Geometry of Least-Squares</i>                                                            | 231        |
| 5.4      | <i>Gauss-Markov Theorem: What We Like about Least-Squares Estimates</i>                     | 232        |
| 5.4.1    | <i>Are Unbiased Estimators Always Best?</i>                                                 | 232        |
| 5.5      | <i>Time Series: An Example of Correlated Errors</i>                                         | 233        |
| 5.6      | <i>Model Selection in Regression: Is There an “Optimal” Model?</i>                          | 233        |
| 5.7      | <i>Effect Size, Model Selection, and Adjusting the Training Error Rate</i>                  | 234        |
| 5.7.1    | <i><math>R^2</math>, Adjusted <math>R^2</math>, <math>C_p</math>, AIC, BIC</i>              | 234        |
| 5.7.2    | <i>Comparing <math>R^2_{adj}</math>, <math>C_p</math>, AIC, and BIC to Cross-Validation</i> | 236        |
| 5.8      | <i>Assumptions for Regression</i>                                                           | 237        |
| 5.8.1    | <i>Collinearity</i>                                                                         | 237        |
| 5.8.1.1  | <i>Variance Inflation Factor</i>                                                            | 237        |
| 5.8.2    | <i>Collinearity Necessarily Implies Redundancy Only in Terms of Variance</i>                | 238        |
| 5.9      | <i>Variable Selection Methods: Building the Regression Model</i>                            | 238        |
| 5.9.1    | <i>Forward, Backward, and Stepwise in R</i>                                                 | 239        |
| 5.10     | <i>Mediated and Moderated Regression Models</i>                                             | 241        |
| 5.10.1   | <i>Statistical Mediation</i>                                                                | 242        |
| 5.10.2   | <i>Statistical Moderation</i>                                                               | 244        |
| 5.10.3   | <i>Moderated Mediation</i>                                                                  | 244        |
| 5.10.4   | <i>Mediation in Python</i>                                                                  | 245        |
| 5.11     | <i>Further Directions and a Final Word of Warning on Mediation and Moderation</i>           | 246        |
| 5.12     | <i>Principal Components Regression</i>                                                      | 247        |
| 5.12.1   | <i>What Is Principal Components Analysis?</i>                                               | 247        |
| 5.12.2   | <i>PCA Regression and Singularity</i>                                                       | 249        |
| 5.12.3   | <i>Principal Components Regression in R</i>                                                 | 249        |
| 5.13     | <i>Partial Least-Squares Regression</i>                                                     | 252        |
| 5.13.1   | <i>Partial Least-Squares in R</i>                                                           | 253        |
| 5.13.2   | <i>Partial Least-Squares in Python</i>                                                      | 259        |
| 5.14     | <i>Multivariate Reduced-Rank Regression</i>                                                 | 259        |
| 5.15     | <i>Canonical Correlation</i>                                                                | 260        |
| 5.15.1   | <i>Canonical Correlation in R</i>                                                           | 261        |
| 5.16     | <i>Chapter Review Exercises</i>                                                             | 264        |

|          |                                                                                        |            |
|----------|----------------------------------------------------------------------------------------|------------|
| <b>6</b> | <b>Regularization Methods in Regression: Ridge, Lasso, and Elastic Net</b>             | <b>265</b> |
| 6.1      | <i>The Concept of Regularization</i>                                                   | 265        |
| 6.1.1    | <i>Regularization in Regression and Beyond</i>                                         | 267        |
| 6.2      | <i>Ridge Regression</i>                                                                | 267        |
| 6.2.1    | <i>Mathematics of Ridge Regression</i>                                                 | 268        |
| 6.2.2    | <i>Consequence of Ridge Estimator</i>                                                  | 269        |
| 6.2.3    | <i>Revisiting the Bias-Variance Tradeoff: Why Ridge Is Useful</i>                      | 269        |
| 6.2.4    | <i>A Visual Look at Ridge Regression</i>                                               | 270        |
| 6.2.5    | <i>Ridge Regression in Python</i>                                                      | 271        |
| 6.2.6    | <i>Ridge Regression in R</i>                                                           | 273        |
| 6.3      | <i>Lasso Regression</i>                                                                | 276        |
| 6.3.1    | <i>Lasso Regression in Python</i>                                                      | 277        |
| 6.3.2    | <i>Lasso Regression in R</i>                                                           | 278        |
| 6.4      | <i>Elastic Net</i>                                                                     | 282        |
| 6.4.1    | <i>Elastic Net in Python</i>                                                           | 282        |
| 6.5      | <i>Which Regularization Penalty Is Better?</i>                                         | 284        |
| 6.6      | <i>Least-Angle Regression</i>                                                          | 284        |
| 6.6.1    | <i>Least-Angle Regression in R</i>                                                     | 285        |
| 6.7      | <i>Additional Variable Selection Algorithms</i>                                        | 288        |
| 6.8      | <i>Chapter Review Exercises</i>                                                        | 289        |
| <br>     |                                                                                        |            |
| <b>7</b> | <b>Nonlinear and Nonparametric Regression</b>                                          | <b>290</b> |
| 7.1      | <i>Polynomial Regression</i>                                                           | 291        |
| 7.1.1    | <i>Polynomial Regression in Python</i>                                                 | 292        |
| 7.1.2    | <i>Polynomial Regression in R</i>                                                      | 293        |
| 7.1.3    | <i>Polynomial Regression as a Global Strategy</i>                                      | 295        |
| 7.1.4    | <i>A More Local Alternative</i>                                                        | 295        |
| 7.1.5    | <i>Least-Squares Regression Line as a “Floating Mean”: Toward a Localized Approach</i> | 295        |
| 7.1.5.1  | <i>Zooming in on Locality</i>                                                          | 296        |
| 7.2      | <i>Basis Functions and Expansions</i>                                                  | 297        |
| 7.2.1    | <i>Basis Functions and Locality</i>                                                    | 298        |
| 7.2.2    | <i>Neural Networks as a Basis Expansion: Generalizing the Concept</i>                  | 298        |
| 7.2.3    | <i>Regression Splines and the Concept of a “Knot”</i>                                  | 299        |
| 7.2.4    | <i>Conceptualizing Regression Splines</i>                                              | 301        |
| 7.2.5    | <i>Problem with Splines and Imposing Constraints</i>                                   | 301        |
| 7.2.6    | <i>Polynomial Regression vs. Regression Splines</i>                                    | 302        |
| 7.3      | <i>Nonparametric Regression: Local and Kernel Regression</i>                           | 303        |
| 7.3.1    | <i>Motivating Kernel Regression via Local-Averaging</i>                                | 303        |
| 7.3.2    | <i>Kernel Regression – “Locally Weighted Averaging”</i>                                | 304        |
| 7.3.3    | <i>A Variety of Kernels</i>                                                            | 305        |
| 7.3.4    | <i>Kernel Regression Is Not Nonlinear; It Is Nonparametric</i>                         | 306        |
| 7.3.5    | <i>Kernel Regression in R</i>                                                          | 306        |
| 7.4      | <i>Chapter Review Exercises</i>                                                        | 307        |

|          |                                                                                               |            |
|----------|-----------------------------------------------------------------------------------------------|------------|
| <b>8</b> | <b>Generalized Linear and Additive Models: Logistic and Related Models</b>                    | <b>308</b> |
| 8.1      | <i>How to Operationalize the Response</i>                                                     | 308        |
| 8.1.1    | <i>Pros and Cons of Binning</i>                                                               | 309        |
| 8.1.2    | <i>Detecting New Classes or Categories</i>                                                    | 309        |
| 8.2      | <i>The Generalized Linear Model</i>                                                           | 309        |
| 8.2.1    | <i>Intrinsically Linear Models</i>                                                            | 310        |
| 8.2.2    | <i>General vs. Generalized Linear Models</i>                                                  | 310        |
| 8.3      | <i>The Logistic Regression Model</i>                                                          | 311        |
| 8.3.1    | <i>Odds and Odds Ratios</i>                                                                   | 312        |
| 8.3.2    | <i>Logistic Regression in R</i>                                                               | 312        |
| 8.3.3    | <i>Logistic Regression in Python</i>                                                          | 313        |
| 8.4      | <i>Generalized Linear Models and Neural Networks</i>                                          | 314        |
| 8.5      | <i>Multiple Logistic Regression</i>                                                           | 315        |
| 8.5.1    | <i>Multiple Logistic Regression in R</i>                                                      | 316        |
| 8.6      | <i>Poisson Regression</i>                                                                     | 318        |
| 8.6.1    | <i>Poisson Regression in R</i>                                                                | 318        |
| 8.6.2    | <i>Poisson Regression in Python</i>                                                           | 321        |
| 8.7      | <i>Generalized Additive Models: A Flexible Nonparametric Alternative</i>                      | 321        |
| 8.7.1    | <i>Why Use a Smoother Instead of Linear Weights?</i>                                          | 322        |
| 8.7.2    | <i>Deriving the Generalized Additive Model</i>                                                | 322        |
| 8.7.3    | <i>GAM as a Smooth Extension to GLLM</i>                                                      | 323        |
| 8.7.4    | <i>Generalized Additive Models and Neural Networks</i>                                        | 324        |
| 8.7.5    | <i>Linking the Logit to the Additive Logistic Model</i>                                       | 324        |
| 8.8      | <i>Overview and Recap of Nonlinear Approaches for Nonlinear Responses</i>                     | 325        |
| 8.9      | <i>Discriminant Analysis</i>                                                                  | 327        |
| 8.9.1    | <i>Bayes Is Best for Classification</i>                                                       | 328        |
| 8.9.2    | <i>Why Not Always Bayes?</i>                                                                  | 329        |
| 8.9.3    | <i>Linear Discriminant Analysis Model</i>                                                     | 330        |
| 8.9.4    | <i>How LDA Approximates Bayes</i>                                                             | 330        |
| 8.9.5    | <i>Estimating the Prior Probability and Parameters of the Discriminant Function</i>           | 331        |
| 8.10     | <i>Multiclass Discriminant Analysis</i>                                                       | 332        |
| 8.11     | <i>Discriminant Analysis and Dimensionality in a Simple Scatterplot</i>                       | 333        |
| 8.12     | <i>Quadratic Discriminant Analysis</i>                                                        | 335        |
| 8.13     | <i>Regularized Discriminant Analysis</i>                                                      | 336        |
| 8.14     | <i>Discriminant Analysis in R</i>                                                             | 337        |
| 8.15     | <i>Discriminant Analysis in Python</i>                                                        | 339        |
| 8.16     | <i>Naïve Bayes: Approximating the Bayes Classifier by Assuming (Conditional) Independence</i> | 339        |
| 8.16.1   | <i>What Makes Naïve Bayes “Naïve?”</i>                                                        | 340        |
| 8.16.2   | <i>Naïve Bayes in Python</i>                                                                  | 343        |
| 8.17     | <i>LDA, QDA, or Naïve Bayes: Which Is Best?</i>                                               | 344        |
| 8.18     | <i>Nonparametric K-Nearest Neighbors</i>                                                      | 344        |
| 8.18.1   | <i>K-Nearest Neighbors (KNN): Only Looking at Nearby Points</i>                               | 344        |
| 8.18.2   | <i>Example of KNN</i>                                                                         | 345        |

|           |                                                                                        |            |
|-----------|----------------------------------------------------------------------------------------|------------|
| 8.18.3    | <i>Disadvantages of KNN</i>                                                            | 348        |
| 8.19      | <i>Chapter Review Exercises</i>                                                        | 349        |
| <b>9</b>  | <b>Support Vector Machines</b>                                                         | <b>350</b> |
| 9.1       | <i>Maximal Margin Classifier</i>                                                       | 350        |
| 9.1.1     | <i>When Sum Does Not Equal Zero</i>                                                    | 351        |
| 9.1.1.1   | <i>So, What's the Problem?</i>                                                         | 352        |
| 9.1.2     | <i>Building the Maximal Margin Classifier</i>                                          | 356        |
| 9.2       | <i>The Case of No Separating Hyperplane</i>                                            | 357        |
| 9.3       | <i>Support Vector Classifier for the Non-Separable Case</i>                            | 358        |
| 9.4       | <i>Support Vector Machines: Introducing the Kernel for Nonlinearity</i>                | 360        |
| 9.4.1     | <i>Enlarging the Feature Space with Kernels</i>                                        | 362        |
| 9.4.2     | <i>Support Vector Machines in Python</i>                                               | 363        |
| 9.4.3     | <i>Support Vector Machines in R</i>                                                    | 364        |
| 9.5       | <i>Chapter Review Exercises</i>                                                        | 367        |
| <b>10</b> | <b>Decision Trees, Bagging, Random Forests and Committee Machines</b>                  | <b>368</b> |
| 10.1      | <i>Why Combining Weak Learners Works: Concept of Variance Reduction Using Averages</i> | 368        |
| 10.2      | <i>Decision Trees</i>                                                                  | 370        |
| 10.2.1    | <i>How Should Trees Be Grown?</i>                                                      | 371        |
| 10.2.2    | <i>Optimization Criteria for Tree-Building</i>                                         | 371        |
| 10.2.3    | <i>Why Not Multiway Splits?</i>                                                        | 372        |
| 10.2.4    | <i>Overfitting, Saturation, and Tree Pruning</i>                                       | 372        |
| 10.2.5    | <i>Cost Complexity or Weakest Link Pruning</i>                                         | 373        |
| 10.3      | <i>Classification Trees</i>                                                            | 374        |
| 10.3.1    | <i>Gini Index</i>                                                                      | 374        |
| 10.3.2    | <i>Decision Trees in R</i>                                                             | 375        |
| 10.4      | <i>Committee Machines</i>                                                              | 379        |
| 10.5      | <i>Overview of Bagging and Boosting</i>                                                | 379        |
| 10.5.1    | <i>Bagging</i>                                                                         | 380        |
| 10.5.2    | <i>A Familiar Example: Bagging Samples and the Variance of the Mean</i>                | 380        |
| 10.5.3    | <i>A Deeper Look at Bagging</i>                                                        | 381        |
| 10.5.4    | <i>Out-of-Bag Error</i>                                                                | 382        |
| 10.5.5    | <i>Interpreting Results from Bagging</i>                                               | 382        |
| 10.5.6    | <i>Bagging in R</i>                                                                    | 382        |
| 10.6      | <i>Random Forests</i>                                                                  | 384        |
| 10.6.1    | <i>The Problem with Bagging Decision Trees</i>                                         | 384        |
| 10.6.2    | <i>Equivalency of Random Forests and Bagging</i>                                       | 385        |
| 10.6.3    | <i>Random Forests in R</i>                                                             | 385        |
| 10.7      | <i>Boosting</i>                                                                        | 388        |
| 10.7.1    | <i>Boosting Using R</i>                                                                | 389        |
| 10.8      | <i>Chapter Review Exercises</i>                                                        | 393        |

## 11 Principal Components Analysis, Blind Source Separation, and Manifold Learning 394

- 11.1 *Dimension Reduction and Jargon* 395
- 11.2 *Deriving Classic Principal Components Analysis* 396
  - 11.2.1 *The Second Principal Component* 398
  - 11.2.2 *PCA as a Least-Squares Technique and Minimizing Reconstruction Error* 398
  - 11.2.3 *Choosing the Number of Derived Components* 400
  - 11.2.4 *Why Reconstruction Error Is Insufficient for Choosing Number of Components* 400
  - 11.2.5 *Constraints on Components* 400
  - 11.2.6 *Centering Observed Variables* 401
  - 11.2.7 *Orthogonality of Components* 402
  - 11.2.8 *Proportion of Variance Explained by Each Component: Covariance vs. Correlation Matrices* 402
  - 11.2.9 *Principal Components as a Rotation of Axes* 403
  - 11.2.10 *Principal Components, Discriminant Functions, Canonical Variates: Linking Foundations* 405
  - 11.2.11 *Principal Components in Python* 406
  - 11.2.12 *Principal Components in R* 407
  - 11.2.13 *Cautionary Concerns and Caveats Regarding Principal Components* 409
- 11.3 *Independent Components Analysis* 410
  - 11.3.1 *Principal Components vs. Independent Components Analysis* 411
- 11.4 *Probabilistic PCA* 414
  - 11.4.1 *Motivation for Probabilistic PCA* 414
  - 11.4.2 *Probabilistic PCA in R* 415
- 11.5 *PCA for Discrete, Binary, and Categorical Data* 417
- 11.6 *Nonlinear Dimension Reduction* 418
  - 11.6.1 *Kernel PCA* 418
  - 11.6.2 *How KPCA Works* 418
  - 11.6.3 *Kernelizing and Computational Complexity* 420
  - 11.6.4 *Reconstruction Error in Kernel PCA* 420
  - 11.6.5 *The Matrices of Kernel PCA* 421
  - 11.6.6 *Classical PCA as a Special Case of Kernel PCA* 422
  - 11.6.7 *Kernel Trick Is Not Simply about Cost* 422
  - 11.6.8 *Kernel PCA in Python* 422
  - 11.6.9 *Kernel PCA in R* 423
- 11.7 *Principal Curves* 425
  - 11.7.1 *Principal Components as a Special Case of Principal Curves and Surfaces* 426
  - 11.7.2 *Principal Curves in R* 426
- 11.8 *Principal Components Analysis as an Encoder* 427
- 11.9 *Neural Networks and PCA as Autoencoders* 428
- 11.10 *Multidimensional Scaling* 429
  - 11.10.1 *Merits of MDS* 432

|         |                                               |     |
|---------|-----------------------------------------------|-----|
| 11.10.2 | Metric vs. Nonmetric (Ordinal) Scaling        | 434 |
| 11.10.3 | Weakness of MDS: “Closeness” Can Be Arbitrary | 434 |
| 11.10.4 | Standardization of Distances                  | 434 |
| 11.10.5 | MDS in Python                                 | 435 |
| 11.10.6 | MDS in R                                      | 437 |
| 11.11   | Self-Organizing Maps                          | 439 |
| 11.12   | Manifold Learning                             | 441 |
| 11.12.1 | Manifold Hypothesis                           | 442 |
| 11.12.2 | Example of a Simple Manifold                  | 443 |
| 11.12.3 | Nonparametric Manifolds                       | 443 |
| 11.12.4 | Geodesic Distances                            | 443 |
| 11.13   | Locally Linear Embedding                      | 444 |
| 11.13.1 | LLE in Python                                 | 445 |
| 11.14   | Isomap                                        | 447 |
| 11.14.1 | Isomap in Python                              | 448 |
| 11.15   | Stochastic Neighborhood Embedding             | 449 |
| 11.15.1 | SNE in R                                      | 452 |
| 11.16   | t-SNE                                         | 453 |
| 11.16.1 | Performance of t-SNE to Other Techniques      | 454 |
| 11.16.2 | t-SNE in Python                               | 454 |
| 11.16.3 | t-SNE in R                                    | 456 |
| 11.17   | Manifold Learning and Beyond                  | 458 |
| 11.18   | Chapter Review Exercises                      | 459 |

## 12 Exploratory Factor Analysis

460

|        |                                                  |     |
|--------|--------------------------------------------------|-----|
| 12.1   | Why Treat Factor Analysis in Its Own Chapter?    | 461 |
| 12.2   | Common Orthogonal Factor Model                   | 461 |
| 12.2.1 | Factor Analysis Is a Regression Model            | 462 |
| 12.2.2 | Assumptions Underlying the Factor Analysis Model | 462 |
| 12.2.3 | Implied Covariance Matrix                        | 462 |
| 12.3   | The Problem with Factor Analysis                 | 463 |
| 12.3.1 | The Problem Is the Users, Not the Method         | 464 |
| 12.3.2 | Factor Analysis Generalizes to Machine Learning  | 464 |
| 12.4   | Factor Estimation                                | 465 |
| 12.4.1 | Principal Factor (Principal Axis Factoring)      | 465 |
| 12.4.2 | Maximum Likelihood                               | 466 |
| 12.5   | Factor Rotation                                  | 466 |
| 12.5.1 | Varimax Rotation                                 | 467 |
| 12.5.2 | Quartimax                                        | 468 |
| 12.6   | Bartlett’s Test of Sphericity                    | 468 |
| 12.6.1 | Factor Analysis in Python                        | 469 |
| 12.6.2 | Factor Analysis in R                             | 470 |
| 12.7   | Independent Factor Analysis                      | 473 |
| 12.8   | Nonlinear Factor Analysis and Autoencoders       | 474 |



|           |                                                                                      |            |
|-----------|--------------------------------------------------------------------------------------|------------|
| 12.8.1    | <i>Unpacking the Autoencoder</i>                                                     | 474        |
| 12.8.2    | <i>Factor Analysis as a Neural Network</i>                                           | 474        |
| 12.9      | <i>Probabilistic “Sensible” PCA (Again)</i>                                          | 475        |
| 12.10     | <i>Mixtures of Factor Analysis: Modeling Local Linearity</i>                         | 475        |
| 12.11     | <i>Item Factor Analysis</i>                                                          | 476        |
| 12.12     | <i>Sparse Factor Analysis</i>                                                        | 477        |
| 12.13     | <i>Chapter Review Exercises</i>                                                      | 477        |
| <b>13</b> | <b>Confirmatory Factor Analysis, Path Analysis, and Structural Equation Modeling</b> | <b>479</b> |
| 13.1      | <i>What Makes a Model “Exploratory” vs. “Confirmatory”?</i>                          | 479        |
| 13.2      | <i>Why “Causal Modeling” Is Not Causal at All</i>                                    | 480        |
| 13.2.1    | <i>Misguided History</i>                                                             | 480        |
| 13.2.2    | <i>Baron and Kenny (1986)</i>                                                        | 481        |
| 13.3      | <i>Is the Variable Measurable? The Observed vs. Unobserved Distinction</i>           | 482        |
| 13.4      | <i>Path Analysis: Extending Regression and Previewing SEM</i>                        | 483        |
| 13.4.1    | <i>Exogenous vs. Endogenous Variables</i>                                            | 484        |
| 13.5      | <i>Confirmatory Factor Analysis Model</i>                                            | 484        |
| 13.6      | <i>Structural Equation Models</i>                                                    | 485        |
| 13.6.1    | <i>Covariance Modeling</i>                                                           | 486        |
| 13.6.2    | <i>Evaluating Model Fit</i>                                                          | 487        |
| 13.6.3    | <i>Overall (Absolute) Measures</i>                                                   | 488        |
| 13.6.4    | <i>Incremental Fit Indices</i>                                                       | 489        |
| 13.7      | <i>Structural Equation Modeling with Nonlinear Effects</i>                           | 491        |
| 13.7.1    | <i>Example of a Nonlinear SEM</i>                                                    | 491        |
| 13.7.2    | <i>Structural Equation Nonparametric and Semiparametric Mixture Models</i>           | 492        |
| 13.8      | <i>Caveats Regarding SEM Models</i>                                                  | 494        |
| 13.9      | <i>SEM Models in R</i>                                                               | 495        |
| 13.10     | <i>Chapter Review Exercises</i>                                                      | 497        |
| <b>14</b> | <b>Cluster Analysis and Data Segmentation</b>                                        | <b>498</b> |
| 14.1      | <i>Cluster Paradigms and Classifications</i>                                         | 498        |
| 14.2      | <i>Are Clusters Meaningful?</i>                                                      | 499        |
| 14.3      | <i>Dissimilarity Metrics: The Basis of Clustering Algorithms</i>                     | 500        |
| 14.4      | <i>Association Rules: Market Basket Analysis</i>                                     | 501        |
| 14.5      | <i>Why Not Consider All Groups?</i>                                                  | 501        |
| 14.5.1    | <i>What Makes a “Good” Clustering Algorithm?</i>                                     | 502        |
| 14.6      | <i>Distance and Proximity Metrics</i>                                                | 503        |
| 14.7      | <i>Is the Data Clusterable?</i>                                                      | 505        |
| 14.8      | <i>Algorithms for Cluster Analysis</i>                                               | 506        |
| 14.8.1    | <i>K-Means Clustering and K-Representatives</i>                                      | 506        |
| 14.8.2    | <i>How K-Means Works</i>                                                             | 507        |
| 14.8.3    | <i>Defining Proximity for K-Means</i>                                                | 507        |
| 14.8.4    | <i>Setting K in K-Means</i>                                                          | 508        |

|           |                                                                           |            |
|-----------|---------------------------------------------------------------------------|------------|
| 14.8.5    | <i>Weakness of K-Means</i>                                                | 509        |
| 14.8.6    | <i>K-Means vs. ANOVA vs. Discriminant Analysis</i>                        | 509        |
| 14.8.7    | <i>Making K-Means Probabilistic via K-Means ++</i>                        | 510        |
| 14.8.8    | <i>Using the Data: K-Medoids Clustering</i>                               | 510        |
| 14.8.9    | <i>K-Means in Python</i>                                                  | 511        |
| 14.8.10   | <i>K-Means in R</i>                                                       | 513        |
| 14.9      | <i>Sparse and Longitudinal K-Means</i>                                    | 516        |
| 14.10     | <i>Hierarchical Clustering</i>                                            | 517        |
| 14.10.1   | <i>Agglomerative Clustering in Python</i>                                 | 518        |
| 14.10.2   | <i>Agglomerative Clustering in R</i>                                      | 519        |
| 14.11     | <i>Density-Based Clustering: DBSCAN</i>                                   | 522        |
| 14.11.1   | <i>Dense Points and Crowded Regions</i>                                   | 522        |
| 14.11.2   | <i>DBSCAN in R</i>                                                        | 523        |
| 14.12     | <i>Clustering via Mixture Models</i>                                      | 525        |
| 14.12.1   | <i>Model Selection for Clustering Solutions</i>                           | 527        |
| 14.13     | <i>Cluster Validation</i>                                                 | 528        |
| 14.14     | <i>Cluster Analysis and Beyond</i>                                        | 529        |
| 14.15     | <i>Chapter Review Exercises</i>                                           | 529        |
| <b>15</b> | <b>Artificial Neural Networks and Deep Learning</b>                       | <b>531</b> |
| 15.1      | <i>The Rise of Neural Networks: Original Motivation</i>                   | 531        |
| 15.2      | <i>Rosenblatt's Perceptron</i>                                            | 534        |
| 15.3      | <i>Big Picture Overview of Machine Learning and Neural Networks</i>       | 535        |
| 15.3.1    | <i>What Is a Neural Network? Minimizing the Hype</i>                      | 536        |
| 15.3.2    | <i>Neural Networks Are Composite Functions</i>                            | 536        |
| 15.4      | <i>Single-Layer Feedforward Neural Network</i>                            | 538        |
| 15.5      | <i>What Is an Activation Function?</i>                                    | 540        |
| 15.5.1    | <i>Activation Functions Do Not "Activate" Anything</i>                    | 541        |
| 15.5.2    | <i>Types of Activation Functions</i>                                      | 541        |
| 15.5.3    | <i>Saturating vs. Non-Saturating Activation Functions</i>                 | 542        |
| 15.5.3.1  | <i>The Problem with ReLU</i>                                              | 544        |
| 15.5.3.2  | <i>LeakyReLU</i>                                                          | 545        |
| 15.5.4    | <i>Which Activation Function to Use?</i>                                  | 546        |
| 15.6      | <i>The Multilayer Perceptron – A Deeper Look at Neural Networks</i>       | 546        |
| 15.7      | <i>Training Neural Networks</i>                                           | 548        |
| 15.7.1    | <i>Backpropagation and Minimizing Error Sums of Squares</i>               | 548        |
| 15.8      | <i>How Many Hidden Nodes and Layers to Include?</i>                       | 550        |
| 15.9      | <i>Overfitting in Neural Networks</i>                                     | 551        |
| 15.9.1    | <i>Early Stopping</i>                                                     | 551        |
| 15.9.2    | <i>Dropout Method</i>                                                     | 551        |
| 15.9.3    | <i>Regularized Network</i>                                                | 552        |
| 15.10     | <i>Types of Networks</i>                                                  | 552        |
| 15.11     | <i>The Universal Approximation Theorem: The Appeal of Neural Networks</i> | 553        |
| 15.11.1   | <i>Visualizing the Universal Approximation Theorem</i>                    | 554        |

|                                                                                                        |                |
|--------------------------------------------------------------------------------------------------------|----------------|
| <i>15.12 Neural Networks and Projection Pursuit</i>                                                    | <i>555</i>     |
| <i>15.12.1 Projection Pursuit Regression and Relation to Neural Networks</i>                           | <i>556</i>     |
| <i>15.13 Summary, Warnings, and Caveats of Neural Networks</i>                                         | <i>558</i>     |
| <i>15.13.1 Neural Networks Are Not Biological Nor Are They Machines</i>                                | <i>559</i>     |
| <i>15.13.2 Neural Networks Need Not Be Nonlinear</i>                                                   | <i>559</i>     |
| <i>15.13.3 Neural Networks Are (Mostly) Nonparametric</i>                                              | <i>559</i>     |
| <i>15.13.4 Including More Layers Leads to Increased Approximation on the<br/>        Training Data</i> | <i>560</i>     |
| <i>15.14 Neural Networks in Python</i>                                                                 | <i>560</i>     |
| <i>15.15 Neural Networks in R</i>                                                                      | <i>561</i>     |
| <i>15.16 Chapter Review Exercises</i>                                                                  | <i>564</i>     |
| <br><i>Concluding Remarks</i>                                                                          | <br><i>565</i> |
| <i>References</i>                                                                                      | <i>566</i>     |
| <i>Index</i>                                                                                           | <i>580</i>     |

# Preface

This book began with sketchy lecture notes prepared for my statistics classes over the period 2018–23 and was largely completed during a sabbatical taken over the year 2023–24 at the University of Montana.

# Acknowledgments

I have many people to thank, far too numerous to mention individually, including all students in the classes I have taught for just over twenty years since I began at the University of Montana in the Spring of 2004. This book could also not have been written without the influence and readings of excellent authors on these topics, again far too numerous to mention. From that day Hays introduced me to set theory to now, I have been blessed by their insights, knowledge, and wisdom. To all the students and scholars who have taught me so much, I thank you.

Thank you as well to all at Routledge for producing this book. Writing a book like this is very challenging. *Finishing* a book like this is exponentially even more difficult. I am thankful to both Adam Woods for originally taking on this project, and to Georgia Thompson for later taking it over. The elusive finish line continually seemed to get farther away the more I tried to finish the book, and I am most thankful to Georgia for understanding and tolerating the multiple missed deadlines required in completing this writing project. Convergence toward a deadline coupled with an increasingly devoted commitment to quality can, in the limit, make meeting deadlines virtually impossible. *All books must end so that new stories can begin.*

Last but certainly not least, and since 2018 always coming first, I thank my beloved dog, Odie, who passed away in the midst of this book being written, but not before blessing me with the greatest year as we together hiked the snowy mountains of Montana in idyllic solitude, serenity, and perfect balance. Magical moments in the sun and the snow. Forever in my heart, thank you Odie, *thank you.*

*To the Reader*

*This book was written with every effort to ensure its contents are precise and rigorous. Please contact me at [daniel.denis@umontana.edu](mailto:daniel.denis@umontana.edu) to report any errata or any general feedback you may have about the book. We will consider your note and possibly include a revision in a future printing or edition, or include a comment/correction in the online errata file. For updated errata, please visit [www.datapsyc.com](http://www.datapsyc.com).*

*Daniel J. Denis*  
October, 2025

## **PART I**

# **Preliminaries and Foundations**



**Taylor & Francis**

Taylor & Francis Group

<http://taylorandfrancis.com>

# Introduction, Motivation, Pedagogy, and Ideas About Learning

Everything has changed, yet at the same time so much has remained the same. In the past, aspiring scientists, whether in the natural or social sciences, would specialize in their discipline while at the same time taking a number of undergraduate and graduate courses in statistics and research methods ranging from theoretical to applied depending on the flavor of the course and the preference of the instructor. In addition to the typical introductory and usually relatively superficial statistics course that virtually every undergraduate must enroll regardless of specialization (e.g., biology, psychology, business, etc.), advanced undergraduate and graduate students are typically required to take a one-half semester of studying foundations (e.g., probability, hypothesis-testing, etc.) leading up to the analysis of variance (ANOVA) while the second half of the year would be typically spent studying more advanced ANOVA models and simple linear and multiple regression as well as possibly including a survey of the generalized linear model (e.g., logistic and Poisson regression). Such exposure would provide the student with a basic understanding of *what's going on* in applied statistics sufficient to get their foot in the door to conducting and interpreting their own analyses for theses and dissertations, or at minimum and probably more realistically, be able to converse with someone (e.g., supervisor) able to *guide* them in conducting and interpreting their own analyses. After such a year-long sequence, a multivariate analysis course might also be offered that would feature topics such as *multivariate analysis of variance (MANOVA)*, *discriminant analysis*, *principal components analysis (PCA)*, *factor analysis*, among others, including possibly *path analysis* and *structural equation modeling* for students in the social and allied sciences. Much of the above coursework would feature, to varying extents, the use of software, such as Statistical Package for the Social Sciences (SPSS), Statistical Analysis System (SAS), and more modernly, R, and possibly even Python.

## 0.1 The Paradigm Shift: What Has Changed

What has been described above as the typical course of study for the advanced undergraduate or beginning graduate student is now *out of date*. *It is no longer enough*. *It is grossly incomplete*. *It is archaic*. You get the picture! In the last 20–30 years, certainly post-2000, a machine learning and data science *Kuhnian paradigm shift* has occurred. In the age we find ourselves in, applied statistics courses are desperately in need of a *makeover*. They are in need of a major “updating,” overhaul, and renovation, not necessarily because foundational elements of modern data science and machine learning differ *that much* from the traditional univariate and multivariate essentials presented in the typical science curriculum, but mostly because the very landscape of what we may more generally call *data analysis* has changed. What is called “data analysis” today is much broader than prior to the data science and machine learning revolution, yet at the same time the foundations of these areas are often found in statistics. For the applied scientist or practitioner,



## 4 Multivariate Statistics and Machine Learning

exposure to traditional courses in statistics that do not incorporate data science and machine learning theory and concepts are now, *incomplete*.

### 0.1.1 A Wide Divide

Consider the current reality – it is entirely possible for a student to graduate from an applied science program having taken a typical statistics sequence yet upon graduating be completely unfamiliar with what is going on in *machine learning* and having little to no idea what *artificial intelligence* is despite the fact that many of the concepts and features learned in their statistics courses are directly applicable to what is occurring in these fields. Indeed, many of the concepts studied in statistics can be considered *foundations* to these modern fields. Hence, the student from a statistics course sitting down to lunch with a student from a machine learning course would soon realize that many of the things they are learning in their respective courses overlap a great deal, and in many cases, may even be very much the *same*. They begin to share notes and compare jargon used from each field and quickly realize the extensive *overlap* of their coursework. They are presumably coming from different disciplines yet may be talking very much about the same things! If they fail to connect and realize this, it may be more to do with the *jargon* used in each field than anything else.

Now, the above is not to suggest that jargon alone is what differentiates these fields. *Far from it*. Machine learning, and its underachieving and overpromising parent *artificial intelligence*, are their own respective fields, and as we will see in this book have already made tremendous advances and contributions for disciplines that are still relatively quite young. However, the overlap, especially at the level of *foundations*, is undeniable. Depending on one's historical and philosophical perspective, one might even say that in many cases statistics gave *birth* to these fields. When a student or even researcher in machine learning speaks of a “learner” to a student just coming from a statistics course as though it is a “new” concept, the reaction of the statistics or science student should not be one of confusion and ignorance. Instead, it should be one of *recognition* of the similar foundations learned in their respective courses. Of course, the student of statistics, coming from an introductory lecture on the analysis of variance, for example, has just been subjected to a *statistical learner* as well, and it would be a shame if that student were unable to identify with the concept brought to the lunch table by the machine learning student. Indeed, those seemingly simple and “basic” *t*-tests were kind of “machine learners” themselves in their own right and in the most general sense. Translated to a dummy-coded regression, weights are optimized, and if we feed more data through the *t*-test or corresponding regression “algorithm,” those scalars get optimized further. *That simple model is learning based on experience*. The statistics student needs to appreciate what the machine learning student is talking about and vice versa, the machine learning student would do well to get in touch with some of his or her own roots in statistical theory.

## 0.2 A Unified Vision: The Bridge

Though machine learning and artificial intelligence are disparate fields, it is high time for a *unified introduction and top-down overview* of these areas for the student of the social and natural sciences. It is well time to construct a *bridge*. For *specialists* in statistics or machine learning the awareness of these bridges is (or should be) old news. But to the newcomer or applied science student, without a roadmap it can be exceedingly difficult to know or even recognize the *landscape*. You may not even know what forest you are in, which makes it terribly difficult to even recognize or navigate the trees.

A unified treatment of these areas should include a strong introduction and overview of topics in *univariate, bivariate, multivariate, machine learning, design and analysis of experiments*, along with select topics from *measurement theory*. To this last point, before one analyzes data or designs

the next fancy algorithm, one should be intimately interested and invested into what *one is actually measuring*. This is true whether one is measuring emotional intelligence or one is measuring product preference via an online retailer. The customer may have “clicked” on product *A* because they *thought* they preferred it over product *B*, until it arrived on delivery day and they realized what they actually preferred was the *image* it falsely represented on the computer screen. In the end, their preference was for the *image*, the *perception*, the *idealization*, the *feeling*, not the *product*, yet the data point is advertised to have recorded *preference for the product*. *The measurement mechanism has failed*. Hence, what was measured in that impressive terabyte SQL database was not what we originally thought. It isn’t communicating what you think it is. Measurement issues abound in science and research, and a good course in statistics and machine learning should address them to some extent regardless of whether that course is for social or natural scientists. *Measurement is the first principle of good science*. Methods to analyze those measurements, of which much of this book is about, are second to that top priority.

What is more, modern texts on statistical and machine learning often tout the discipline as somewhat “novel” even though, of course, it overlaps a great deal with traditional and *classical* statistics in both univariate and multivariate domains. To reiterate, a presentation of machine learning that ignores this foundation or goes “skimpy” on the statistical side of things makes it possible for the graduate student to take a computer science course, for instance, yet fail to realize that many of those same foundational elements can also be found, and often have their *origins* in, *classical statistics*. For example, “unsupervised learning” in the name of techniques such as principal components analysis sounds “new,” but actually had its origins in 1901 with Karl Pearson. It’s a *learning algorithm* now, whereas it used to be simply a *statistical technique*. Oh my, have times changed! Karl Pearson would look befuddled had we told him his minimizing of squared perpendicular distances was a “learning algorithm.” Education needs to emphasize *history*, where these things came from and how they morphed into the verbiage and jargon of today. Very little is truly “new.” Most is usually a mixture of historical elements and ideas.

### 0.3 The Data Science and Machine Learning Invasion: Questions and Answers

Data science, machine learning, artificial intelligence have all invaded society with more pandemonium than even the Beatles in the 1960s. Even for those well-versed in traditional statistics, today’s climate of new topics can make even the experienced professional feel like a novice in the sea of new quantitative language. Questions that regularly arise from students and researchers (and the internet) confronting these areas for the first time include – *What exactly is machine learning? What’s the “machine” in machine learning? Is it basically the same as statistics? If not, how exactly is it different? Where does “Big Data” fit into all this? What is data science? Isn’t data science simply statistics with computation? How is data science the same or different from computational statistics? Is data science a fad that will soon pass?*

The above are only a few of the questions that students and researchers may have about this new quantitative climate and how to incorporate all of these ideas into their work and general understanding of what denotes the larger umbrella of *data analysis* these days. Or at minimum, be able to come across articles that feature these kinds of things and not feel like an outsider as though one were reading theoretical physics for the first time. The book you hold in your hands aims to reveal not only the inner workings of these methods to some extent (and albeit at a *relatively* rather shallow and superficial level, all considered), but also to provide a sense of the “big picture” of *what’s going on* in today’s general quantitative climate.

*The biggest picture, seeing all else from the top of the mountain, provides the greatest perspective.* The infinite set has infinitely many subsets and so the learner should aim to understand the

largest structures possible. This can be a painful process! As an analogy, jazz is inordinately complex, that is, until one sees the bigger picture. Then all the “special case” theory are just details, difficult details, but details nonetheless. The elements that make up the wider vision are important, but seeing the overall framework is vital to understanding and appreciating those smaller pieces.

#### 0.4 Who Should Read This Book?

This book was written for a course in multivariate statistics *and* machine learning for the scientist or practitioner. It was also written for specialists in the area who may benefit from some of the teaching and insight regarding how topics are presented and linked, the “undertones” that exist connecting the various methodologies that the reader may not have been aware of when originally studying and learning these techniques for the first time. The book is a “*big picture*” perspective on these fields. If the applied statistics course for scientists is being taught in room one and the machine learning course in room two, this book is the hallway in between.

##### 0.4.1 Textbook Limbo

Having been an instructor of a multivariate course for the better part of twenty years, I have become frustrated in trying to find the ideal resource that reflects today’s mix of *multivariate*, *machine learning*, and *data science* more generally. Even within the realm of traditional multivariate texts, some books feature a discussion of factor analysis and latent variable models while others do not. Many traditional texts on multivariate analysis are quite rigorous in their development of say, *principal components analysis*, yet leave out discussion of these important factor-analytic and latent variable models of which entire books and monographs have been written in both applied statistics and psychometrics. On the other hand, modern machine learning texts are virtually obligated to discuss factor-analytical models under the more general rubric of *manifold learning*. While it is not irregular for instructors to use several different books in teaching a course, it can be difficult for introductory courses to suddenly switch gears from one text to another simply because one text omits an important and fundamental methodology. Personally, I have always found it virtually impossible to find the “right text” for my more advanced multivariate courses.

The current text, perhaps unrealistically, attempts to “cover it all” to some extent, so that upon completion of the book the student will have their *foot in the door* into this “modern empire” of statistics, machine learning, data science, etc. After studying this book, the reader should have a fair idea of *what’s going on* in multivariate statistics *and* machine learning and be able to hopefully use this book as an entry point into the literature. The book can also be used as a reference or refresher for researchers already somewhat familiar with the material.

Demonstrations using software (R and Python) are provided to get users *started* (and *started only*) in applying these techniques to their own data. Most of the software applications are relatively easy and straightforward, adapted directly from R packages and Python sources so the user can follow up with these resources to learn more about the cited examples. They are on familiar “toy” data sets and not some arcane data sets that would be difficult to follow up on for further study, demonstrations, and exploration.

By featuring examples adapted from R packages and online Python sources it hopefully encourages students and readers to more fully “explore” the source’s capabilities. In most cases, the examples we feature in the book are mere “snapshots” and incomplete demonstrations of what the package or Python library is capable of simply to get the proverbial “ball rolling.” In this way, it is hoped that the book serves as a “gateway” for either further class or independent learning and guides students in the direction of the skills required by today’s data scientists, which includes adapting to new packages and libraries almost *daily*. Upon encountering a novel data problem,

the phrase “*I think R or Python has a package or library for that,*” becomes a common thought in tackling the data or research problem of the day. *Learning new things daily is what data craft is all about.*

To reiterate and emphasize, it is extremely important to realize that these demonstrations and examples are only meant to provide the reader with, once more, a *foot in the door* into these applications in getting started. The code is but a mere gateway to so much more that is *possible*. Statistical analysis and machine learning isn’t a *cookbook*, and very few if any textbook examples historically ever map on one-to-one to what is actually needed in *real research*. There are very good reasons why most R packages are upward of well over 100 pages, some well over 300–400! The skill of the data scientist is being able to be *creative, flexible*, and possessing a keen awareness of where to “dig” to follow up on the fairly trivial examples presented in this book and others. *Data can be a mess. Modeling it can be even messier.* In practice, each data set is unique and often requires a unique approach. Software demonstrations in this book are merely “glimpses” into the computational aspects and serve to demonstrate some of the theoretical development in the book.

#### 0.4.2 Theoretical vs. Applied vs. Software vs. “Cookbooks”

In the statistics textbook market, there are numerous kinds of books for sale. Some are very technical books on the theory of statistics and their mathematical elements, the field of so-called *mathematical statistics*. These books usually feature virtually no scientific or software applications. Their emphasis is primarily on theoretical aspects of statistics and proving theorems. While at first glance these books might appear “deep,” they are often only deep *mathematically* and not in the development of *statistical ideas*, though a mathematical statistician would likely say the ideas are *in the mathematics*, which at that more advanced level is entirely correct. Software books on the other hand give minimal explanation and theory, assuming the reader already “knows what they are doing,” as it were, as it pertains to understanding the material. For example, those big bulky SAS manuals have traditionally been of this sort in that they “recap” the concept somewhat and then go on to show the reader how to implement a methodology in software. Lastly, the “cookbook” is even more incomplete in that such books simply tell the reader *what to do* via software instruction without telling them *why*. The most ideal of books at an introductory level is probably one that contains *some theory, some application, and some software*. The book you hold in your hands contains that mix.

##### 0.4.2.1 Watered-Down Statistics

The study of statistics can be terrifying for most who are not aspiring quantitative specialists. It’s scary for the specialist as well, the only difference is that they have learned to push through the fear and swim among the sharks daily. All of this is for good reason, *statistics is difficult*, the complexity of which is only fully realized once one begins working with statistical formulae and applying them to data, or reading more advanced literature on statistics. As an example, I was recently asked by a researcher whether Cohen’s  $d$  values can be compared across data sets. That is, can we say one study with  $d$  equal to 0.6 had a larger effect than another study with  $d$  equal to 0.5? The answer is, as are so many answers in statistics, *yes and no* and more complex than first thought would suggest. A few perturbations of data points in one study and the effects could be reversed; a slight change in the standard deviation can have a rather substantial effect on the standardization of the mean difference. So, yes, they can be compared across data sets, *but not without a keen awareness of what went into each computation and knowledge of each data set*. Many answers are of this nature in statistics, such that the answer usually comes with *caveats*, sometimes many of them. For reasons such as this, when students actually embark on understanding statistics, it is rarely black and white,

and gray zones can, well, make one feel as though they have bees in their heads. *The seemingly simple problem can become quite complex very fast. Understanding statistics and how it should and should not be applied and interpreted on data is hard.* Figuring out optimal experimental designs can also be exceedingly challenging.

So, what's the solution? Make it easier of course! Anyone can write an “easy” statistics book, just as easily as one can *completely misrepresent the discipline and the inherent complexity of statistics and quantitative science more generally*. Books like that fly off the shelves and are sold mostly to individuals who do not want to study nor learn statistics in the first place and are merely looking for an *escape route*. Consequently, they learn “some” statistics, but at risk of it being terribly misunderstood. A little knowledge is a dangerous thing. Memorizing that the mean “is a measure of central tendency” with a formula showing the user how to compute it is dangerous. How so? For one, the *arithmetic mean* is only one of many *possible* means and it often doesn't adequately indicate the center of the distribution! “Incomplete” statistics books (e.g., software and cookbook manuals) are fine for those who are already aware of the complexities of statistics because they can navigate around the shortcomings in such books, but they are nothing short of *malpractice* if used in teaching as *primary* (not *secondary*) sources for students coming to the discipline for the first time. Without unpacking and working with statistical formulae to truly understand what they reveal vs. what they do not, statistics teaching is more likely to *mislead* than it is to properly *educate*. Alleviating anxiety by teaching “easy statistics” is unethical as such teaching completely misrepresents the subject. *These students are not learning statistics. They are learning something else.* The “easy books” can certainly be useful in a course, but should never replace a *primary* and more rigorous resource.

### 0.4.3 *Prerequisites to Reading This Book*

The prerequisites to reading this book are modest. We assume the reader has had at least one introductory statistics course at the undergraduate level. Preferably, the student will have also had an introductory course at the college level in calculus and linear algebra, or at least some *exposure* to these areas. The introductory course in statistics is mandatory, while the calculus and linear algebra courses are optional but preferred. The necessary prerequisites are reviewed in Chapter 2 and as needed throughout the book, but at a quicker pace than would be the case in a truly introductory course in these subjects. For instance, we may refresh the reader on why we are passing to the limit, but we assume the reader has had exposure to limits and derivatives before. Likewise with concepts from linear algebra, we discuss the *why* behind the concept and may re-introduce it briefly, but it would be beneficial if the reader has seen the concept before. From a generally applied perspective, it suffices that the reader has a sense of where the mathematics used “fit” in the bigger picture instead of presenting the mathematics as a sole topic as would be the case in an appendix or simply referring the reader to a book on pure linear algebra devoid of statistical application, for instance.

## 0.5 **Pedagogical Approach and the Trade-Offs of Top-Down, Bottom-Up Learning**

We would like to spend some time justifying the pedagogical approach used in the book and how it is similar to and different from other approaches that might be employed in teaching material of this sort. We also wish to give instructors and students ideas for how to learn this and similar material. It is hoped that both instructors and students will find this discussion fruitful in contextualizing their own teaching and learning. Nothing in what follows is meant to be rigid or dogmatic regarding how statistical and machine learning methods should be taught or assimilated. What follows is only meant to serve as an invitation to explore and discuss these approaches.



### 0.5.1 Top-Down, Bottom-Up Learning

There can be said to be two ways of learning the material of multivariate statistics and machine learning (Deisenroth et al., 2020). There are risks inherent in both approaches. The first approach is what we will call a *top-down approach*, where students are instructed from the highest and most general “overview” global perspective on what these methods accomplish and how they can be used. For example, when teaching principal components analysis, one could instruct students that PCA is designed to assess the dimensionality of the variables subjected to analysis. The description, though partially correct, is at the same time grossly incomplete. A *bottom-up approach*, on the other hand, would be that of beginning with fundamental mathematics such as essential ideas of vectors, matrices, and most importantly, linear combinations, which quite naturally leads into the definition of what a component actually *is*, that of the aforesaid *linear combination*. But again, quite incomplete without a blending of the aforementioned top-down perspective regarding the “big picture” from the top of the mountain, which is *dimension reduction*.

As an instructor, given that we only have a limited amount of time with students (or readers), we want them to take away the most essential *lessons* that they will carry with them for a *lifetime* and provide for them a *foundation* on which they can generalize and extend their abilities and understanding. We do not want them to get lost in the trees and not recognize the forest. The top-down approach with regard to PCA (as just one example of a technique) is incomplete of course because it neglects to *contextualize* PCA as *one* such technique for assessing underlying dimensionality, and of those techniques, only assesses *linear dimensions*. PCA, put into proper context, is but a special case of a wide variety of dimension-reduction techniques. The danger with so-called “cookbook” approaches to statistics and data analysis is that a student could come away from a course in multivariate statistics believing that PCA is *the one and only* technique for dimension reduction, which would be blatantly false. Not only does it deprive the student of learning, it is embarrassing for them to discuss or present the method in a public forum if they are asked about the method to any degree and hold fast and true that PCA is *equivalent* to dimension reduction rather than simply one *option* for it.

Hence, even if one begins with a top-down approach, either in teaching or learning, one should quickly get to the “bottom” to make these important distinctions, and the further the “bottom” we go, the more we get into the technical details, differentiations and distinctions, right up to necessary qualifiers such as *the assumption that we are discussing only applies to real numbers*, as an example. That’s probably too far, but the point is that students should be taught proper *context* in learning statistics and machine learning.

*Much of our thinking is grounded in metaphor and the art of substituting symbols, whether those symbols be words, pictures, or even musical tones.*

Equally, the problem with a *bottom-up approach* is that there can be so much focus spent on distinctions that the student may never grasp the bigger picture that PCA (to use the same example) is a dimension-reduction technique in the first place and how it can be applied to address scientific questions. After a semester of vectors, matrices, and other mathematics which may or may not include proofs, the student could feasibly come away from the course not truly understanding in any global way *what PCA is for, when it can (and cannot) be used*, and *what the big picture is* behind it. I have had first-hand experience in meeting many students who have had calculus, linear algebra, some statistics under their belts, yet who are unable to have a discussion about what these subjects are truly *about* because they were too busy intaking technical computational detail and

solving problems blindly or by rote computational heuristics via planned and patterned “problem sets” that they never quite grasped the bigger picture of what they were even “doing.” *Zero insight into the bigger picture.*

The pedagogical approach adopted in this book is to be mindful of the bottom-up, top-down distinction and not emphasize one *too much* at the expense of the other.

### 0.5.2 *Ways of Writing a Book: Making It Pedagogical Instead of Cryptic*

Perhaps the best way to become an overnight genius is to write a book nobody understands then claim authority over the knowledge! That is one approach to writing a book, albeit a poor one and very egoistic. Alternatively, one can write a book that many if not most understand and become known as a great communicator of challenging material. We have all come across mathematics texts that were “unreadable,” and the only audience for the book was between the ears of the author, while at the same time we have come across books that are far too general and appealing to too wide of an audience as to entirely misrepresent the subject.

Mathematics (or at minimum, mathematical *symbolism*) should not be avoided in a book like this, yet at the same time making such mathematics purposely *cryptic* benefits nobody. Rigor should not be avoided in teaching, but students should be instructed on *why* such rigor exists, the importance of differentiating concepts instead of blanketing them in imperfect understanding. The student should be given the opportunity to appreciate why rigor is prized by the mathematician or analytical philosopher in the first place.

### 0.5.3 *Standing on the Shoulders of Giants: A Companion to Advanced Texts*

Aside from this book being a stand-alone introductory textbook and reference to multivariate statistics and machine learning, it can also be thought of as a *companion* to understanding other texts based on the same or similar topics. The author’s aim was to make this text stand alone in its own right, but also be a “gateway” to understanding what the “next level” books discuss, so that the reader may follow up from this book and get into those more advanced texts with relative (or at least *more*) ease.

For example, if you are a newcomer to statistical learning, understanding existing standards in the field such as *Elements of Statistical Learning* (Hastie et al., 2009), *Probabilistic Machine Learning* (Murphy, 2022) or *Pattern Recognition and Machine Learning* (Bishop, 2006) can be at minimum difficult and daunting and at most impossible without a good teacher at your side. One of the aims of this book is to be that *companion*, that *teacher* to provide an interpretation of the concepts discussed in those higher-level books. With our emphasis on foundations, our hope is that the reader can then excursion into more advanced sources as well as begin perusing the research literature themselves having a foundational base from which to grow.

### 0.5.4 *Making Equations “Speak”*

The simplest of examples from introductory statistics will illustrate the style we adopt throughout the book. The reader will undoubtedly be familiar with the classic *z*-score:

$$z = \frac{y - \mu}{\sigma}$$

The reader will also likely remember being taught that converting to  $z$ -scores standardizes a distribution such to have a mean of zero and standard deviation equal to one. Perhaps such was the extent to which this concept was explored. If you look at the  $z$ -score more closely, however, what is it *exactly*? We see that it is a *ratio*, a ratio of  $y - \mu$  to  $\sigma$ . But what is  $y - \mu$ ? It is a *deviation*, a deviation from the mean to be more precise. What is  $\sigma$ ? It is another deviation, this one the *standard deviation*. Hence, we see the  $z$ -score is a comparison via a ratio of *two deviations*, one that you are computing based on  $y - \mu$  to one that is “average” or “standard” in the data, that of  $\sigma$ . Hence, if the observed deviation is about as large as the average deviation, then  $z$  is equal to approximately 1. If it is much larger, then  $z$  will be much greater than 1. This is one interpretation regarding what a  $z$ -score is actually communicating.

As much as possible throughout the text, space permitting, we seek to *unpack* the meaning behind equations to appreciate what they are communicating. Equations and formulas are, after all, *concepts*, and concepts should be explored for all of their meaning.

### 0.5.5 The Power of Problems

It is often said that in learning any technical area such as statistics, working out problems is the way to mastery. Tackling problems is not the only way to learn, but it definitely helps focus energy on elements of what you are learning. One can read on a topic to no end, but until you come across problems that take time, energy, brainstorming, and experience to solve, the learning may remain latent. That is, it hasn’t consolidated yet, akin to preparing for a life event yet not truly knowing if you are truly prepared until the event actually arises. I have taught many students who brazenly believed they understood statistics, that is, until asked to explain even basic statistics at a dissertation defense. As is true with life in general, *reality is the true test of one’s knowledge*. “Training” will only get you so far, as every athlete knows.

Each chapter in the book contains a brief set of problems, most are open so that the student can think more globally about the concepts learned to see where it takes her. Most are of the “discussion type.” Intellectual exploration of ideas is encouraged throughout this book, and problem sets attempt to foster this activity. The selection of problems is relatively arbitrary and we encourage instructors to expand or customize existing problems to suit the needs and scope of the course they are teaching. In the author’s own teaching, problem sets are usually improvised or customized to each lecture or class, and we encourage instructors to do the same. An “open problem” can easily turn into a more rigorous closed-answer analytical one with just a few additional elements. The problems in the book should be considered nothing more than good “starting points” to *initiate discussion* and are relatively few in number given space considerations for the book. If the reader or instructor would like more targeted and specific problems of a technical or analytical nature, they should feel welcome in contacting the author who would be more than happy to provide problem sets to suit the needs of their specific course.

### 0.5.6 Computing Languages

R and Python are the dominant languages currently in use for statistics, data science, and machine learning. Hence, they are the languages used for demonstrations of techniques in this book. However, it should be emphasized that *this book is not about computing*, and the reader should realize that computing languages come and go, but *foundations* are generally consistent. “Digging up code” in manuals and the internet is (relatively) easy, even if it can be time-consuming and test our patience! Understanding *why* you want to dig such code up is much more difficult and requires actually understanding the underlying principles and concepts upon which the code is based. The emphasis of this book is on *understanding first, computing second*.



Due to the sheer number of topics we treat in this book, we do not provide code for *every* topic. Instead, we are selective, providing code and examples for topics as a general survey of how Python and R can be used for various applications. For one method, we may provide code examples in both Python and R, while for others, only for Python or R individually. Not only was this to keep the book of manageable size and focus on the most important matters (i.e., explanation of what the actual procedures accomplish), but it is also for the reason that some procedures are simply more amenable to one software package over the other. R is featured a bit more throughout the book than is Python. As an instructor using the book, they should feel free to bypass examples using the software they have not adopted.

### 0.5.7 *Notation Used in the Book*

Half of the battle with reading a book on multivariate statistics or machine learning can be making out the notation used. Even if you are experienced in the area, you can spend a great deal of your time trying to figure out how the author is using this or that notation. What a waste of time! We avoid that as much as possible in this book. Mathematics is a beautiful subject in its own right and to those who dislike it, it is nonetheless a necessary evil to understanding multivariate and machine learning, but *abuse* of mathematics is all too rampant; otherwise great texts ruined because the author does not explain what the mathematics mean or the symbols used in describing such mathematics. We try to avoid those pitfalls the best we can in this book, though we no doubt do so imperfectly.

Our use of subscripts is purposely non-consistent throughout the text. At times, we subscript *variables*, whereas other times we do not. To use a simple example of this approach, a *z*-score can be defined as

$$z = \frac{y - \mu}{\sigma}$$

or like this,

$$z = \frac{y_i - \mu}{\sigma}$$

where  $y$  is now purposely indicated as a variable,  $y_i$ . Both  $y$  and  $y_i$  denote a variable in this context, and both forms are acceptable for this and other statistics and equations featured throughout the text. As well, the transpose of a matrix  $\mathbf{A}$  can be denoted  $\mathbf{A}^T$  or just as well simply as  $\mathbf{A}'$ . In places using  $\mathbf{A}^T$  seemed more “natural” than  $\mathbf{A}'$ , whereas in other places  $\mathbf{A}'$  was preferred as it is a bit less cumbersome. Random variables versus *realizations* of those random variables is also not always consistent. For example, formally, for a given random variable  $X$ , a realization of that random variable could be notated as  $x_p$ , such that  $X = x_p$ , however using uppercase throughout the text we found would be too cumbersome and overall preferred the smaller notation  $x$  or  $x_i$  even when potentially referring to a random variable. Lastly, in a regression context,  $\beta x$  or  $\beta x_i$  was, in general, deemed preferable to  $\beta X$  simply because again we favored the smaller  $x$  notation regardless of whether or not  $x$  was considered fixed or random.

## 0.6 **Nobody Learns a Million Things: The Importance of Foundations and Learning How to Learn**

A great teacher once said, *geniuses chunk information, that's what makes them so smart*. Contrary to how undergraduates (and beginning graduate students) often perceive their professors, no

person, even those running around with PhDs, can know *everything* (some even know very little!). But the smart ones *know where to look*. They know which “mental folder” to open and pursue. What is even more vitally important is that they will have a sense of when they are right or wrong. They will also get a sense of when the answer is unachievable or uncertain. The maturity to achieve these milestones is no small feat and is gained through experience. The undergraduate knows a lot, but the seasoned scholar is much more tentative with her own knowledge. *Knowing when you don’t know is probably the most important of all knowledge.*

The philosophy adopted in this book is that it is far better to learn and understand a *few things* really well than to try to learn *everything*, which as mentioned, is always impossible for anyone. With the explosion of statistical learning and such in the current climate, it can be tempting for a book writer to present a “catalogue” of methods, a wealth of “facts” the reader can access anytime they wish. And though, as mentioned, this approach has its place and there are a number of excellent books that already attempt this, for an introductory text or university course a better approach would be to emphasize the more common and *foundational principles*, the rest of which become derivatives and extensions or special cases of these majestic pillar areas. Such is the approach taken in this book. Though our approach is generally “top-down” in that we seek to give the reader a bird’s eye view of what is going on in statistics and machine learning, we prioritize communicating the *foundations* upon which most other topics one could come across are then “learnable.” Without the foundations in place, one’s understanding of new things has no legs to stand on. *Good education starts with strong foundations.*

### 0.6.1 Essential Philosophy of Science and History

Understanding the mathematics of multivariate statistics and machine learning is insufficient for the scientist. One must also have a grounding in the scientific *context* in which these tools are used. Ask the average student in the sciences (or surprisingly, even in mathematics) what philosophy is, and you will probably get some version of pop culture’s interpretation similar to the response you would get if you asked the same question regarding the nature of psychology (“oh yeah, I’ve heard of Freud”). However, closer to truth and the position advanced in this book (and my other books) is that *philosophical thinking is the basis upon which all other knowledge is derived or created*. All of the concepts, statistical and mathematical formulas that we will see in this book have *foundations*. Those foundations are conceptual and very deep. Concepts, even “rigorous” ones do not arise or evolve as concrete objects. Instead, they are first “mushy” and imperfect. There is perhaps no better example of this than the evolution of calculus and the idea of *limits*. The limit concept was not “invented” by Newton or Leibnitz, but instead evolved over a couple thousand years. The great inventors helped “finished the job” as it were (along with several mathematicians afterward), but it should always be understood that they completed a project that was in the works for a very long time. So it is with even advanced statistics and machine learning. The foundations to these “modern” tools often go back centuries. Knowledge is a *progression*, it usually does not simply arrive “on the scene” without prior development.

Why is understanding and appreciating philosophy of science so important and relevant for the scientist? Consider the issue of causation. Statistical analysis could never reveal causation. Causation, whatever it may mean in the first place (for centuries, philosophers have debated the issue), concerns itself mostly with *research design*, at least in experimental science. Further, methods such as neural networks, at least historically, have been more focused on simply getting predictions and classifications “right” without feeling the need to know “why” those predictions are accurate. Analysts using neural networks have more or less been satisfied with the unknown function remaining a “black box” as it were. In many areas, we’re not trying to necessarily create a

mathematical “causal” model as one would, in say, biology, but instead are simply trying to create an algorithm that will predict or classify successfully and minimize error and misclassifications in doing so. One is more tolerant of the prediction equation being a “black box” than in sciences where explaining the predictors is of paramount importance. For instance, understanding what factors predict the orbit of a satellite is not simply a black box problem. We have to know “why” so that we understand variations in trajectories.

Consider how a student might learn statistics or machine learning in a typical course. They perhaps take a course on the subject, study the materials used in the course, and then most likely prove results associated with those objects or apply those objects to empirical problems often using software. Presumably, one can learn what correlation *is* through its formulaic definition just as one can learn what a *group* is in abstract algebra and then go on to utilize that concept in solving problems. All this is fine and good, but a more *thorough* and *deeper* understanding of these topics can be achieved by studying their *history*. Pearson’s correlation is not merely a formula in a textbook. Understanding correlation means appreciating and understanding where it came from, how it evolved, and ultimately how it revolutionized both statistics and science. This can be said to be true of virtually all statistical and mathematical objects.

Indeed, the seeming infinite regress that occurs when one begins questioning, as though one is a child, eventually brings one to philosophical and historical inquiries. What you typically see in textbooks, including the current one you are reading, is the current “state of the art,” of a discipline. But beneath that state of the art is a long history of *ideas*, *concepts*, *controversies* that culminated in the “polished” object you currently see in your textbook. Pragmatically, there is a need to cut that search off at some point, but if you question further about the objects you are learning you will eventually become a philosopher and historian, because, quite simply, there is *nowhere else to go to learn of the true foundations*. As we will emphasize throughout this book, *true understanding is found in the foundations*, though we stop well short of examining historical and philosophical underpinnings. The point we make is merely for the student to recognize the importance of understanding statistics and machine learning in proper historical and epistemological *context* and to be aware that these directions can be pursued, should one choose well beyond this book.

### 0.6.2 *Beyond the Jargon, Beyond the Hype*

We hold that the purpose of education is to see inside of the “black box” down to its simplest of principles instead of mapping one’s perception of a mathematical tool to some sophisticated and overblown ideas of what that tool actually *is* and what it can do. A few Google searches will convince anyone “in the know” that artificial intelligence and machine learning are continuously marketed as being mysteriously complex. Let the world think what they will. The job of the student and aspiring professional is to learn what these things actually *are* stripped down to their most basic and essential fundamental principles. This is what “expertise” actually is, awareness of what is going on *inside* of the *perceived complexity*, not being part of the populace that has false perceptions, misunderstandings, and misguided glorifications about the tool.

As an example, consider the word *algorithm*. The *idea* of an algorithm is not new. It is *ancient*. *Algorithms have existed forever*. Even the Babylonians were working with algorithms (Knuth, 1972). Yet without understanding just what *is* an algorithm, the newcomer might be fooled by today’s climate into thinking they are “modern” in some sense. One might even use the word algorithm and automatically associate it with cutting-edge technology. True, modern algorithms are new, but the *concept* is old, and being aware of this can help demystify the nature of the algorithm in today’s modern climate. Conceptual understanding knows no label, it often cannot be encapsulated in a single word or phrase or definition. It is more of an *idea*. All otherwise “precise”

definitions arise from much *fuzzier* concepts. And if we traced concepts such as algorithms all the way back, we would certainly discover that the earliest of peoples were in fact thinking algorithmically. *Modern ways of describing concepts or ideas do not necessarily imply those things or ideas are modern.*

### 0.7 The Power and Dangers of Analogy and Metaphor: Ways of Understanding

Every explanation is a *way of understanding something, no matter how near or distant the explanation may be*. At first this may seem like a radical point of view. Understanding what is a chair can be helped by understanding what a bench is, but the study of the bench is not the “end game.” It too is a way of understanding something. The concept of a “bench” is elusive, the definition is only an *approximation to understanding its nature*. The sooner we realize this, the sooner we can accept analogies and metaphors as vehicles to understanding concepts in statistics and machine learning, while also appreciating that the truest “forms” (to be somewhat Platonian about it) are not in the mathematical equations. Those too are *approximations*.

As an example, suppose a student takes a course in linear algebra to understand the objects of that subject. Suppose they wish to understand what the *rank of a matrix* is all about. The rank of a matrix is the maximum number of linearly independent rows or columns of a matrix. Is that the “end game” to understanding rank? No! It is but an *approximation*. Not only is there a multitude of ways of conceiving rank from different angles in mathematics, the reason *why* rank exists is much deeper. There is a wider *context* to understanding rank, just as there is a wider context to understanding anything else, whether that be in mathematics, statistics, machine learning, or English literature for that matter.

Consider the following analogy that might help in understanding what rank is:

*Rank is sort of an indication of whether the matrix under consideration contains overlapping “information.”*

Of course, the above is not the full story, but the point is that it *approximates* the full story, at least to some degree. The concept of “information” as a generalization of a column or row vector is a weak generalization of what a column or row vector is, but the analogy helps “encapsulate” the concept and serves a purpose so long as one realizes there is always more to learn about the concept.

The danger, of course, is that all formalism gets lost and all that remains are imperfect, imprecise, and inexact analogies. The purpose of using analogies and metaphors is not to “water down” the material; it is simply to approximate true forms a bit more closely to bring the student into the *neighborhood* of the actual object, where encapsulating the “true” object is somewhat elusive. Defining the arithmetic mean as *where most of the data falls* is a terrible definition and just as terrible of an approximation. Settling on that definition in undergraduate statistics would be doing a tremendous disservice to students. Hence, one must be very cautious in using analogies and metaphors too liberally, even with the awareness that all definitions, even the ones closest to the concept, are nonetheless of the analogy/metaphor type, even if at first glance they do not appear so.

Throughout the book we use analogies and metaphors in places while usually emphasizing that they are just that, *approximations to understanding the true concept*. They are “ways” of trying to get *closer* to the object. So long as that is understood, such tools may be helpful and useful in understanding. Indeed, one of the dangers of focusing too heavily on precision and formality is that the concept might be lost altogether. As discussed, concepts are “fuzzy,” they exist within a

philosophical “cloud” of sorts where the well-defined object can experience a degree of interpretation without losing the essence of what it actually is.

### 0.7.1 *The Infinite Regress of Knowledge: A Venture into What It Means to “Understand” Something and Why Epistemology Is Important*

Statistics, data analysis, and machine learning are *hard*. Anyone can compute a correlation coefficient or have software do it for them. That is not the difficult part. The difficult part is making sense of empirical observations and applying analytical mathematical approaches in trying to understand something elusive we call *reality*. Many students are amazed at how difficult a statistical problem can become when given just a little thought. Others simply think we’re making the problem difficult on purpose and never learn to appreciate finer distinctions. Rigorous courses in statistical consulting spend a long time assessing and debating over statistical problems that end up having potentially very complex solutions, much different from the typical and overly simplified undergraduate solution obtained by referring back to the course textbook to see if the “right answer” has been achieved. The icy winter driveway pales in comparison to how “slippery” statistics can be when applying them to data. Entire conclusions can be flipped around by looking at the data in a slightly different *way*.

But how does this thing called *understanding* work? It has been my experience that many times “breakthroughs” in understanding, those “ah ha!” moments are about things students have been told before, perhaps many times, but now it has finally been learned “anew” in a much, much deeper way. It is grasped at a much deeper level, and seemingly for the first time. Technical understanding parallels other types of understanding, such as how to navigate through life and the proper mental mindset for peace and success. “*Life is what you make it*” is what we have always been told, yet one day we finally “get it” and realize the truth and depth of that maxim to its fullest. We learn it for the very first time even though we have heard it all our lives.

Understanding does not typically occur *linearly*; it usually occurs in *leaps and bounds*, or through “steps” as in the step-like function of Figure 0.1, where one day the understanding and grasp of the material seems to hit a new unexpected level, seemingly almost overnight.

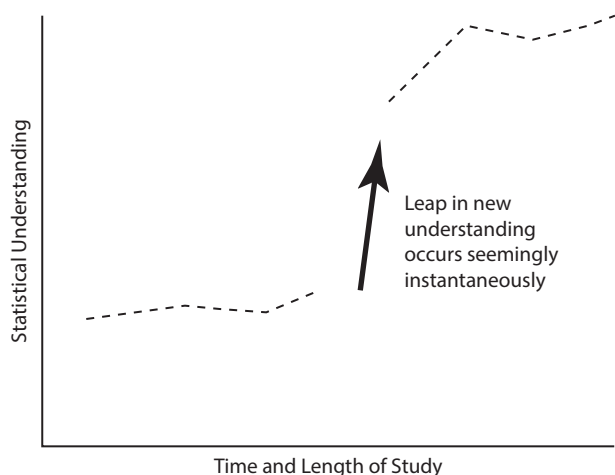


*Figure 0.1* Statistical understanding rarely occurs linearly, but instead in leaps and bounds as indicated in the step-like function. The upward arrow represents a “Eureka” moment where new understanding seemingly occurs in an instantaneous “leap.”

As practical advice as you embark on your learning of statistics and machine learning, *do not try to understand everything at once*. Allow for “approximation” and “iteration” as you study. Allow for fuzziness each time you look at it. *Do not let perfection be the enemy of good* but rather study these things often with a fresh perspective and hopefully the fuzziness will begin to be replaced by understanding.

In this respect, we follow general and excellent ideas provided by Hubbard and Hubbard (2002) in their advice on how to read mathematics more generally:

Reading mathematics is different from other reading ... there are two parts to understanding a theorem: understanding the statement, and understanding the proof. The first is more important than the second ... If you're still having trouble, skip ahead to examples. This may contradict what you have been told – that mathematics is sequential, and that you must understand each sentence before going on to the next. In reality, although mathematical writing is necessarily sequential, mathematical understanding is not: you (and the experts) never understand perfectly up to some point not at all beyond. The “beyond,” where understanding is only partial, is an essential part of the motivation and the conceptual background of the “here and now.” You may often (perhaps usually) find that when you return to something you left half-understood, it will have become clear in the light of the further things you have studied, even though the further things are themselves obscure.”

(p. 2)

#### 0.7.1.1 Epistemological Maturity

One focus of the book is to situate statistical models and other *ideas* within a wider *context*. For example, set theory is often regarded as one of the most unifying areas of mathematics (though many would say now that *category theory* eclipses that perspective). A sign of maturity in knowledge of statistical and related models is being aware and understanding the more *global structures* of which smaller structures are special cases. These global structures can be more difficult to “see” as one considers more complex models. However, a simple example will illustrate the concept.

Most readers familiar with the analysis of variance will readily be aware that the between-subjects *t*-test can be considered a special case of the wider between-subjects ANOVA model. This is what we mean by *maturity* – one sees the higher-order structure from which others can be deduced and situated. This process continues for increasingly complex models. While at first many users of statistical methods may prefer a “cookbook approach” for simply selecting the “correct” method for a given problem, the maturity of the user in relation to statistics becomes evident when they realize there is never only a *single* way to analyze data. The cookbook approach is not necessarily always *wrong*; it is usually simply grossly *incomplete*.

If the student understands statistics at a higher level beyond the recipe book, not only do they become aware of different possibilities when analyzing data, their ability to interpret research findings skyrockets. The student no longer assumes that PCA is *equivalent* to dimension reduction, for instance, and thus when the conference speaker reports, “Our PCA did not reveal any components with eigenvalues greater than unity on the correlation matrix we analyzed, hence, the data are not amenable to dimension reduction,” the knowledgeable researcher is immediately aware of the incorrect conclusion to the research report. *The PCA did not reveal linear dimension reduction, the bigger picture of whether underlying dimensions exist in the data has not yet been answered, and may even be unanswerable, since the underlying manifold may not yet have been uncovered. The discovery of dimensionality depends greatly on the tool used in the search.* This is no trivial



difference in conclusions, it is the difference between an *incorrect* vs. *correct* interpretation of the research outcome. *The distinction matters.*

The above is a perfect example of the hierarchical thinking we are attempting to generate in this book. Statistics is slippery; you can think you know it, you can think the formula or data say one thing, but underlying that conclusion is a new *base rate*, a new *denominator*, and if you're not aware of that underlying base, you are bound to conclude it is winter because you are standing in your walk-in freezer! The ratio  $2/4 = 0.5$  is with certainty, until we consider that fraction relative to the new base of 5, then  $(2/4)/5 = 0.1$ . We were so sure 0.5 was the final answer until we incorporated the new base. Context is everything when it comes to statistics, mathematics, and science more generally. The narrow view of looking at only a few of the many variables in existence can seriously bias the conclusion. Of course, the best of mathematicians, statisticians, scientists, and researchers know this, it is second nature to them. But to the student of these areas, this is what is most important in learning: *there is always context to any conclusion; change the context, and the conclusion usually changes as well.*

## 0.8 Format and Organization of Chapters

A survey of each chapter follows:

*Chapter 1* features a discussion of the philosophical foundations of the methods featured in this book as well as the more broad topics of statistical learning and machine learning.

*Chapter 2* features an overview of the mathematical and statistical foundations required for the rest of the book. We discuss the “pillars” of statistics and machine learning, namely probability, statistics, linear algebra, calculus.

*Chapter 3* features a (very) brief introduction to R and Python software. Both software packages are used throughout the book in providing brief “snippet” demonstrations of techniques.

*Chapter 4* denotes the actual beginning of the statistical and machine learning segment of the book. We begin with a discussion of familiar univariate ANOVA models.

*Chapter 5* surveys simple linear and multiple linear regression models, effect sizes for such models, mediated and moderated regression models, principal components regression, partial least-squares, and canonical correlation analysis. Regression is the bedrock of many statistical and machine learning methods and much of this chapter lays the foundation for material to be discussed later in the book.

*Chapter 6* features a discussion of regularization in regression. We introduce the concept of regularization, then discuss the most common ways of regularizing a model, which includes ridge, lasso, elastic net. We also survey least-angle regression.

*Chapter 7* features a discussion of nonlinear regression models, including polynomial regression, splines, kernel regression, and others. We also relate some of this material to neural networks discussed later in the book.

*Chapter 8* features a discussion of the generalized linear model, which includes the popular logistic regression, Poisson model, and others. Generalized additive models and discriminant analysis are also discussed. We close out the chapter with a discussion of the nonparametric technique of  $k$ -nearest neighbors.

*Chapter 9* surveys support vector machines (SVMs) as a competitor to discriminant analysis for when the response is binary. We discuss SVMs for both the linearly separable case and non-separable case, which will necessitate using kernels to address the nonlinearity in way of expanding the feature space.

*Chapter 10* features a discussion of decision trees and committee machines, including bagging, random forests, and boosting.

*Chapter 11* features dimension-reduction techniques such as principal components analysis and manifold learning more generally. Independent components analysis and probabilistic PCA are also surveyed, in addition to nonlinear dimension-reduction techniques such as kernel PCA. Principal curves, multidimensional scaling, self-organizing maps, locally linear embedding, isomap, sne, and *t*-sne are also discussed.

*Chapter 12* surveys the method of exploratory factor analysis, including the many caveats and potential problems in interpreting factor analysis. Independent factor analysis, nonlinear factor analysis, and mixtures of factor analysis are also discussed.

*Chapter 13* features a discussion of confirmatory factor analysis, path analysis and structural equation modeling. These models can be interpreted as extensions of regression and exploratory factor analysis and can be considered a special case of the wider dimension-reduction or manifold learning paradigm.

*Chapter 14* is about cluster analysis and data segmentation. We survey such methods as *k*-means and hierarchical approaches to cluster analysis, including density-based clustering as well as mixture models.

*Chapter 15* surveys the method of neural networks. We develop the neural network from first principles of regression, then contextualize neural networks relative to other statistical and machine learning approaches such as projection pursuit.



# 1 First Principles and Philosophical Foundations

In this first chapter, we introduce and survey “big picture” topics that are pertinent to the fields and evolution of statistics and machine learning. These areas of study, just as is true of virtually all areas of inquiry, rest on deep *philosophical assumptions*. To get “deep” is to get philosophical. To the uninformed, the word “philosophy” might bring to mind someone casually questioning the meaning of life or similar pursuit whether in a university classroom or on a Sunday afternoon hammock. Even worse, to the naïve the word might bring to mind something one does if one is determined to not gain employment! To the educated, however, philosophical foundations are everything and are at the very core of virtually all knowledge. *Anything studied enough, questioned enough, critically evaluated enough, will find its way to philosophical assumptions and foundations, into “mushy” territory where distinctions are made not in terms of symbols such as in mathematics, but in pre-symbolic forms in terms of “ideas” that often cannot be easily operationalized. Transforming the “mushy” into “rigorous” is, in many cases, brutally difficult.*

## 1.1 Science, Statistics, Machine Learning, and Artificial Intelligence

Attempting to distinguish between what exactly *statistics*, *machine learning*, and *artificial intelligence* are can seem like a trivial waste of time for knowledgeable researchers. However, for the student, a comment on these distinctions, if there are any, is very important as they are the ones coming to these fields for the first time and do not have the luxury of seeing things in the perspective of a rearview mirror to be able to note which distinctions are important and which are indeed, trivial; that is, which features are overlapping versus which are unique to their respective areas.

Historically, numerous attempts have been made to define just what *is* artificial intelligence, and none of them are truly satisfactory. None can be said to have truly “captured” the concept, in large part because whenever one tries to define something so precisely one usually misses the mark, akin to trying to get a hold of a single observation on a continuous scale – *the more you zero in on it, the more elusive it appears*. Hence, as is true of most things, artificial intelligence is a *concept* that cannot be truly captured by precise definitions and vague language. Even defining precisely what mathematics is can take up a dissertation! The best one can do is get a general *sense* of what it is and what it is not and go from there. Still, we may be able to settle on the following imperfect definition:

*Artificial intelligence is the science of machines, usually computers, carrying out tasks that would typically require something called “intelligence” if the same task had been performed by a human being.*

Artificial intelligence (AI) can hardly be considered a “science” in the spirit of laboratory science. Proponents of *weak AI* hold that a machine is intelligent if we can get it to carry out sophisticated tasks. Proponents of *strong AI* hold that artificial intelligence is the ability of machines to “mimic” human intelligence. Of course, strong AI may not even be achievable, but weak AI is possibly so, especially with the rise of machine learning.

*Machine learning* is often defined as the ability of computers to learn from data without having to be explicitly programmed. Machine learning is related to artificial intelligence because an intelligent system should, at minimum, be able to adapt to changes in its environment, which is precisely what machine learning algorithms purport to do. While machine learning is largely associated with computer science, much of its history can also be found in cybernetics and control science. More recently, the origins of machine learning are often associated with the psychologist Frank Rosenblatt who essentially conceived of a learning machine that we would today call a basic *neural network* (Fradkov, 2020). It was work on the backpropagation algorithm in the 1980s that truly initiated modern interest in neural networks more specifically and when eventually combined with the cheap availability of parallel computing and memory, the stage was set for the “gold rush” of machine learning in the 2000s (Fradkov, 2020).

Where does *statistics* fit into all this? How closely is machine learning related to statistics? The following are a couple of quotes from experts in the area that highlight the closeness of statistics and machine learning:

Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions.

(Goodfellow et al., 2016, p. 95)

If the statistics field had incorporated computing methodology from its inception as a fundamental tool, as opposed to simply a convenient way to apply our existing tools, many of the other data related fields [such as machine learning (ML)] would not have needed to exist – they would have been part of statistics.

(Friedman, 1997, p. 27)

Then there is the field of *computational statistics* which is described as *computationally intensive methods in statistics* (Rizzo, 2019) and which focuses on such techniques as the bootstrap and numerical algorithms that overlap a great deal with the subject matter of machine learning. Some even distinguish between *statistical computing* vs. *computational statistics*, the former having more emphasis on numerical methods and algorithms while the latter featuring additional topics such as exploratory analysis, data partitioning, and others (Rizzo, 2019).

When one browses the literature on statistics and machine learning, one will definitely notice differences. Machine learning “feels” more about the *algorithm* than it is about the *data*. For example, in statistics we use linear regression to model data, but in machine learning we are studying how effective linear regression is when compared to other “algorithms.” The computer science perspective is the theoretical study of algorithms, how they perform, how complex they are, their computational efficiency, how much they can store in memory, etc., whereas statistics feels more about fitting a precise scientific model to data. Of course, both overlap a great deal, but have slightly different *flavors*. We should always remember, however, that the computer is nothing more than an electronic machine that speeds up computation in either case. Much of computer science has little to do with the computer. It has everything to do with mathematics and statistics, to which computer scientists have contributed to immensely.

As we will see throughout the book, many effective machine learning algorithms have been inspired by age-old mathematical ideas for *function approximation*. Indeed, one does not need a machine at all to do machine learning. The stranded person on a deserted island can (at least in *theory*, a lot of theory!) perform simple machine learning just as the high-tech computer scientist, but without the benefit of automation or an electronic machine. “Machine learning” is simply a name to reflect that many processes are *implemented* by a machine, namely, the computer. But make no mistake, machine learning itself, the *concept*, is as old as it gets. Yes, the Greeks were doing “machine learning” in some capacity long before humans “invented” it.

### 1.1.1 *Mathematics, Statistics, and Computation*

There is sometimes the perception by nonspecialists that “math is math” and that there is little creativity, in a sense, in developing such math. The author has served on student committees where the perception of mathematics is that it is *static*; *it is what it is* and not subject to invention, creativity, amendment, or reconstruction. Of course, this perception of mathematics is wholly false, as creativity in mathematics is what mathematics is all about. However difficult mathematics is, the mathematics itself is entirely (mostly) logical, and solutions are often correct or incorrect, at least at relatively elementary levels. The high-school student completing the problem set quickly looks at the back of the book to see if she got the solution correct. If the solution at the back of the book does not match the solution she obtained, her answer is deemed incorrect and she must then figure out why. Rarely is there flexibility in the solution for elementary problem sets. The “feedback loop” is relatively clear and straightforward.

The further away we move from mathematics, mathematical statistics, etc., the more we stumble upon areas where clear-cut and unique solutions often do not exist. Instead, there are often a *variety of solutions available*, some better than others, and many dependent on the preference or analytical style of the practitioner. This is often the most frustrating of experiences for the newcomer to applying mathematics and statistics to empirical problems. “Good solutions” are not unique and usually require the practitioner to infuse a mixture of methodological and philosophical thinking into the effort of arriving at the *best* solution. That is, the mathematical solution is only the *beginning*; she must now interpret what that solution *means* in the context of the scientific problem. This is where students often have difficulty, struggling with regards to what is the most “optimal” (in some sense) solution. Rigorous thinking is good, but in most things you also have to allow for compromise in “gray” areas and swim in a degree of flexibility. Within *pure* mathematics, most is quite rigorous, but only if fundamental philosophical issues are ignored that are at the foundations of such surface rigor. Deeper study into the philosophical assumptions of mathematics and one is no longer doing mathematics.

### 1.1.2 *Mathematical Systems as Narratives to Understanding*

One reason, I believe, that students can be “put off” by mathematics isn’t because of the difficulty of the subject. It is because they do not appreciate or understand what mathematics actually *is*. It can be very difficult to motivate someone to do something when they do not know what it is you are asking them to do and why. Asking a student to compute the quadratic formula to solve a quadratic equation is one thing, but explaining to them *why* solving for roots is such a big deal in the first place is quite another. Most students are excellent at blindly working the formula (basic arithmetic is really not *that* hard), but understanding the “why” beyond such rote manipulation is much more challenging.

Mathematics can be perceived in many ways, but we find it most useful for the student to perceive mathematics as a *narrative*. It is a *way* of explaining something, just like other narratives are

ways of explaining other things. Of course, mathematical ways are often more *defined*, *exact*, and *precise*. A linear equation is quite precise, whereas using the English language to describe how your day went is quite vague. It is worth noting that not all things we describe are amenable to description using mathematics. The climate of one's soul is probably not amenable to mathematical description, for instance. Self-concept is probably not amenable to mathematical description despite psychologists' efforts to measure such things.

Mathematics, much beyond the early teachings of it merely being computation is often said to be about *form*, *structure*, *symmetry*, *equivalences*, *analogy*, and so on. However, these properties are not unique to mathematics. All of these elements are present in ventures such as music, art, physics, and even psychological theory. The only difference in the last of these is that rigor is much more difficult to implement, that is, the "rules" are not nearly as rigid. What makes mathematics unique is the relative *simplicity* (not *complexity*) in definitions that allows for rigor and communication among participants in the discipline. For instance, a *function* can be crisply defined in mathematics. Attempting a definition of the soul in philosophy or psychology can take up a dissertation.

## 1.2 The Scope of Data Analysis and Data Science: Expertise in Everything!

The field of data analysis and data science can easily feel overwhelming. Why? *Because you need to know everything!* At least it can feel that way. As has been discussed many different times by a number of authors, data science lies at the intersection or heart of so many specialty areas. To understand deep learning, for instance, you must have a strong grasp of statistical principles. But the machinery (not necessarily the *concepts*) underlying statistics is mathematics and if you venture into more foundational issues within mathematics itself you will soon encounter philosophical and historical issues, so you should to some extent be aware of those areas as well. But we're not done. Not even close. You also need to be a computer scientist to some extent so you can apply all of this theory to the computer, understand the strengths and limitations of the computing you are doing, and be comfortable enough with code that you can improve or tweak a program if needed, or at minimum know if it needs tweaking and where to look for answers. So, so far, you're to some extent a *mathematician*, *statistician*, *philosopher*, *historian*, and *computer scientist*. That's a tall order for anyone.

But we're still not done. Data science is applied to substantive areas such as business or a science (e.g., economics, psychology, sociology, biology, physics, astronomy), so you need to be an expert, to some extent, in the area of application as well. It becomes very difficult to know where and how to apply the technical elements you're learning if you are unfamiliar with the discipline and the problems that the discipline seeks to solve. If it's a scientific area, then you are also required to have knowledge and experience in areas of *research design*, *experimental* vs. *non-experimental designs*, *cross-sectional* and *longitudinal models*, and so on. Indeed, as we will discuss in this book, most if not all of machine learning does not make even one mention of design issues, and so it is possible that an expert machine learner may be somewhat naïve about important distinctions such as that between *experimental* vs. *quasi-experimental designs*, and so on. These design issues rear their heads daily in fields such as biology and psychology. Couple all of these areas with the problem of measurement such as in sciences like psychology or economics with latent variables, and you then need to also be an expert at psychometrics or measurement theory more generally. This is all to say that the expertise required to be a good data scientist can feel overwhelming and essentially impossible to ever achieve.

### 1.2.1 Theoretical vs. Applied Statistics and Specialization

There exists a *paradox of specialization*, such that if you focus too much on one area at the expense of others, you then become out of balance. For instance, since the mathematical foundations of

statistics and data science are largely found in areas of linear algebra, analysis, and optimization, one can easily be tempted to study these areas in some depth, but the more you study, the more you become a specialist in these areas at the expense of the professional you are seeking to become. There is no natural “hierarchy” of knowledge such that if you study the foundations in one area it will naturally make you an expert in others. *What you specialize in makes you a specialist in that area.* Figure 1.1 is a prototype of the well-known Venn diagram of data science showing the intersection of many different areas of expertise. If you are to be a good data scientist it is important to remain focused on the common intersection of the three circles. Veering too much in any direction, while allowing yourself to deepen in that specific specialty, essentially detracts from the objective of becoming a good data scientist.

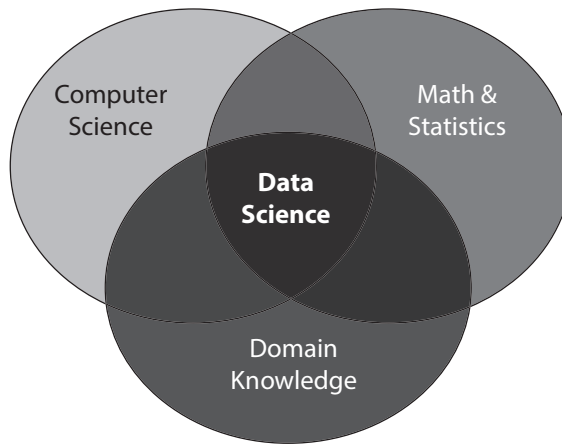


*Figure 1.1* Data science can be said to be at the intersection of various skills and areas of expertise. Specializing too much in one area at the expense of another can detract from specializing in the “Data Science” intersection.

### 1.3 The Role of Computers

Computer scientists are professionals who study such things as the capacities of computers to compute things, limits on what can be computed, and all the rest of the details regarding computation. However, for our purposes, computing can be seen as a way of getting things done quickly and in an automated fashion. Computers do what we tell them to do, and they do it extremely well, fast, and reliably (Guttag, 2013). As Stillwell (2016) notes, computation has always been a part of mathematics [and statistics], but it is only in the past century (1900s) that it became a mathematical *concept*.

As a scientist using multivariate or machine learning methods, computing is an important element in accomplishing your goals. However, most things that require computing, at the level we are talking about in this book at least, can be dug up and searched. How to conduct a spline regression via a computer isn’t nearly as important as understanding the benefits and limitations to conducting a spline regression and what precisely spline regression actually *is*. It is this *understanding* that is paramount, at least in this book. From this vantage point, computing is seen as simply a *means to an end* to getting the job done. Now, of course if your work involves databases of humongous size, then the ability to even compute on such large databases comes into question. Though we do touch on these areas slightly in this book, *computational complexity* is not our chief concern.

For the most part, we assume that the data you will be working with is of reasonable computational size. This book is not about *data engineering*. It is about statistics, science, and machine learning.

### 1.3.1 The Nature of Algorithms

In discussing the role of computers in today's climate, it is an ideal time to note that today's statistical and machine learning "Zeitgeist" seems to be all about *algorithms*, almost as if such things are new and were invented yesterday! Indeed, the very word *algorithm* carries with it a "sexy" appeal among students.

But what is an algorithm exactly? A very general definition is that it is *a method or set of instructions for carrying out a procedure* (Tanton, 2005). Under this general definition, preparing a meal is an example of an algorithm, just as walking your dog is an algorithm. Driving a car is an algorithm just as brushing your teeth is one. Writing this book was an algorithm. Algorithms are present in virtually every facet of our lives. Quantitatively, an algorithm can be as simple as long division or one could argue that even adding two numbers is an example of an algorithm. Misconceptions of algorithms being "new" to modern data science are just that, *misconceptions*.

### 1.3.2 Algorithmic Stability

As we have discussed, the focus of machine learning is a bit less on the data and a bit more on the actual algorithms that carry out the data analysis. How stable are they? How stable is a procedure such as regression across similar or different samples? Does it have such high variance that even slight perturbations of the data would result in wildly different outcomes? Questions such as these fall under the general rubric of *algorithm stability*. A stable algorithm is generally one for which if the training data are altered or perturbed slightly, the output of the algorithm more or less matches the degree of perturbation in the data and that as  $n$  increases these changes eventually disappear altogether (Liu et al., 2017).

*Uniform stability* specifies a bound by which two samples cannot exceed in order to be considered stable. More narrowly, a learning algorithm is uniformly  $\beta$ -stable if when a hypothesis trained on two different samples,  $h_s$  and  $h_{s'}$ , that differ by only a single point, the difference in loss functions  $L$  is equal to or smaller than a constant  $\beta$  with probability equal to 1. That is,

$$p[(|L(h_s) - L(h_{s'})|) \leq \beta] = 1 \quad (1.1)$$

where  $S$  is the original sample,  $S'$  the sample with perturbation, and  $\beta$  is the *stability coefficient* that sets the upper bound (Liu et al., 2017; Mohri et al., 2018).

A related concern is how fast or efficient is the algorithm or computation more generally. A computation is said to run in *polynomial time* if the number of elementary steps can be expressed as a polynomial function of the size of the input. Running in polynomial time is generally considered a favorable characteristic of an algorithm. A computation not running in polynomial time is often running in *exponential time*.

## 1.4 The Importance of Design, Experimental or Otherwise

Statistics and machine learning without consideration of design issues, from a scientific point of view, is vacuous. Indeed, one of many features that makes this book unique from other books on multivariate analysis and machine learning is the constant reminder throughout that *design issues are most important from a scientific point of view*, not the actual method or algorithm applied to



the data. That is, if you are doing *science*, then design matters most. Statistics, machine learning, and computing come in second.

For example, you can perform a cluster analysis with your fancy machine learning software and algorithm, but it will never tell you “why” people who drive Hondas form a cluster from those who drive Nissans. It will tell you what is *associated* with each cluster possibly but it will likely not tell you *why*. Only experimental designs can even come close to approximating such conclusions. Hence, throughout the book, we remind the reader that this or that statistical or machine learning tool often cannot answer the deeper scientific questions of most interest. Only good design can provide approximate answers to those questions. A statistical or machine learning model cannot tell us whether vaccine A is better than vaccine B when it comes to treating COVID-19, but a good experimental design might.

### 1.5 Inductive, Deductive, and Other Logics

In philosophy there are generally two broad ways of reasoning, of “getting to the truth” as it were, *inductively* and *deductively*. Crudely, inductive logic is that of drawing conclusions from the *specific to the general*, whereas deductive logic implies drawing conclusions from the *general to the specific*. As an example, if I have never seen dogs before and witness ten dogs all with four legs, I may wager that *all dogs have four legs* based on the observations I have witnessed. This is an example of inductive logic, where the particular is the set of dogs I have witnessed and the general is that of all dogs, most of which I have not witnessed and may never witness. The larger set of observations could even be infinite in size. On the other hand, if I somehow already knew that all dogs have four legs, then it is an easy matter to *deduce* that the next dog I meet will have four legs. In this case, we are going from the general to the particular and is an example of deductive logic. *Boolean logic* is an example of a deductive process; for instance, statements such as if  $a > b$  and  $b > c$ , then  $a > c$ , the law of *transitivity*, is an example of a deductive “implication” statement in Boolean logic (Lucci & Kopeck, 2016). Empirical science is necessarily inductive as in most cases we simply cannot have access or observe all events and must draw conclusions based on only a select few. Science can be said to have *deduction envy* in that it would love to be deductive, but such is generally not philosophically possible.

Generalizing from the specific to the general is usually imperfect and comes with error. *Statistical inference* is a way of generalizing from the specific to the general and relies heavily on probability theory. When we see ten dogs with four legs and induce that all dogs have four legs, we are taking a “leap” in such a conclusion. That is, we do not *know* if all dogs have four legs; we are “gambling” or “betting” that they do based on our current observations. This is how science generally proceeds. *Science is a gamble*. Indeed, *inductive learning* is about finding functions that best approximate an unknown function, which is essentially what *statistical learning* is about.

There also exists *fuzzy logic*, which implies that a proposition can be “sort of true.” This is related to the more broad idea of *fuzzy sets*. In fuzzy logic, one is not either dead or alive, but instead one is dead to a *matter of degree*. Likewise, fuzzy set theory assigns membership to sets by matter of degree, not absolutely. It must be emphasized that inductive and deductive logic are well a part of our lives and science; they form the basis of much of our thinking about what is “true” or “false.” The Greeks are largely responsible for providing us with the rigors of deductive logic, where mathematical statements, for instance, are only regarded as *true* if they can be rigorously deduced.

More recently, theoretical logicians such as Boole and Frege elevated logic into its own mathematical discipline. Despite the long and historical study of logic, it should be emphasized that the “goodness” of logic relies on the assumption that *logic itself is correct*. If our assumptions about

the power of these logics ends up being found false one day (presumably by way of even more logical thinking!), much of mathematics if not all would fall to the ground as would other modes of thinking as well. That is, if essential *arguments* such as the following could somehow be refuted one day (e.g., through a truth table shown to be false), or at minimum no longer generally assumed to be true, it would pose a revolution (to put it *mildly*) in science and thinking in general:

$$\begin{array}{l} p \rightarrow q \\ p \\ \therefore q \end{array}$$

where the above reads that if  $p$  then  $q$ ,  $p$ , therefore  $q$ . Of course, at first glance, the proposition that the above *modus ponens* could actually be incorrect as a way of concluding  $q$  sounds entirely irrational if not idiotic. More than likely, these essential capstones of inference will continue to hold the test of time. We merely point out the nature of *constructing knowledge* of this kind, that so much of it depends on prior assumptions, or provable assumptions based on even deeper assumptions no matter how “convincing” or “obvious” those assumptions may be. History can sometimes surprise us, as we will soon discuss in reference to Gödel’s theorems.

We can demonstrate simple logical arguments in software. In R, for instance, logical arguments can be carried out quite easily. For example, we can ask R to evaluate certain expressions as true or false:

```
> 1==5
[1] FALSE
```

where in the above, `==` is called a *relational operator* (Davies, 2016). Another example is the following:

```
> 5 == 4+2
[1] FALSE
```

Since logic contains “1” and “0” values corresponding to true and false, we can treat logical values as numbers as in the following where we add up “true” values:

```
> T + T + T
[1] 3
```

### 1.5.1 Consistency and Gödel’s Incompleteness Theorems

Euclid’s *Elements* is regarded as one of the most famous of all mathematical texts ever written, and more specifically in geometry. The fifth postulate of the text was for a couple thousand years assumed to be an axiom, but in the 1800s it was shown that Euclidean geometry was not the *only* geometry possible. One could construct an edifice of *non-Euclidean geometry*. This event helped to introduce a crisis into mathematics.

Up to late 19th century (1800s), much of mathematics was thought to be based on logic, and given this, one could reasonably expect that the entire edifice of mathematics would be essentially a *closed system* that is entirely logically consistent. Consistency, in this regard, refers to the idea that if a statement is true, it cannot simultaneously be false. More succinctly, a consistent system is one that is free from *contradictions*. For instance, if  $p \rightarrow q$ , a contradiction here, for example,



might conclude  $q$  and  $\neg q$  (i.e., not  $q$ ) *simultaneously*. A consistent system does not contain such contradictions.

After the invention of the calculus in the 1600s, it was discovered that calculus was not based on a theoretically solid foundation. A group of mathematicians attempted to rigorize calculus and were largely successful. The new discipline became known as *real analysis*, or simply *analysis*, and purported to be a rigorized and deeper version of calculus. For example, the notion of continuity at the time of Newton and Leibnitz was quite informal. Real analysis rigorized continuity and gave it a very precise definition. There would be no more informal talk regarding what was meant by notions such as continuity. Mathematicians delivered a rigorous definition for these and other key mathematical ideas. Mathematics prided itself on rigor, precision, and well-defined statements and definitions.

While the attempt to put mathematics on a rigorous logical foundation was relatively successful, the idea that it could be *entirely* put on such a coherent and consistent foundation was challenged by Gödel. Gödel showed that there are some statements in mathematics that could not be proved from other statements *within the system*. The theorems advanced by Gödel in showing this were known as his *incompleteness theorems*. The second of his theorems essentially held that it is impossible to program a computer to answer all mathematical questions. As Tantom (2005, p. 229) notes, “Gödel’s work dashed centuries of hope of finding a small set of basic axioms on which to base all of mathematics.”

The above surprise was just the beginning of reconceiving mathematics not as a bullet-proof edifice of irreversible truths, but rather as a *construction* which like many other things requires something to initiate the construction. Not everything within the constructed system can be proved or justified. This was exactly what was found by Gödel who propositioned that mathematics, more precisely *Peano’s axiomatization of arithmetic*, contains within it statements that cannot be proved with reference to the system. That is, it must require something *external* to the system for the rest of the system to be proved. In other words, any first-order theory containing Peano’s axiomatization “is either self-contradictory or incomplete for the purposes of deriving *certain* [emphasis added] arithmetic statements” (Muggleton, 1991). As Kline (1972, p. 1026) remarked:

The rigorization of mathematics may have filled a nineteenth-century need, but it also teaches us something about the development of the subject. The newly founded logical structure presumably guaranteed the soundness of mathematics; but the guarantee was somewhat of a sham. Not a theorem of arithmetic, algebra, or Euclidean geometry was changed as a consequence, and the theorems of analysis had only to be more carefully formulated. In fact, all that the new axiomatic structures and rigor did was substantiate what mathematicians knew had to be the case. Indeed, the axioms had to yield the existing theorems rather than determine them. All of which means that mathematics rests not on logic but on sound intuitions.

Kline hits the nail on the head. *Dead center*. While analysis provided a rigorous “foundation” for calculus, it was essentially yet still a *human construction* for ideas already assumed to be true. Again, the classic case here is that of continuity. Continuity was assumed to already exist; all that mathematicians did was provide a more rigorous description or account of it. The contributions were extraordinary in nature, no doubt, but many of such contributions simply provided justification for concepts already assumed to exist without justification. The assumptions (and intuition, as Kline put it) still came *first*.

### 1.5.1.1 What Is the Relevance of Gödel?

The reader may reasonably ask why any of this Gödel talk is relevant to this book. In truth, it isn't, other than to augment awareness of the mathematical system on which statistics and machine learning are based. In all of the statistical and machine learning principles and methods we will survey, it should be recognized that these are essentially *human constructions*; they are not written in stone and not *entirely* deductions from a smaller set of fundamental truths. Too often students believe that mathematics is rigid, dry, and outside of the realm of creativity. Nothing could be further from the truth. Mathematics, and by extension statistics and machine learning, are *creative enterprises*. Rigorous ones, but creative just the same. Rigor does not imply the absence of creativity; it simply requires stronger justification for new ideas and principles born out of such creativity. The rules are much “stiffer” than creativity in music or art. Sure, in jazz there are “rules,” but in the end, you can play what sounds “good.” When reflected on even more deeply, it may be realized that *we make up the rules in all of these*, including in mathematics, statistics, and machine learning, yet more accountability for such rules exists in these latter three since most rules are a consequence of *logic*.

## 1.6 Supervised vs. Unsupervised Learning

In an attempt to classify statistical and machine learning methods, the distinction *supervised* vs. *unsupervised* is often made. We first discuss the distinction, then explain why such a delineation is incomplete and insufficient. As is true of many ways of “splitting” a concept, what may at first seem clear and useful quickly becomes “fuzzy” and “muddled” upon closer and deeper philosophical scrutiny. As we will see, the difference between what is supervised vs. unsupervised gets quite “murky” very fast.

As the story goes, in *supervised learning* we have prior-established designated labels on the response. Supervised learning is sometimes called *learning by teacher* because we have already defined the classes we are trying to predict. For example, if we are using education level to predict salaries *equal to or greater than* vs. *less than* \$100,000, we have clearly identified the labels  $\geq 100k$  and  $< 100k$ ; thus, the problem is supervised. Methods such as linear regression and discriminant analysis are generally regarded as supervised learning. We know what we are trying to predict or model.

In *unsupervised learning*, on the other hand, we do not have prior-established classes or labels. That is, we are not attempting to predict or classify into categories because the categories are currently unknown, or, at minimum, unused. Dimension-reduction techniques such as *principal components analysis (PCA)* or *cluster analysis* are often regarded as unsupervised. Unsupervised methods are often considered more *exploratory* in nature than their supervised counterparts.

There is also a third category, that of *semi-supervised learning*, which is when we do not have all of the data and can alleviate the need for labeled data by capitalizing on the unlabeled data (Mohri et al., 2018). Semi-supervised learning approaches are beyond the scope of this book.

### 1.6.1 Fuzzy Distinctions

Despite the traditional arguments for the distinction between supervised vs. unsupervised, if forced to choose one over the other for all methods, we would likely conclude that most techniques are *supervised* in some sense and that unsupervised techniques really do not exist. To do this requires us to generalize our idea of “supervised” somewhat to a bit more than simply denoting labels on the response. It requires us to accept that most if not all methods are, in one way or another, *user-driven* at least to some extent, even if minimal in some cases.

Consider that the moment the user is required to make any decisions regarding the method, he or she is either directly or implicitly supervising the approach. Consider principal components as an example, which is usually assumed to be an unsupervised method. As we will see later in the book, there are many different ways of conducting PCA. Choosing among these versions is an act of supervision, just as are choices regarding whether to standardize data and analyze the covariance or correlation matrix as input. Further, once the dimension reduction has been performed, the decision of how many components to retain is a supervised *act*. The situation is even more supervisory in a related so-called unsupervised method, that of *factor analysis*, where the user often must decide *a priori* how many factors to extract from the covariance or correlation matrix before even running the procedure. Telling the computer how many factors to compute is an extremely “supervisory” decision based on our revised version of what supervisory means.

As another example, consider *cluster analysis*, another so-called unsupervised technique. Such a procedure requires *a priori* for the user to select a metric by which distances will be operationalized. Once these inputs have been made, we allow the algorithm to “do its thing” in a non-supervised way, but the choices that govern the use of these methods are quite similar to the choice made in, for instance, the type of regression one performs in a supervised learning context. Should the user go with ordinary least-squares for a given problem or weighted least-squares? How about least-angle regression? In the end then, though the distinction between supervised and unsupervised is useful, when considered in a deeper way and with a bit of tinkering with the definitions, we arrive at the conclusion that *all methods are supervised, some less so than others, and some not as explicitly as others*.

It should be emphasized that many distinctions such as these in multivariate statistics or machine learning, when *critically challenged*, become quite *fuzzy* and blurry to the point where the distinction has trouble holding water, as it were. This is perfectly fine and does not mean the distinction isn’t useful or wrong; it simply means we should not think too *rigidly* about any of these definitions. Definitions such as these are useful, but they are often weakened when we put them under a microscope and tweak them slightly. Never *memorize*. Instead, *conceptualize*.

## 1.7 Theoretical vs. Empirical Justification

In philosophy, there is a difference between *theoretical* vs. *empirical* justification. We illustrate these differences with a couple of powerful and recent examples.

### 1.7.1 Airplanes and Oceanic Submersibles

The airplane lifting off the runway is an empirical demonstration that *airplanes fly*. It is only one demonstration of the idea. If many such airplanes successfully lift off the ground, we gain confidence in the idea that “airplanes fly.” That is, the certainty that airplanes fly is due to empirical justification, empirical “proof” we might crudely say.

The above is different from theoretical justification. The theoretical justification comes from an analytical argument, usually based on mathematics in this context, that airplanes *should* fly. That is, if the theory of flight is correct based on precise engineering aeronautic principles, then the airplane should fly. As much as possible, scientists should analytically justify what they seek to observe, or even have observed, *empirically*.

Why is this important? Recall the OceanGate expedition of 2023. The Titan oceanic submersible was successful in delving to the bottom of the ocean in visiting the *Titanic* many times such as to suggest the idea that it was a safe vessel to embark on such excursions. However, as critics noted after the submersible went missing (and was later found destroyed due to implosion), theoretical justification for the suitability of the vessel was lacking. That is, the vessel was not *analytically*

*justified* for such excursions. Empiricism may reveal something to “work” a few times, but only *theoretical justification* can give us increased certainty that such will work in general for a large number or even infinite number of trials.

### 1.7.2 Will the Bridge Stay Up If the Mathematics Fail?

A classic example of the distinction between theoretical vs. empirical justification is perhaps the construction of a bridge. If one day the mathematics that led to the construction of the strong bridge suddenly were found faulty, would the bridge still stay up, or would it immediately fall? Of course, it would likely stay up, *at least for a little while, maybe even longer*. However, it is the theoretical justification of the bridge’s construction that we typically rely on to ensure its long-term reliability and sturdiness. Theory, or rational argument, makes the knowledge on which the bridge is constructed more *certain*; not *guaranteed*, since the theory itself is tentative, but simply more certain.

Theoretical and empirical justification must work hand in hand, however, and serve to inform each other, but one should never take a few empirical demonstrations to “justify” a theory that has not been justified *a priori* independent of those empirical demonstrations. This is the strength of sound mathematics and proofs, or more generally, rigorous *philosophical argument*. Such serves as a rational “guarantee” that the way we are thinking about constructing the bridge is sound.

With regard to the OceanGate submersible, perhaps to a large extent its prior success was based on capitalizing on random chance. Without theoretical justification, the experiences or instances of what you have obtained could simply be *lucky instances*. The fact that the submersible successfully visited the *Titanic*  $n$  times does not guarantee a successful visit on the  $(n + 1)^{\text{th}}$  time. Regardless of how many successful trips there were prior to the accident, those successful trips were nonetheless a *sample*, and in this case not a random sample of all potential trips since most likely the probability of failure of the submersible structure increased the more times it traveled down to see the *Titanic* due to increasing structural fatigue of the vessel. A sample of successful voyages was observed, but making any kind of inference to the population of potential voyages requires some guesswork. *Statistical inference is guesswork. It is the art of imposing rigorous quantification on such guesswork.*

## 1.8 Level of Analysis Problem

Some physicists will claim that their discipline is the “fundamental” one, meaning that subjects such as biology, psychology, etc., are merely special cases of this fundamental discipline. Let’s briefly examine what this might mean, and how it introduces the *level of analysis problem*.

Suppose for a moment the physicist is correct, that all science reduces down to near infinitesimal particles, the presumed true building blocks of the universe of which all else is constructed. For the moment, we will call this the *deepest* level of analysis. Even if this is the case, the psychology experiment that finds a group difference in psychological health between those who practice mindfulness vs. those who do not could be considered a higher-level analysis (in the sense of a *top-down* view only) of these infinitesimally small particles. Whether the particles also conform to this group difference structure is simply a different *perspective*. The point is that the psychology experiment has revealed at a higher “bird’s eye” level a group difference and that group difference is just as meaningful as analyzing the atoms at a lower level. *The representation of the data is at different levels, but the phenomenon under study may be essentially the same.*

Hence, aspiring scientists should be aware that there are different *levels of analysis* and that they all contribute to understanding. Debating over which is the most “fundamental” unit of analysis is an interesting debate, but is not one that should detract from the triangulation of many different approaches to studying a given phenomenon.

## 1.9 Base Rates, Common Denominators, and Degrees

So much of everything in research is about *base rates*. As a data analyst or researcher, you should always be aware of the base rate phenomenon whenever you either conduct or perform research. It is literally impossible to contextualize research correctly without such an awareness. Regarding the *Titan*, many experts in the area warned that the submersible may have problems due to its structural integrity or lack thereof. Upon first hearing there were concerns about the sub, it may have been tempting to interpret this information as *unique* to that particular submersible. The news reports brought to mind of a particular sub with significant weaknesses and dangers. Upon hearing these reports, however, it behooves the critical listener to consider *whether such dangers were unique to all submersibles of which the dangers with this one were simply a special case*.

What we are getting at with the above question is the concept of a *base rate*. Surface knowledge can be very misleading and is too often the only knowledge we “see.” There is usually always a base rate to any knowledge, but too often it’s the surface knowledge that will be incorrectly emphasized at the expense of the background or context layers. Now, it turns out that the issues with the submersible were *uniquely* significant and likely led to the demise of the sub and all who were aboard, but the point is that when you learn of abnormality or presumably “unique” problems with anything, always ask yourself whether those things are truly unique to the object you are working with or whether there is a deeper underlying *base rate* such that the unique characteristic is actually much more common than initially believed. Simply because an event is observed does not imply that such an event is *unique to the particular object you are working with*, even if on the surface the mind can quickly treat the characteristic as if it is unique.

Gigerenzer (2003) calls this awareness of base rates a *mind tool* that all researchers or anyone consuming research should practice and nurture in their thinking. The (first) attempted assassination on President Trump in 2024 was presumably carried out by a “loner” type. However, many human beings would fit that description and many more human beings likely have psychologies exponentially darker than that of the gunman. Simply because we observed a characteristic of the gunman does not explain, by itself, the behavior. Awareness of base rates here is essential. When the “warning signs” are characteristic of say, 20–30% of the population, they are not warning signs anymore. *They are features of human nature*.

Relativity also plays a role in the consideration of base rates. One begins to think less in terms of *categories* and more in terms of *degrees along a continuum*. Once more referring to the *Titan* submersible, it was said in the aftermath of the implosion that passengers should not be allowed into experimental submersibles or indeed any experimental transport due to the risks involved. But what is *not* experimental? Few would regard airline travel as experimental. Due to its safety record it has been put into the “category” of non-experimental travel. However, from a philosophy of science perspective, *airline travel is indeed still experimental; it is simply less so than venturing on the OceanGate submersible*. Airline travel is still a relatively recent invention. Relative to the entire span of humanity, people have been flying airplanes for a very short time. Contextualizing events along a continuum instead of assigning such events “black and white” labels is epistemologically fruitful. For many travelers, their Delta flight tomorrow may be “experimental” enough of a risk!

### 1.9.1 Base Rates and Splenic Masses

A failure to consider base rates can have drastic consequences on decision-making. German Shepherd Dogs are well known to be very intelligent and for the most part well-tempered and trainable dogs, which is one reason why they are often employed as officers in police departments and beloved by owners as family. However, they are unfortunately known to have health problems, especially as they age. One typical health problem is that of tumors of the spleen, which may be benign or malignant (cancerous). Though cancer is of course detrimental, the worst part of a splenic mass is the chance that it ruptures into the abdomen and the dog bleeds internally.

Suppose you perform an elective ultrasound on an 11-year-old male German Shepherd. The ultrasound reveals a splenic mass. As a dog owner, you are of course concerned that the mass could rupture but have no idea how long the mass has actually been there. The mass, of course, is an abnormality on the spleen, meaning that the dog was unlikely to have been born with the mass. It appeared at some point during the dog's life, though at what point, you do not know. You lack an ultrasound *baseline* with which to compare the current data. It is an abnormality on the spleen, but is it an abnormality for an 11-year-old German Shepherd? Likely not. Consider data reported recently on the prevalence of splenic masses on a random sample of Golden Retrievers aged 6 through 13 that found that a whopping 52.8% of Golden Retrievers in this sample were found to have splenic masses! Further, since it is well known that splenic masses are much more common in elderly German Shepherds than young ones, the percentage of elderly dogs (i.e., a *conditional distribution*) carrying splenic masses is probably closer to 70–80%. Hence, the detection of a splenic mass on the ultrasound of the 11-year-old is most likely an *expected* finding, not an *anomaly*. The point here is that there are likely many dogs that carry splenic masses, many of which are likely benign and for which those masses never pose a problem or the dogs die of other causes. Indeed, the next 11-year-old (or older) German Shepherd that you observe on a walk down the street with their owner *likely has a splenic mass*. That a mass was found on the ultrasound does not make it an anomaly; *the detection of the mass and doing an ultrasound on the dog may be the actual anomaly*. This is what we mean by *base rate thinking*, and once you start thinking in terms of base rates, you will see them *everywhere*.

Indeed, in the literature on splenic tumors in dogs, suppose it is reported that approximately  $\frac{1}{2}$  of tumors on the spleen are benign and  $\frac{1}{2}$  malignant (this statistic is not entirely accurate, but is convenient for this discussion). However, suppose this estimate is typically based on the *presentation* of the tumor in an emergency bleeding state, that is, when the dog begins bleeding from the tumor and is brought in for emergency treatment. The correct statement should be  $\frac{1}{2}$  of *bleeding* tumors are benign and  $\frac{1}{2}$  malignant, not  $\frac{1}{2}$  of tumors *unconditionally* (i.e., bleeding or not bleeding). The Venn diagram in Figure 1.2 depicts the base rate idea we are referring to.

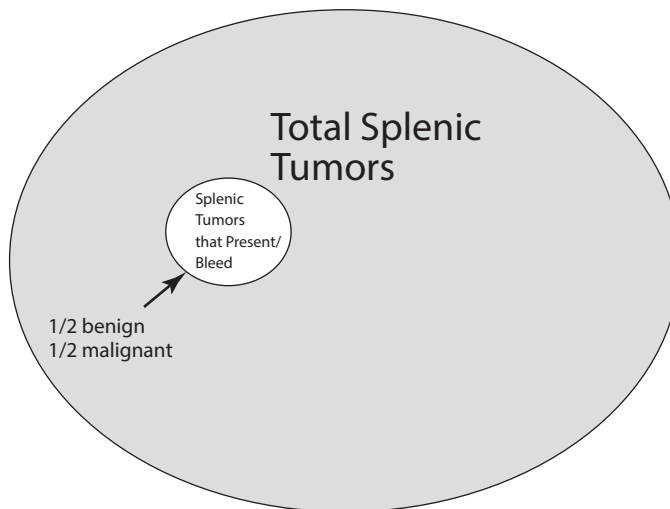


*Figure 1.2* Of all total splenic tumors, suppose the statistic  $\frac{1}{2}$  benign and  $\frac{1}{2}$  malignant refers only to those that are bleeding upon presentation. The actual number of dogs with tumors that are non-bleeding is likely much larger and hence may not be subject to the aforementioned proportions (benign vs. malignant). The total number of dogs with splenic tumors is the base rate.



The  $\frac{1}{2}$  and  $\frac{1}{2}$  estimate is based on tumors that are based on presentation in a vet's office, and hence "known" because they are usually bleeding or causing symptoms in the dog. Dogs that do not have reason to present, however, may have tumors as well, as evidenced by the Golden Retriever study. Many of these dogs will die of other causes, and hence the probability of a tumor being malignant is much less *if the tumor is not bleeding*.

Awareness of base rates is a powerful tool in this regard such that even experienced veterinarians may neglect when interpreting novel cases. Veterinarians and other medical professionals are trained in medicine, not *statistical decision theory*. As a student learning statistics you will become very aware of when your medical provider is not *thinking clearly* when it comes to important decisions.

### 1.9.2 *Probability Neglect*

The above discussion of base rates is also related to another idea that dominates the interpretation of probabilities, that of *probability neglect*. For the German Shepherd, a veterinarian discussing a few cases of ruptured tumors might arouse great concern over the possibility that the current tumor in your dog ruptures. The focus is then on the rupturing tumor but ignoring the true probability that the tumor does not rupture. It is as much a *statistical phenomenon* as it is a *psychological* one, where intensity is magnified for the current event at the expense of other events that are likely more probable. The idea of the tumor rupturing is at the forefront of our minds, yet we ignore the probability that it does *not rupture*, which could be much higher.

Another example of probability neglect is that of the experience post-9/11, 2001, in which the population at large was afraid to fly. The presenting information of the tragedy of seeing the planes crash into the towers (and Pentagon and field in Pennsylvania) dominated people's consciousness, when in reality, the probability of a plane crashing, even after the tragedy, continued to likely be remote. "*I don't want to fly because it is too dangerous,*" says the individual post-9/11, but then travels by car on the interstate where the probability is significantly higher of an adverse event. *We must be very cautious not to ignore probabilities simply because the event we are currently considering evokes strong emotion.*

### 1.9.3 *The "Zero Group"*

Yet another general application or instance of base rate thinking is what we shall refer to as the *zero group* in research. This is the group that is not participating in the study but may actually be of most interest. Without awareness of there being such a group, the investigator may entirely miss it and not be aware of it.

As an example, consider the online question of whether people like Honda vehicles. The ratings are exceptionally high, but what may be being recorded are only those who like Hondas! If the true percentage of individuals liking Hondas is 30%, and those 30% are the only ones responding to the survey, it can be falsely concluded that 100% of individuals like Hondas. The 70% who do not like Hondas never responded. This is what we are calling the *zero group* in research. *It is the group that never makes its way into the study but is of utmost interest to the calculation of the effect or the prevalence of what is being measured.*

There is usually (perhaps *always*) a zero group that may be out of sight that one actually does want to study. The key is to be *aware* of this possibility so that you can incorporate it into your scientific conclusions even if you cannot directly analyze the group. Another example of a zero group was the 2016 election (or possibly 2024 as well, for that matter) where Trump voters refused to be polled or were hesitant to admit they would vote for Trump. The pollster was attempting

to measure something called *voting preference*, but the most important group, the Trump voter, eluded them either through improper sampling or no tried and true way to vet dishonest responses.

### 1.10 Statistical Regularities and Perceptions of Risk

To extend on the above discussion, *statistical regularities* often make extremely risky engagement seem less so. For example, consider once more airline travel. Is airline travel risky? Is it dangerous? Of course it is! Traveling at greater than 30,000 feet in what is essentially still an experimental capsule is a risky endeavor. Yet, from a statistical point of view driving to the supermarket in your car is riskier. How we qualify risk changes depending on the perspective we adopt. A car traveling on a highway at 65 mph is less risky than an airplane traveling at 30,000 feet going 400 mph, but not *statistically*.

Statistical regularities are well known for reducing our *perception of risk*, but this is only so in the sense of past behavior. That is, if an airline flight is similar to most other airplane flights before it, then statistically it comes with little risk. However, despite the modeling of your flight today as having come from a distribution of past flights, we simply have no knowledge of the current event. *It is entirely new*. It is entirely possible (though highly improbable) that most flights from now on go on to crash; we simply do not know as we have not yet observed these new instances. All we have for certain are probabilities which we apply to new data under the *assumption* that such data are being drawn from similar probability distributions as past events.

Hence, one might say that the flight at 30,000 feet is risky in *absolute* terms, but statistically, it is much less so. In statistical parlance, the flight is *relatively* risk-free because the Bayesian prior of it being successful is quite high (we will discuss Bayesian statistics shortly). However, philosophically, the prior tells us nothing about the actual new physical event. It only models its probability under the *assumption* that the new event was drawn from a distribution made up of similar prior events.

#### 1.10.1 Beck Depression Inventory: How Depressed Are You?

Risk is of course also prevalent in the diagnosing of clinical disorders. Suppose as a clinician you administer the *Beck Depression Inventory (BDI)* to a new client and that client scores in the range of mild depression. The BDI is a well-known screener for depression. A good clinician knows that no sole diagnostic criteria should be used to diagnose a client. However, what does a high BDI score signify? *It does not signify the disorder*. It instead can only indicate a “sign” of the disorder and classic Bayes reasoning needs to be applied to obtain the actual probability of depression. If the client’s prior probability of the disorder is low, then a high BDI score may simply not be enough to indicate or diagnose depression. This is a basic principle of revising probabilities, yet it has been shown time and time again that such basics are not understood or appreciated by many who diagnose medical and clinical disorders. Patients’ lives have been turned upside down due to inaccurate reporting of HIV test results, for instance. This is more than a triviality; it is of vital importance. *A failure to communicate statistical results properly to a clientele is equivalent to malpractice. Patients and customers have a right to be told accurate probability estimates and given guidance on how to interpret them.*

### 1.11 Decision, Risk Analysis, and Optimization

The above discussion of risk and decision-making quite naturally ties into ideas of formal optimization in mathematics and statistics. In many methods featured in this book, the objective, from



an optimization point of view is to *minimize* or *maximize* some function, which in the end is essentially translated into minimizing the risk of making a wrong decision with as little error as possible. The area of study called *risk analysis* or the related area of *decision analysis* are fascinating disciplines of study and permeate many applied areas of practice. In the end, many statistical analyses are performed with the end goal of making a *good decision*. However, few if any decisions come without risk. Though one can argue there are, in theory, perfect decisions, they are only “perfect” in the sense of being rigorously thought out and rationally conceived. They all contain consequences and costs of being wrong, at least in the short term.

Every day we are confronted with decisions and we make them with imperfect knowledge. Whether we formally conduct a decision analysis on such decisions or intuitively try to balance the risk of making a wrong decision with the benefit of making a right decision, these decision tasks permeate our lives. There is a whole literature on the psychology of risk and how human beings manage risk and the interested reader is encouraged to consult such literature should these topics be of interest to her. For now, it is simply enough to survey a few features of this wide and exciting area of research.

### **1.11.1 The Risk of Making a Wrong Decision**

Consider the following hypothetical scenario – you have just paid off your home, valued at approximately \$400,000. While legally you were required to have fire homeowners insurance when assuming a mortgage with the bank, now that you own it outright you are no longer required to hold any insurance. You ponder the question carefully as to whether you should still purchase insurance or pocket the premium you would pay and devote those dollars to something else. The only way to truly address this decision problem is to assign probabilities to competing events and evaluate *expectations* or *utility*. If the price of insurance is \$1,000 per year, it may seem a no-brainer to pay the premium since the cost of losing the home to a fire would be devastating financially at a total loss of \$400,000, the market value of the home. However, if you are one to seldom leave your home and the risk of fire is remote, perhaps the \$1,000 spent on an insurance premium is best spent elsewhere. Of course, most homeowners would pay the premium as the risk of losing the home, even if small, is too large as to not guard against such an event. Yet, at the same time of not being able to presumably tolerate such risk, the same individual regularly and voluntarily goes joy-riding in their car on a very dangerous stretch of highway where the risk of losing one’s life is much greater than the risk of losing one’s home to a fire. The point we are making is that risk tolerances can vary widely depending on the given problem and the perception of risk, rational or irrational, can be variable depending on the context and situation.

### **1.11.2 Statistical Lives and Optimization**

The idea of risk also translates to notions of statistical lives. Returning to the airline example discussed earlier, airlines seek to minimize loss in terms of injury and deaths. Yet at what cost? Certainly not at *any* cost. Airplanes regularly land in thunderstorms and adverse conditions even though such conditions pose more risk than others, yet such risk is *calibrated* against other interests, such as retaining customer satisfaction by a timely landing, keeping flight schedules on track, as well as the interests of the crew in wanting to be done with their flight as a result of fatigue. That isn’t to suggest a flight crew would recklessly land in a severely dangerous thunderstorm, but it would be tremendously naïve to think that tolerable amounts of risk are not existent and implicit in such landings. When those calibrations begin to impose too much, such that they risk exceeding the risk *threshold*, which is usually probably subjectively determined case-by-case, then a more dangerous level of risk arises which may lead to accidents. The point we are making here is simply

to say that no behaviors are risk-free, and an airline that purports to manage a risk-free operation of flight is simply not being truthful. *Risk is minimized but such minimization is subject to constraints and competing interests.* Optimization algorithms in statistics and machine learning also optimize, but usually not *unconditionally*. They usually optimize subject to constraints and conditions.

### 1.11.3 Medical Decision-Making and Dominating Criteria

In basic decision theory, we say a decision is *dominated* by one or more criteria if that decision trumps a competing decision based on selective input or information. For example, consider the dog owner once more regarding the detection of a splenic mass via ultrasound. Is it benign or is it cancerous? Of course, a benign mass will not spread to other parts of the body, whereas a malignant mass is more likely to do so. The decision is whether to remove the spleen via surgery. In this example, one can spend much time and expense trying to determine whether the mass is benign or malignant, yet the decision to remove vs. not remove the spleen may be *dominated* by the possibility of the mass bleeding or rupturing, which could put the dog's life in danger regardless. One could CT-scan the body to look for microscopic metastasis; however, at what value would that be to the decision of whether or not to proceed with surgery? Likely minimal, given that the possibility of a bleed *dominates* the surgery decision. The decision for surgery is said to be *dominated* by the bleed possibility over the nature of the mass (benign vs. malignant) and hence surgery is recommended (base rates aside) regardless of whether the mass ends up being benign or malignant. Many times we spend countless hours grappling over inputs to a decision that in the end do not matter as these elements are dominated by other criteria.

The above is just one example of how certain decision criteria might dominate others in particular situations. As another example, have you ever witnessed the perfectly built home, but in the wrong location? A true majestic masterpiece of a home, but in such a poor location! *The house fails to sell because the location decision dominates the favorability of the home.*

In statistics, decisions you make along the way in your analysis dominate others. For example, if you build your entire analysis on the arithmetic mean as a basis of central tendency, when in fact the median would have been a more accurate portrayal of the data, even if the analysis is quite complex and detailed this crucial decision dominates all others. If you build on a questionable foundation, strong as the edifice may appear, it will be structurally unsound. Foundations first! Newton's theory of physics, impressive as it is, Einstein found was built on incomplete foundations. Critical questions about foundations are what lead to scientific and other revolutions.

## 1.12 All Knowledge, Scientific and Other, Is Tentative

As a final point regarding our review of scientific principles, we should emphasize the fact that all knowledge is tentative pending further development or discovery. Within the realm of science, this idea is well known. Perhaps the clearest example of this is with Newtonian physics. It was "correct" years ago, but now is less correct given Einstein's theory of relativity. Likewise, aspartame used to not be considered a carcinogen, but only recently has it been classified as possibly one. What is true scientifically yesterday is no longer today. This is how science functions and is the norm. Scientific "facts" are always *conditionally* or *tentatively* true based on further evidence.

Surely then, statistical and mathematical knowledge must be unconditionally always true, right? No! While at first glance it may appear that such knowledge is steadfastly certain, history has shown us differently. Perhaps the most alarming example of this was with Euclidean geometry and the discovery of non-Euclidean geometries. Euclidean geometry was *for certain* the only geometry for over a couple thousand years. Had that "fact" not been considered tentative by very bright minds who investigated foundations while all others busied themselves in further building *on* the

edifice of Euclidean geometry, we would have never come to know that Euclidean geometry was not a “fact” of mathematics. Likewise with theorems of mathematics or statistics, a proven theorem is generally considered true and timeless, but errors in proofs are regularly found such that it makes them less certain and enduring almost overnight.

The moral of the story here is to always hold knowledge, whatever the kind, as *tentative* given more information. One might *act* as though such knowledge is true, while at the same time being open to revising one’s opinion of such knowledge if new information contradicts it. The earth is currently known to be approximately round, or so, but if tomorrow’s discovery provides contradicting information we must have an open mindset such that we are willing to revise our beliefs. Likewise, climate change is a theory, not a fact. As new evidence enters into the equation we must be prepared to revise our belief in climate change. As we will see, this way of thinking has very strong Bayesian undertones.

### 1.13 Occam’s Razor

Occam’s razor is the idea that if there are competing theories to explain data, the simplest explanation shall be preferred over the more complex assuming that each competing theory accounts for the data equally well. While the principle is usually appealed to be based on intuition and plausibility (i.e., it sounds like a *reasonable* position to take with regards to theory-building), the principal can actually be more rigorously theoretically defended (Stoffi et al., 2022). Occam’s razor is far from being a statistical or machine learning concept alone as we can apply the principle even informally to everyday life.

One area where Occam’s razor is especially applicable as it relates to machine learning models is when constructing a neural network where as we will see, one of the decisions faced by the user is the number of hidden nodes and layers to include in the network. All else equal and given that competing neural network constructions result in more or less the same predictive power, Occam’s razor requires keeping the model as simple as possible while still maintaining its ability to generalize well. Hence, if a neural network with a single simple hidden layer does as well as one with multiple complex layers, the simpler model is to be preferred. However, as Einstein (and others) reminded us, *everything should be made as simple as possible, but no simpler*. A “too simple” of an explanation or theory that hinders explainability is not optimal either.

Theoretical explanations can become quite complex in science and industry. Pragmatic and simple explanations are often better. Why does a stock go up in the stock market? Yes, we can consider the multidimensional and multifactorial economic influences and market forces, yet on a more descriptive, pragmatic level, stocks go up for primarily one reason and one reason only: *more people than not are buying it*. Of course, such isn’t much of a theory as to *why* more investors are buying the stock, but it is the simplest and most practical explanation regarding why the stock is increasing.

#### 1.13.1 Parsimony vs. Complexity Trade-Off

Choosing the simplest theory or model is not the only priority of the scientist or quantitative modeler. Even if a very complex model explains the data better than a simpler one, it does not necessarily imply the more complex model should be chosen. For example, in a multiple regression model one can make the model as complex as one desires by adding more and more predictors. As is well known,  $R^2$  for the model will necessarily increase given the inclusion of an increasing number of predictors even if the addition in variance explained (greater than zero) by those predictors is negligible. Hence, a model with 1,000 predictors will have a higher  $R^2$  than a model

with 100, but variance explained should not be our only criterion for model selection, especially if the purpose is scientific explanation. If the goal is simply raw prediction and the availability of such data on multiple variables is easy to come by, then choosing the model with 1,000 predictors is the correct choice. One example of this might be *Amazon* collecting numerous predictors from online purchases that add to its ability to predict your next purchase. Choosing the simpler, more parsimonious model is not priority given the ease by which such data are collected by Amazon's virtually unlimited storage and upper-limit computing power.

However, if we are trying to explain the behavior of migrating birds, for instance, building a theory on 1,000 predictors may be unwieldy if not impossible. The complexity of the model is too rich, and hence simplifying the model, even at the risk of lowering predictive power, may be called for. From a scientific perspective, *the goal of modeling is rarely to generate models that fit perfectly, but instead to explain as much as possible using the least amount of information as possible.*

These ideas motivate the use of *penalized* model fit statistics that attempt to strike a balance between providing a good fit while minimizing the number of predictors providing that fit. In this book we will survey a number of penalized versions of statistics and unpack them to see exactly how they work.

### 1.14 Overfitting vs. Underfitting

In continuing our discussion of theory and model-fitting, we note there to be a delicate balance between *overfitting* versus *underfitting* a theory to data. The general idea behind overfitting is when a model is tailored too specifically to the training data and will likely not generalize well to the test data. At the opposite extreme, *underfitting* is when it is not specific enough to the training data. The goal of statistical inference and machine learning is that of *generalization* and hence both overfitting and underfitting have consequences. Not surprisingly, the magnitude of these consequences is related to the sample sizes used in such generalization or lack thereof. If the sample size is too small and the complexity of the hypothesis class too rich, overfitting is almost guaranteed to occur. On the opposite end, for increasingly larger sample sizes, a lack of complexity in a model will be more prone to underfitting.

Consider a very simple conceptual example of over vs. underfitting. Suppose you are a therapist attempting to identify the issue with your client's daily anxiety and elevated heart rate. After referrals to medical practitioners, you have successfully ruled out any biological issues. You suspect however that the "jitters" have to do with the amount of caffeine intake by the client. After asking the client to change their caffeine intake over the next while, you theorize that their anxiety is a function of having exactly 4.357 cups of coffee per day. Perhaps you have good reason to theorize that exact amount for this *particular client* after experimenting with various caffeine intake amounts. That is, your theory "fits" the client extremely well and seems to explain his behavior. However, as a general theory, how well will it do? Suppose you take your results and seek to generalize to future clients, telling them, "You must drink less than 4.357 cups of coffee per day." Of course, common sense tells us that the theory built on the specific individual will not generalize to future clients, at least not very well.

What has happened here from a theoretical point of view is that you have *overfit the data*. You paid too much attention to *particulars* of the given individual (sample data) and likely capitalized on random noise in the individual's specific caffeine sensitivity, so that when you attempted to generalize to new clients, the theory failed miserably as essentially a *point prediction*. How might you change the theory so that it generalizes better? You might instead fit a theory such as "Moderate intake of caffeine is predictive of anxiety level." This theory will likely generalize

better to new individuals (i.e., to new data) because it no longer relies as heavily on the specifics of the given individual you are dealing with.

On the opposite end of things, consider what *underfitting* would look like in this case. Suppose you tested the theory on your client as to simply *not drink hot beverages*. That is, you theorize that if your client avoids hot beverages altogether, such will have the effect of decreasing anxiety. Of course, that's far too *general* of a theory; you must make it more specific. You have *underfit* the data. Murphy (2022) gives an even simpler example using a coin-flipping paradigm. Suppose you flip a coin 20 times and obtain 15 heads on the 20 flips. The maximum likelihood estimator resulting from this exercise would be heads for future flips. But clearly, this *overfits* on the training data as most likely all future flips will not be all heads. That is, we overfit the theory on too small a sample of data.

The above examples can be generalized easily and more broadly to statistical and machine-learning problems. Consider the data in Figure 1.3 (1), a bivariate plot of points.

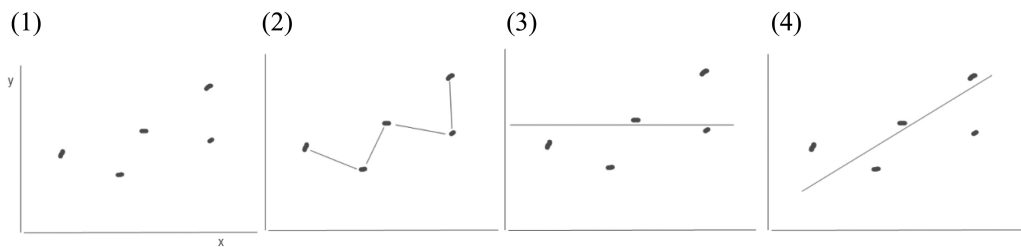


Figure 1.3 (1) The first plot of  $y$  on  $x$  is a scatterplot of points. (2) The second plot connects the dots of all points, an instance of overfitting. (3) The third plot underfits the data by simply drawing in the mean of  $y$ . (4) The fourth plot is the approximate line of best fit, which is a compromise between overfitting and underfitting.

Consider what overfitting would look like in this case. We could simply connect the dots (Figure 1.3, (2)). That is, we are modeling the theory (curve) too closely to the training data. What would underfitting look like? Suppose we simply modeled the mean (Figure 1.3, (3)). This is an example of underfitting as it essentially ignores the richness of the data. A nice compromise would be the line of best fit, typically the regression line (Figure 1.3, (4)). The regression line provides an *inexact* fit, but the theory here is general enough to possibly generalize to new test data.

In all of the models we refer to in this book, the more *specific* and *detailed* the model fit to data, in general, the more likely it is to overfit. *Do not be in a hurry to produce a model that fits the (training) data perfectly. Any such model can easily be built! The interest in model/theory-building is to produce a model that not only fits the current data reasonably well, but that also generalizes to new data.*

Of all the models we will study in this book, models such as neural networks are especially prone to overfitting. Why? Because, as we will see, due to allowing complex nonlinear relationships, they tend to capitalize on small perturbations in the data that may fail to generalize to new data. This is one reason why, despite the meteoric advances in model complexity in the statistics and machine learning literatures, classic linear models are still often preferred over more complex models in many contexts.

### 1.14.1 Solutions to Overfitting

There are usually a variety of ways to deal with and cope with overfitting. Many depend on the specific method used. Sometimes simply taking averages of predictions made by a number of different models fit to the training data is sufficient for minimizing the effects of overfitting. As we will see later in the book, such is the approach essentially advocated by such procedures as *bagging*.

Perhaps the most common strategy to deal with overfitting is simply to *regularize* a model. Regularization is a general technique in which, all else being equal, simpler models are favored over more complex models and hence in general is in the spirit of Occam's razor. This often translates to adding a penalty term to the loss or empirical risk function. We now briefly discuss regularization and then develop it at a deeper level later in the book.

### 1.14.2 The Idea of Regularization

The idea of *regularization* permeates several models in statistics and machine learning and we will see this principle applied regularly throughout the book. The *principle of regularization*, or shall we say the *spirit* of it, is much more general than the specific instances in which it is used or the individual method to which it is applied. If the principle is understood, then every time regularization is encountered, even in different forms, the reader will more easily understand it despite the precise details of how it is being applied in the given context. But what is regularization exactly? Probably the most general idea of regularization is the following:

*Regularization simplifies models to reduce the probability of overfitting.*

By guarding against overfitting, *generalization error* is reduced (Goodfellow et al., 2016). Regularization helps *stabilize* a model, and stable learners (or models) are less likely to overfit. If small changes to input data result in large changes in outcomes, as already discussed, the model is regarded as *unstable*. As a human analogy, a good friendship can withstand turbulence; that is, the outcome does not change in the face of adverse events. The friendship still remains strong. A weaker friendship is easily changed based on an input that breaks the friendship apart.

Most often in practice, regularization involves adding a penalty term to the maximized or minimized objective function (Murphy, 2022). Early stopping of an algorithm also constitutes an attempt to regularize a model. Other times, simply collecting more data on which the model is applied is a suitable solution. While a polynomial may overfit say, 20 data points, it is unlikely to overfit 200 or 2,000, and the generalizability of the model will often improve.

## 1.15 The Measurement Problem

In the age of sophisticated algorithms and terabytes of data, it can be all too easy to pay insufficient attention to *what the data actually consist of*, that is, what they actually represent. Sure, we all desire novel algorithmic fancy and modern ways of analyzing such data, *but what if the data are not representing that which we sought out to measure?* What if that number in the spreadsheet does not correspond to what we believe it represents? If the data point does not represent what we sought to measure, no sophisticated statistical or machine learning algorithm will save it. What the data point *is a function of* can be easily taken for granted, an assumption that what we sought to measure we are in fact measuring. This can be a very dangerous assumption. In this section, we briefly survey some issues related to the larger problem of measurement in the social and natural



sciences and discuss some of the more common ways of thinking of scales of measurement. We then very briefly discuss Likert scale data.

### 1.15.1 *What Is Data?*

To begin a discussion of measurement and reliability and validity, we should first ask ourselves what *data* are in the first place. At a philosophical level, which as mentioned virtually *everything* boils down to sooner or later when scrutinized enough, data is simply *information*. In the age of computerized data analysis, we of course associate a piece of data with a number in a spreadsheet, but it should be recognized that that numerical representation of data is simply a special case of the wider possibility of what data *could be*. For example, learning that your friend is not trustworthy is a piece of data. Looking up in the sky and seeing a sunny day is data. Your internet connection going out momentarily is yet another piece of data. The first to walk the planet were data analysts, not very good ones we might argue from our modern point of view, but data analysts just the same. What distinguishes today's data analyst from centuries ago is that today we are more *systematic* and *precise* in how we go about defining, storing, and analyzing data.

### 1.15.2 *The Philosophy and Scales of Measurement*

Measurement can be interpreted as the assignment of numbers to objects via a set of rules. Crudely, if I am much taller than you, measurement implies that we have a rule that says I should get a higher number (measurement) than you and preferably by a well-defined amount on precisely measured units (e.g., inches). The problem with measurement, at a philosophical level, is the extent to which what we are actually measuring is a reflection of the true score we wish to measure. Make no mistake, the score we obtain with whatever instrument we use is but a *measurement*. It is not a *pure* reflection of the true score. In measurement theory, the true score  $T$  is a function of the observed score  $O$  plus error  $E$ :

$$T = O + E \tag{1.2}$$

To cut right to the chase, the measurement of something like height or weight contains more  $O$  than  $E$ , whereas the measurement of something like personality or intelligence, likely, or at minimum *one could argue*, contains more  $E$  than  $O$ . The key to (1.2) is that just because you measure something this does not necessarily imply that you are tapping into the true score. Much of your measurement may simply be *error*. The measurement of temperature with a dysfunctional thermometer may be more error than actual temperature, for instance.

### 1.15.3 *Reliability*

Measurement of data is only as good as such a process is reliable and valid. Reliability generally refers to the idea of consistency just as it does in ordinary speech. Informally, a measurement process is reliable, in one sense, if repeating the measurement produces the same or at least *similar* recording if the attribute under consideration has not changed. For example, measuring your height today and then tomorrow should provide the same (or at least nearly so) measurement. Such a measurement process is deemed reliable if it does. If your height measurement today was 60 inches and tomorrow 65, and assuming you didn't experience a growth spurt overnight, the measurement process would be deemed quite unreliable.



Following McDonald (1999), we assume that the population correlation  $\rho$  between  $T$  and  $E$  is equal to 0, that is,  $\rho_{TE} = 0$ . By the *variance-sum law* which holds that the variance of uncorrelated variables can be summed, the variance of  $Y$ , now the observed score, is a sum of the variances of both  $T$  and  $E$ , that is,

$$\sigma_Y^2 = \sigma_T^2 + \sigma_E^2 \quad (1.3)$$

From (1.3) we can deduce that  $\sigma_T^2 \leq \sigma_Y^2$  and  $\sigma_E^2 \leq \sigma_Y^2$ , which implies

$$\rho = \frac{\sigma_T^2}{\sigma_Y^2} = \frac{\sigma_T^2}{\sigma_T^2 + \sigma_E^2} \quad (1.4)$$

and where it can be easily shown that  $\rho$  is bound between 0 and 1 where  $\rho$  is called the *reliability coefficient* of  $Y$ . There are different forms of reliability. We survey one of the most popular ones now, that of internal consistency.

#### 1.15.3.1 Coefficient Alpha

The split-half reliability is the correlation between two halves of a test corrected to full length by the Spearman-Brown prophecy formula (Cortina, 1993),

$$r_{sh} = \frac{2r_{12}}{1 + r_{12}} \quad (1.5)$$

where  $r_{12}$  is the Pearson correlation between the two halves. Cronbach's alpha is equal to the *mean* of all split-half reliabilities assuming item standard deviations are equal (Cortina, 1993; Sijtsma, 2009). The package *coefficientalpha* (Zhang & Yuan, 2023) can be used in R for carrying out reliability analyses. An example from the package's documentation yields the following data on ten items:

```
> library(coefficientalpha)
> data(example)
> head(example)
  item1 item2 item3 item4 item5 item6 item7 item8 item9 item10
1 -0.850 -0.503  0.617 -1.002 -0.861 -0.568 -1.847 -0.406 -1.437 -1.405
2  1.123  0.077 -0.532  0.812  0.835 -0.795  0.910  0.577  1.504  1.036
3 -2.391 -0.364 -1.377 -0.610 -0.797 -0.927  0.056 -0.634 -0.863 -0.748
4 -2.455  1.947    NA -2.484    NA -1.520  0.764  2.049 -2.561  1.982
5 -0.472 -0.549 -0.360  0.110  0.253 -0.527  0.332 -0.218  0.200 -0.397
6  0.630  1.245  0.264  0.409  0.936 -1.086  0.745  0.583  1.228  0.338

> alpha(example)
```

The alpha is 0.915.

We see that alpha is relatively high at 0.915. In general, alpha values of 0.80 or higher are usually deemed acceptable. While alpha is regularly used, on a theoretical level there are problems with the coefficient and it can be especially prone to misunderstanding (Panayides, 2013; Sijtsma, 2009).

#### 1.15.3.2 *Test-Retest Reliability*

Test-retest reliability generally refers to the consistency or stability of a measure across two or more time points. It is generally assessed using the correlation coefficient. For example, suppose we administer a test at *time A* then correlate that to test results at *time B*. To the extent that the correlation is strong indicates that reliability is likewise tenable.

Special software functions are usually not needed to assess test-retest reliability as it can easily be evaluated using a Pearson (or other, such as intraclass) correlation coefficient.

#### 1.15.4 *Validity*

The validity of a measuring instrument is an assessment of whether that instrument measures what it is designed to measure. For instance, a tire pressure gauge that reports the correct tire pressure is a valid measure of tire pressure. Flight instruments on an airplane are valid if they report correct readings of such things as speed, altitude, and so on. An intelligence quotient (IQ) test is considered valid if it actually measures something called “intelligence.”

How do we know altitude readings on an aircraft measure something called *altitude*? We know because engineers correlate such readings to measurement instruments having readings that are already known to measure altitude. Hence, when a new altitude measure is developed, it is a fairly easy matter to correlate it to well-known measures. Likewise with a temperature gauge, you can be quite confident that you have 36 psi (*pounds per square inch*) in your tires measured by your new gauge if your old and reliable gauge also measures 36 psi.

The measurement of IQ or similar psychological constructs is not as easily validated because, for one, there is no agreement on not only *how* IQ is best measured, but also whether even something called “intelligence” exists! It can be exceedingly difficult to validate something for which there is no agreed-upon criterion such as speed, altitude, or tire pressure. Psychological sciences have attempted to define IQ rigorously, but such definitions ultimately come down to what experts in the area wish to call *intelligence*. Of course, the criteria are not simply “made up” or contrived. Far from it. Experts spend a great deal of time pondering over what kinds of items reflect processing ability and such, and hence IQ tests are often based on rather strong theoretical underpinnings. Despite this, “capturing” the essence of IQ with a psychometric test that features subjective responses some believe to be an impossible task. If, however, we can agree on a particular test as a *criterion*, then validating newer instruments can be correlated to the original criterion, just as the aforementioned altitude readings can be correlated to more well-known readings.

It should be emphasized as well that at a philosophical level, the criterion for something like IQ and that of altitude is *always* human judgment. We are the ones who “invented” the idea of altitude, just as we are the ones who invented the idea of intelligence. Of course, that may be exaggerating the point slightly, but it is simply to say that *constructs are always constructed*. It just so happens that some constructs are extremely well-agreed upon and have lasted centuries without much disagreement, while others (such as IQ) are more novel and their existence is even subject to debate.

There are different kinds of validity which all more or less rely on the correlation coefficient in some form or another in the establishment of evidence. For instance, if an instrument has *convergent validity*, it should correlate highly with other tests purported to measure the same criterion,

whereas if the instrument has *divergent validity* then it should not correlate with tests that purport to measure different constructs. For an excellent overview of validity, see McDonald (1999).

### 1.15.5 Scales of Measurement

Stevens (1946) proposed four scales of measurement: *nominal*, *ordinal*, *interval*, and *ratio*. We let  $o_i$  represent the object we wish to measure. The measured object is denoted by  $m(o_i)$  while the true object is denoted by  $t(o_i)$ . Nominal scales are not true measurement at all, since they simply assign a different label to each object as merely an identifier. That is,  $m(o_i) \neq m(o_j) \Rightarrow t(o_i) \neq t(o_j)$ . The classic example of nominal scales is that of football jersey numbers on the backs of football players. That one football player has a different number than another simply implies that it is a different football player and nothing more. The numerical assignment certainly does not imply that one football player is “better” than the other.

Ordinal measurement also carries the implication of nominal measurement,  $m(o_i) \neq m(o_j) \Rightarrow t(o_i) \neq t(o_j)$ , but now with an added *magnitude* property. That is, if  $m(o_i) > m(o_j)$ , such implies that  $t(o_i) > t(o_j)$ . Ordinal measurement is the first instance of what we regard as measurement instead of simply assigning numbers to different objects as in the nominal case. Be sure to note, however, that  $m(o_i) > m(o_j)$  does not imply *how much*  $t(o_i) > t(o_j)$ , it only implies that  $o_i$  has *more* of the quantity than  $o_j$ . Hence, that you finished first in the race, me second, and our mutual friend third tells us nothing about the magnitude or distance between first and second vs. second and third place. It only tells us the order, hence the name *ordinal*. To assume distances between objects to be equal, we require interval measurement, which we discuss next.

The interval scale is characterized by being an ordinal scale but with the added quality that for all distances  $i, j$ ,  $m(o_i) - m(o_j) = c$ , where  $c$  is some amount or constant, implies  $t(o_{i+1}) - t(o_{j+1}) = c$ . That is, exact distance between objects is a characteristic of the interval scale. As an example, the distance in temperature on the Fahrenheit scale between 0 and 10 is the same *amount* of temperature change as the temperature change from 10 to 20, at least measurably.

The ratio scale has all the properties of the interval scale but with an important addition. Zero on the ratio scale implies *zero* of the quantity we are working with. That is,  $m(o_i) / m(o_j) = c$  implies  $t(o_i) / t(o_j) = c$ . The only adjustment we need to make to express the scale as ratio distinct from an interval one is to denote the origin as a *true zero point*. On the interval scale, zero is simply an arbitrary point; that is, it is just another point on the scale. That it is zero carries no special meaning. On the ratio scale, however, *zero means zero*.

If a scale is ordinal, then it necessarily possesses characteristics of a nominal scale. If a scale is ratio, it necessarily possesses characteristics of an interval, ordinal, and nominal scale, and so on for all scales in between. For example, temperature on an interval scale simultaneously implies a scale weaker than interval such that a temperature of 10 degrees is more than a temperature of 5, which is characteristic of an ordinal scale. We leave it to the reader to think up more examples. The idea that more primitive scales (e.g., nominal) can be considered special cases of more advanced scales (e.g., ratio) is referred to generally as *collapsing scales*.

### 1.15.6 Likert Scales

Likert scales were invented by Likert (1932) and are designed to elicit subjective measurement from individuals on a scale arbitrarily constructed by the researcher. In sciences such as psychology and sociology, measuring attributes of interest can be extremely difficult; some would even

say impossible in many cases. For example, how can we measure how much you like pizza? We could observe how much you like pizza by following you around to see how much you eat of it every day, but such would be too timely and obtrusive. A more efficient way, if less valid, may simply be to *ask you* how much you like pizza and numerically scale your response. This is an example of a Likert scale.

Likert scales are a useful attempt to measure what we want to measure, but to be precise, they are not actually measuring the question in the scale. Instead, they are measuring the individual's *response* to the question. For example, when we measure the person's opinion on the overall quality of healthcare, their actual opinion might differ from the response they are providing. Their *response* and their *actual opinion* are not a one-to-one guarantee. This is the first problem with Likert scale data.

Second, the type of Likert scale can have a great influence on the accuracy of measurement. It is not difficult to come up with examples where participants will either be too *optimistic* or *pessimistic* on a particular item. It is equally easy to come up with examples where a participant may purposely mislead on the particular item. Human beings are not always honest, and if they are, they likely are not aware of their biases. A wealth of psychological research has been done in support of this idea. Asking a participant their level of anxiety is a *subjective representation* of their level of anxiety. *There is no guarantee that it is their actual anxiety.* This is especially true since nobody really knows precisely what anxiety is in the first place! We think it *exists*, but we can't "point to it" as we can something like height or weight.

Hence, so long as the weaknesses and drawbacks of Likert scale data are recognized, they can be useful, but they need to be used and interpreted with care. Research has also been performed to help determine whether numerical distances on Likert scale data actually correspond to psychological measurement. See Wakita et al. (2012) for details.

#### 1.15.6.1 *Statistical Models for Likert Data*

How should Likert data be analyzed? A first thing to note regarding any data is that the numbers do not know where they came from nor how they should be treated. *Scales of measurement are ideas we impose on data.* Hence, a data set can be analyzed in any way we wish, though the analysis will only be reasonable if it conforms to properties we impose on or assume of the data.

*Categorical data methods* are usually most appropriate for data at the nominal level since numbers in the nominal scale most often represent categories. Chi-squared analyses and log-linear models are well-suited for these types of data. Data that are at the ordinal level of measurement have more options for analysis. For one, we could simply treat the data as interval or ratio, drawing conclusions quite cautiously. Or, we could adopt methods specifically targeted for ordinal data, which we now briefly discuss.

#### 1.15.6.2 *Models for Ordinal and Monotonically Increasing/Decreasing Data*

For two variables, continuous or not, *Spearman's  $\rho$*  (pronounced "rho") measures the monotonic relationship between variables where the relationship can be monotonically increasing or monotonically decreasing. This is in contrast to the classic Pearson  $r$  which assesses linearity. As depicted in Figure 1.4, Spearman's  $\rho$  only cares that data on  $x$  are increasing or decreasing with data on  $y$  and does not require linearity or any other parametric structure.

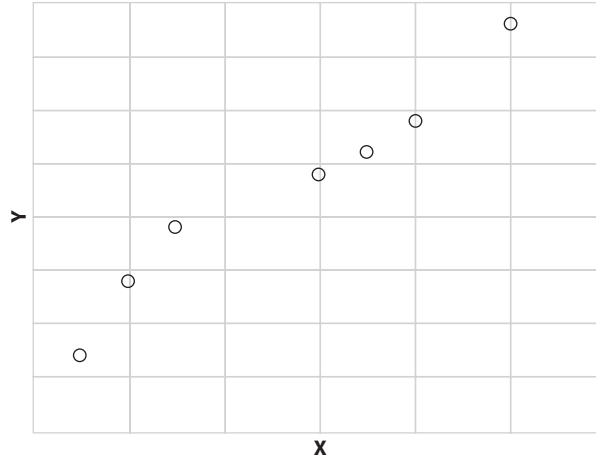


Figure 1.4 Positive monotonically increasing relationship. As values on  $x$  increase, they are associated with increasing values on  $y$ , however the relationship is monotonic; it need not be linear or any other parametric form.

Other tests that can be used with ordinal response variables include *Wilcoxon-Mann-Whitney* for two groups and the *Kruskal-Wallis* test for multiple groups as a counterpart to parametric ANOVA. The *sign test* can be used for nominal or ordinal data and the *Friedman test* is also a useful alternative for repeated-measures data. Space does not allow us to discuss nor demonstrate these tests here. For details, see Siegel and Castellan Jr (1988) which is a classic resource on non-parametric tests. For more cohesive models, *ordinal logistic* and the *ordered probit model* can be used. If the odds in the model are proportional to one another and differ only by a constant, this gives rise to the *proportional odds model*. For details, see Fox (2016, pp. 401–402).

Another option for modeling monotonic data is that of *isotonic regression*, which in contrast to ordinary least-squares regression specifically models data as a non-decreasing approximation. See Neykov (2019) for details. Isotonic regression for the two-variable case, that is, *bivariate isotonic regression*, can be carried out using the R package *Iso* (Turner, 2023).

## 1.16 Overview of Statistical and Machine Learning Concepts

Having surveyed some fundamental philosophical assumptions and ideas underlying science, statistics, and machine learning, we now turn to surveying statistical and machine learning concepts themselves. Many of these concepts have long histories in statistics where others are more modern and apply mostly to machine learning. Topics surveyed here are selective; they are the ones the reader should be familiar with in order to understand the rest of the book.

### 1.16.1 Probably Approximately Correct

The idea of *probably approximately correct (PAC) learnability* arose from statistical learning theory for the purpose of identifying hypothesis classes or *sets* (usually infinite in nature) that are actually learnable and provide low generalization error (Haussler, 1990; Valiant, 2013). Hypothesis classes are essentially subsets of virtually all functions that *could* be selected in modeling some phenomenon. For instance, suppose we select a hypothesis that minimizes empirical risk on a

sample. How do we know such a hypothesis will also minimize population risk? If the hypothesis class is so-called *PAC learnable*, then we know it will probably also minimize population risk. That is, *PAC learnability tells us that the selected function will, essentially, likely have low generalization error*. That is, a PAC model will *probably* generalize well to new data.

A more direct and pragmatic way of understanding PAC learnability is to consider how well a hypothesis or concept class will *not* predict. That is, any hypothesis that is seriously wrong will almost certainly be “discovered” with high probability after a small number of examples or data points in a sample because it will (very soon) make an incorrect prediction. Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong. That is, it has been “tested” via experience, which implies it must be *probably approximately correct*. Any learning algorithm that returns hypotheses that are probably approximately correct is called a *PAC learning algorithm*, and such a designation can help provide bounds on the performance of various learning algorithms. For a precise and technical characterization of PAC learnability, see Mohri et al. (2018, p. 11).

### 1.16.2 *No-Free-Lunch Theorem*

As mentioned at the outset of this book, one element that distinguishes the machine learning paradigm from that of classical statistics is a focus more on the *algorithm* or *method* than the data. That is not to say the data are not important; they most certainly are of course, but the *performance* of the algorithm or procedure is what is studied most. Questions such as “Does logistic regression outperform discriminant analysis, and if so, under what conditions?” dominate the current machine learning zeitgeist.

Given this emphasis on the algorithm, it makes sense to ask then whether one algorithm dominates all others. That is, we ask the question:

*Is there a best universal learner?*

The answer to this question is contained in what is known as the *No-Free-Lunch Theorem*, which essentially holds that *there is no universal learner*. There is no “best” algorithm under all conditions for all data. Unpacked, what this implies is that for any learning algorithm there is a data set for which it may be very accurate and another data set for which it is very poor. The assumptions we enter with in one substantive situation, often called the *inductive bias*, are not the same as in another domain, and hence a universal learner across all domains does not exist. When we say a learning algorithm is “good,” we are only quantifying how well its inductive bias matches the properties of the data we are currently working with (Alpaydin, 2020). However, as good as the learner may be in one setting, there is always a task in which it will fail.

As a simple example, linear regression may work wonderfully in predicting income based on education levels but fail miserably in predicting COVID-19 probabilities of death. An equivalent and more crude interpretation of the theorem is that for even the most sophisticated algorithm one can conceive, over all conceivable possible tasks it will have the same *average* performance as merely predicting that every point belongs to a particular class (Goodfellow, 2016). Why will the performance of an algorithm chosen to model a specific situation likely be better? Because we will have chosen that algorithm for the specific task at hand coupled with our knowledge of the data. In other words, we are no longer averaging across all tasks. We are selecting a specific task and using an algorithm well-suited to that task.

As an analogy, modeling IQs across all possible theoretical distributions is hopeless and if we combined (or averaged) all distributions, our model of IQ would be quite poor. Fortunately, we do not employ all distributions; we usually employ a *normal distribution* because it best approximates

the empirical shape of the distribution of IQ. The normal distribution isn't the "best" distribution for all tasks; it is the best for particular tasks such as the modeling of IQ. That is, we "narrow down" from all distributions by making *assumptions* regarding the task we are working with. This is a powerful way to conceive of assumption-making when it comes to fitting statistical distributions – *assumptions help narrow down the search and selection of an algorithm*.

Think of this next time your chosen algorithm fits well to a given data set and be sure to not exaggerate how "good" your algorithm actually is. There is no *best* algorithm for all circumstances. Algorithms and statistical procedures must be chosen wisely for each task, which implies knowing not only something about the data but also something about the substantive area to which it is being applied. Linear regression, for instance, can never be the "best algorithm." It can only be the best coupled with a particular data set which suits it well (e.g., one that exhibits linearity and homoscedasticity of variances).

### 1.17 V-C Dimension and Complexity

The Vapnik-Chervonenkis or "V-C" dimension is a measure of a classifier to perform. It characterizes its *learnability*. It is the maximum number of points that can be "shattered" or less formally, "separated" by a hypothesis class. As a simple example of the theory, consider the following plots (adapted from Hastie et al., 2009, p. 238), where each point takes its turn as class = 1 (bolded square) vs. class = 2:

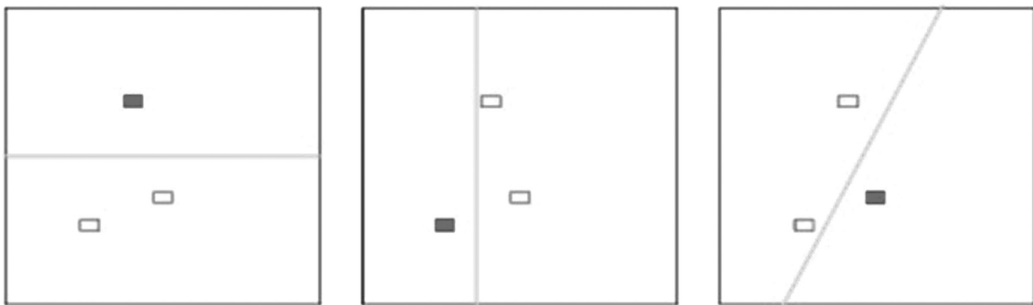


Figure 1.5 Used with permission of Springer Nature BV, from The Elements of Statistical Learning by Trevor Hastie et al, © 2009; permission conveyed through Copyright Clearance Center, Inc.

As shown above, we can see that a linear function is able to "shatter" the two-group structure in each case. Hence, we would say that the class of linear functions for the above is able to shatter three points.

The above is but a trivial example of V-C dimension, as its theory gets much more complex. The details of the theorem can be found in Vapnik and Chervonenkis (1971). The theorem effectively can help provide an upper bound on *optimism* of a set of algorithms (Hastie et al., 2009) and tells us the capability or capacity of a concept class, that is, what the class is *able* to do, such as in the above example of lines shattering three points. A learning machine with a high number of parameters in general has a high V-C dimension; however, this is not necessarily always the case. One could have a very wiggly line that shatters many points yet only has a single parameter (see Hastie et al., 2009, for an example).

### 1.18 Parametric vs. Nonparametric Learning Methods

Virtually all statistical and machine learning approaches can be distinguished between parametric vs. nonparametric, though in reality many approaches will fall in between these two extremes. What distinguishes a parametric vs. a nonparametric approach? As with most things in statistics, the distinction quickly becomes *fuzzy*, but we will do our best to delineate the primary features.



*Parametric* methods make assumptions about the functional form *thought* or *assumed* to underlie the observed sample data. *Nonparametric* methods do not assume such a form, but instead try to estimate the functional form more directly from the observed data. In a sense then, we can generally say that parametric models *arrive on the scene* with a Bayesian prior of sorts about the kind of distribution and function that may have given rise to the observed data, whereas nonparametric methods generally simply want to “get to know the data” without prior beliefs or functional form assumptions getting in the way. This is an oversimplification of the distinction, but still useful as an approximation.

Why make parametric assumptions at all? Why not just let the data “speak?” Making parametric assumptions often facilitates the analysis so that one does not have to start all the time at the very “beginning” as though one knows absolutely nothing at all about what functional form might have given rise to the data. As a result, *parametric approaches typically require less data than do their nonparametric counterparts*. For example, if you already have a good idea that the parametric form of the relationship will be linear, then you will need less data to fit your model than if you allowed the functional form to be entirely determined by the data, which would entail a more nonparametric approach. *There are usually benefits to being able to assume a prior functional form to data. However, if the assumption is incorrect, accuracy and validity of the analysis will be compromised.*

Another way of appreciating this difference is with respect to the number of parameters required in each type of modeling. Using the linear model as an example once more, once we fit the intercept and slope, the job is done, regardless of how much data we are fitting the line to. Two parameters get the job done regardless of whether we fit the line to ten observations or 10,000. The parametric line for a smaller sample size keeps the data from “speaking too loudly,” which is reasonable, since there isn’t much data to begin with. In other words, we are, through our imposing of a parametric form, *helping the data determine its underlying form*. However, as sample size increases and we get more data, we begin to “trust” the data more with determining its own underlying structure, and hence are more likely to allow it to “speak for itself” than restrict what it can say via a smaller *allowance* of a parametric form. After millions of observations, if the data suggest the underlying form is a very complex polynomial, we will have more trust than if the data were quite small in number. When the data were smaller, we imposed a line (or some other parametric form) so that the conclusions do not get too unrealistic by perhaps “connecting the dots” in the smaller sample.

### **1.18.1 Flexibility and Number of Parameters**

Another and very useful way to think about the distinction between parametric vs. nonparametric approaches is via *flexibility* and the number of parameters required to carry out the modeling. As we have said, parametric models (e.g., a linear model) have a fixed number of parameters coming to the modeling scene already packaged and ready to be fit to the data. In the nonparametric situation, however, the number of parameters needed for the modeling is theoretically and pragmatically unbounded. As we will see when we fit such things as regression splines, we can pretty much fit as many knots as we wish to best approximate the curve. We have greater *flexibility* in how we model the data when compared to a parametric approach. The effective number of parameters is essentially unbounded (James et al., 2013).

### 1.18.1.1 Concept of Degrees of Freedom

Degrees of freedom can be conceptualized in a variety of ways. All ways pretty much point to the same concept. For example, in a system of equations, degrees of freedom refer to the number of variables needed to complete the solution set, that is, if we know values for  $x, y$ , but not  $z$ , then the degrees of freedom for the system are equal to one. In a physical system, degrees of freedom refer to the minimum number of coordinates required to describe the state of the system (Tanton, 2005).

The basic sample variance as an estimator of the population variance is probably the simplest *statistical* illustration of this concept, where  $n-1$  is used as the denominator to express the loss of a degree of freedom in

$$s^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1} \quad (1.6)$$

The numerator  $\sum_{i=1}^n (y_i - \bar{y})^2$  features the estimation of  $\mu$  through  $\bar{y}$ , hence the loss of the degree of freedom.

More universally in terms of fitting models, the concept of degrees of freedom is related to the *complexity* of a model. In general, *the more complex a model fit to training data, the more degrees of freedom that model possesses* (e.g., spline with several knots or higher-order polynomial fit). A model with more degrees of freedom will generally fit better than a model with less, and hence overfitting becomes very probable as degrees of freedom increase (Murphy, 2022). An example is that of an exaggerated higher-order polynomial fit essentially at some point accounting for mostly noise in the empirical observations.

### 1.18.2 Instance or Memory-Based Learning

Nonparametric methods in the machine learning literature sometimes go by the name of *instance-based learning* methods or *memory-based learning* as a result of their heavy reliance on the data in determining fit. At a computational level, memory-based methods store the entire or at least much of the data set when making predictions about future data. As a result of this storage requirement, such memory-based methods are generally fast to train but can be slower to make predictions (Bishop, 2006).

### 1.18.3 Revisiting Classical Nonparametric Tests

If the reader has taken a classic statistics course, she may recall encountering nonparametric statistics, at least in a single chapter usually toward the end of the book. Nonparametric statistical tests included such things as the Kruskal-Wallis test and a host of others. Given our characterization of nonparametric tests above in the most general way, we can now revisit earlier teachings on nonparametric tests as traditionally and “classically” taught. What was it about those tests that made them nonparametric? In line with our definition above, it was the fact that they presumably did not make parametric assumptions that made them nonparametric. Or, at minimum, made *fewer* of them. For instance, the Kruskal-Wallis test does not assume observations come from normal populations as the competing parametric ANOVA test does. These nonparametric tests often operate on *ranks* and such, and thus often make minimal, if any, assumptions about the underlying population.

The problem with this characterization of nonparametric tests is that the modern statistician versed in machine learning may not recognize those “simple” nonparametric tests as being examples of today’s nonparametric expanse largely due to increased computing power. Today’s nonparametric methods can seem much more vast, much more elaborate, much “bigger” than those simpler nonparametric tests of the classical age in statistics. When we say “nonparametric” today, we typically are not referring to those simpler tests, we are instead referring to things like the *bootstrap*, *splines* or other algorithms that make *fewer assumptions about the underlying population*, and for good pragmatic reason as well, such that in many cases of “Big Data” today we are closer to being able to approximate the entire population in our data recruitment anyhow. The key to today’s nonparametric methodologies isn’t so much about not making parametric assumptions, even though this still is their defining characteristic. The “spirit” of today’s nonparametric landscape is that of *letting the data speak for itself*. It’s about getting the parametric assumptions out of the way and replacing them with modern computing capabilities. Efron and Hastie (2016) summarize the newfound rise of nonparametric methods:

Two words [parametric and nonparametric] explain the classic preference for parametric models: mathematical tractability. In a world of sliderules [sic] and slow mechanical arithmetic, mathematical formulation, by necessity, becomes the computational tool of choice. Our new computation-rich environment has unplugged the mathematical bottleneck, giving us a more realistic, flexible, and far-reaching body of statistical techniques. But the classic parametric families still play an important role in computer-age statistics, often assembled as small parts of larger methodologies.

(p. 53)

As referenced earlier, a good resource on classic nonparametric tests is Siegel and Castellan Jr (1988).

### 1.19 Dimension Reduction, Distance, and Error Functions: Commonalities in Modeling

Most processes or data do not have thousands of intrinsic parameters that underlie them. Instead, there are usually only a few that govern things at a lower dimension. The lower-dimensional structure is said to be *embedded* in the higher-dimensional space that we come across in our work. That is, we say there is a *manifold* or *subspace* that “lives” within this higher-dimensional space. Dimensionality reduction seeks to reveal or uncover that manifold or subspace.

The search for essential dimensions has dominated science in one way or another since its infancy. Instead of thousands of “things going on,” there may be just a few things underlying the surface. At first glance, dimensionality reduction is quite naturally associated with techniques such as *principal components analysis*, *factor analysis*, etc., but the truth is that dimensionality reduction is implicitly present in virtually almost every technique we deal with. Even the basic *t*-test and its transformations (e.g., linear contrasts, ANOVA) can be shown to contain essential primitive concepts of dimension reduction, though at an elementary level of presentation this is not immediately apparent nor would it make sense to teach *t*-tests as related to dimensionality-reduction techniques in an introductory course.

#### 1.19.1 Dimension Reduction: What’s the Big Idea?

The idea of dimension reduction is to *project* data points onto lower dimensions. The general question posed by dimensionality reduction is the following:

*What would these data points look like if they existed in these lower dimensions?*

As we will see later in the book, this is basically what PCA is all about, assuming the underlying dimensions are linear functions. We may have 1000 variables, but when we reduce the dimensionality from 1000 to say, 3, we then want to know what the scores *would have been* on these 3 dimensions. This will be answered by projecting the data onto those three dimensions and then computing *component scores*, which are the scores yielded on those now three dimensions.

As an example of what we are talking about and to preview what lies ahead in the book, PCA can be defined most generally as

$$\ell_i = a_1 y_1 + a_2 y_2 + \cdots + a_p y_p \quad (1.7)$$

where  $\ell_i$  is a linear combination of weighted values of observed variables  $y_1$  through  $y_p$ . Even if the data are not truly on a linear manifold (we will define “manifold” later in the book; for now it may be regarded as simply a *shape* of sorts), PCA is often still useful and a very good *approximation* to the true underlying manifold. PCA, as we will see later, is essentially seeking to discover a new *basis* from which to project the data. A *basis*, as we will review in our survey of linear algebra, is *the maximum set of linearly independent vectors that spans the space*. When we collect data in its raw form, it is already on a basis but that basis is relatively *arbitrary*. For instance, the typical Cartesian plane, from this perspective, is an arbitrary basis but constrained at a 90-degree angle. Transforming the data to a new basis is what PCA and many other techniques accomplish. We then project the data from that basis to learn what the data look like from that *vantage point*, the vantage point being the new basis. The basis in the case of PCA is the set of orthogonal eigenvectors that result from the eigen decomposition (or singular value decomposition). The basis is orthonormal, which means orthogonal and each eigenvector having a length equal to unity.

What shall our dimensionality reduction look like? What is the form of these lower dimensions? We have two general possibilities: *linear* vs. *nonlinear* dimensionality reduction. While traditionally linear dimensionality reduction has been the norm, relatively recent techniques have allowed for sophisticated nonlinear reduction, or what is more generally called *nonlinear manifold learning*. While linear reduction may capture a process relatively well, in some cases a nonlinear manifold is required to capture more of the complexities and intricacies of the underlying manifold.

### 1.19.2 The Curse of Dimensionality

Given that many methods come down to dimensionality reduction, it is well worth noting at this point that dimensionality can be extraordinarily difficult to *tame*. Nonparametric methods, as we have discussed, often attempt to model data at a very *local* level. It would at first seem that these methods are well-suited for handling the dimensionality problem. However, as the number of dimensions increases, modeling local behavior becomes unwieldy. Hastie et al. (2009) describe the *curse of dimensionality* with reference to local regression, a technique that models the regression surface at a more local level than typically can be accomplished by ordinary least-squares:

Local regression becomes less useful in dimensions much higher than two or three ... It is impossible to simultaneously maintain localness (low bias) and a sizeable sample in the neighborhood (low variance) as the dimension increases, without the total sample size increasing exponentially ... When the dimension to sample-size ratio is unfavorable, local regression

does not help as much, unless we are willing to make some structural assumptions about the model.

(pp. 200–201)

To paraphrase, *locality* is a relative term and not simply a function of a distance metric. As we will revisit when we discuss *k*-nearest neighbors, as the dimensionality of the space we are working in grows, even to capture a relatively local neighborhood of that sizable dimensionality often involves covering a significant quantitative range of the variable(s) in the model. Considered another way, as dimensionality grows, it becomes virtually impossible to account for all possible “cells” in a multidimensional matrix, and hence the very definition of *locality* becomes less meaningful unless *sample size* grows substantially to account for the increasing volume of the space.

To use a *k*-nearest neighbor analogy that we will refer to when considering that method in more detail later in the book, if your *nearest neighbor* mathematically is a house on the planet Mars, such fails to convey the true spirit of locality. That is, it’s hard to argue a house on Mars is a neighbor of yours. It may be a neighbor *mathematically*, but such is only because of the huge dimensionality of the problem in which there is a necessarily insufficient sample size to account for all “cells” in the data from your house to the house on Mars. In other words, *that there is no house on the moon should not, by itself, make Mars, by default, a nearest neighbor*. Mars being a neighbor is more an artifact of the dimensionality of the space you are working in, not because Mars is truly a neighbor. Increasing sample size would eventually, and most likely, put a house on the moon, and the moon would be your neighbor now instead of Mars. In brief, as the dimensionality of the problem increases, that Mars may be your “nearest neighbor” becomes a much less meaningful statement than if we were to truly identify your nearest neighbors more *locally*, that is, in a smaller volume of space.

## 1.20 Distance

Perhaps even more rudimentary of a “unifier” of the methods and models of this book is that of *distance*. Every approach and method, whether in multivariate statistics or machine learning, in some way assesses distance between objects (Johnson & Wichern, 2007). For instance, “patterns” and “clusters” can only be discerned if we are able to measure distance between observations. Which objects are similar? Which are different? One important point to keep in mind, however, is the following:

*Similarity, measured by a distance function, does not necessarily imply “alikehood.”*

For example, a weight of 10 pounds is alike a height of 10 inches, but only in terms of measured distance, not in terms of substantive material. As is the case with most statistical procedures, we must be acutely aware of the distinction between numerical representations of data and the scientific nature of the material underlying the numbers. As William James once said, *we must not confuse data with the abstractions we use to analyze them*.

We discuss the concept of distance further in the following chapter on mathematical foundations, there highlighting the most common measure of distance, that of *Euclidean distance*.

## 1.21 Error Minimization

Error minimization is yet another unifying theme of most of the methods in this book. In all methods we seek to, in a sense, hit a nail straight on the head as much as possible and minimize error

in doing so. This idea of minimizing error occurs in virtually every method, either implicitly or covertly behind the scenes.

As an example, consider once more principal components analysis. Though typically performed using eigen analysis or SVD (*singular value decomposition*) and reproducing all the original variance in the variables, the most sought-after result of a PCA is that of a single component accounting for all the variance. Hence, the target can be considered a single component. The error function is the extent to which we require other components to make reasonable sense of the data, or, depending on the application, to reproduce the given image via data compression if the substantive application is that of image compression. Likewise, in structural equation modeling we seek to reproduce the covariance matrix; the extent to which we are unable to *exactly* is the extent to which we are in error. *Minimizing error when imposing some kind of functional or near-functional form onto data is a goal of many statistical and machine learning methods.*

## 1.22 Training vs. Test Error

When we consider  $f(x)$ , this is regarded as the *true* unknown function. We typically estimate  $f(x)$  using  $\hat{f}(x)$ , where  $\hat{f}(x)$  is the estimated function. Do we care whether the function is a “black box” or something we can “unpack” and get to know better? If we do, then we would like to know exactly what  $f(x)$  is; if we do not, then black-box predictability may be of primary interest without actually getting to know the functional form. Perhaps *Amazon.com* simply wishes to know what product you will most likely buy next; it may not care *why*; it simply cares to know which one.

When we *train* a model, we fit it to sample data. However, as we will discuss later in the book, *training error* (i.e., lack of fit) is usually not of primary interest. Rather, we instead wish to know how well the model fits on data it has *not* yet seen before. This is known as *test error* and is usually the final verdict on how well a model fits, at least for the given instance of fitting it to sample data. A useful analogy is that of a boxer training for a fight; we care less about how the boxer does on sparring practice opponents and instead care more about how he or she does against a new fighter. “Fight night” is the *true test* of the boxer’s skill, not the training sessions in preparation for the fight.

## 1.23 Cross-Validation and Model Selection

Cross-validation is a major “theme” of statistical learning and machine learning that differentiates itself somewhat from “classical” statistics of the past. When we come across a research paper that applies a statistical method to data and reports a measure of fit, the magnitude of the fit statistic is usually *optimistic* because by fitting the model to the sample data we may have capitalized on random or chance factors. As we have already discussed, we may have *overfit* the model. That is, the R-squared reported of say, 0.80, may be much too high of a realistic assessment of model fit. Yes, it is accurate on the *training data*, but as we have emphasized, we are much more interested in how the model performs on the *test data*.

The *test error* is the average error that results from using a statistical learning method to predict the response on a new observation – that is, a measurement that was not used in training the method. This definition and the wider concept are critical in evaluating a model. As mentioned, *training error typically underestimates the test error rate*. Estimating test error is not as easy as it sounds, however. Ideally, we would have a designated test set of observations (say, 20–30% of the original data set) that we could directly apply the statistical model to in order to observe how it performs. However, often we do not have such a data set at our disposal and hence a number of



techniques have been developed to aid in estimating test error by simply using the available training data (e.g., Akaike Information Criterion (AIC)).

For the moment, assume we do have enough data such that we can set aside some of it for empirical cross-validation. There are different approaches that have been developed for cross-validating on the new sample. The *validation set approach* designates a subset or *holdout* of observations on which to test (e.g., 20–30%). *Leave-one-out cross-validation* adopts a similar philosophy but sets aside a single observation to predict and hence can be considered a special case of the validation approach for which  $n = 1$ . The test observation is  $(x_1, y_1)$  and is not used in the fitting, so  $(y_1 - \hat{y}_1)^2$  provides an approximately unbiased estimate of the test error. It is unbiased, but highly variable since it is only a single observation. By “highly variable” is implied that if we used a different observation we might get a quite different result. So, we repeat for all observations and thus the leave-one-out estimate for the test mean squared error (MSE) is the average of all  $n$  test error estimates:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n MSE_i \quad (1.8)$$

where  $MSE_i$  is used for quantitative responses (for discrete or categorical responses, other measures of error are typically used). Leave-one-out cross-validation is less biased than the validation set approach. That is, it is less likely to overestimate test error as the validation approach typically will. Also, there is no randomness in this approach as with the validation approach. Since we use all  $n$ , the idea of random sampling or splitting into arbitrary validation sets does not exist. Yet another approach is that of *k-fold cross-validation*, a kind of compromise between the validation set and leave-one-out approaches. K-fold cross-validation produces “folds” of greater number than  $n = 1$  as in the leave-one-out case. Keep in mind that when we examine real data, we do not know the true test MSE, and so we can’t *fully* determine the accuracy of the cross-validation estimate. When performing cross-validation, sometimes we want to know the exact value of the minimum, whereas other times we just want to know which method gives the lowest minimum.

We will revisit cross-validation as we apply it to select methods throughout the book. However, we need to issue an important caveat: the process of cross-validating a model can result in a relatively elaborate exercise of simulating various parameters via *grid searches* and other exploratory visual tools as to seek out and arrive at the “best” model or tuning parameter(s) in terms of minimizing error, but these tools should also be interpreted in conjunction with substantive concerns of the area you are working in. In this way, “fine-tuning” a model is not usually as straightforward as following this or that computational cross-validation “routine” and should be considered in a wider scientific context.

Further, trying *ranges* of tuning parameters via grid searches and varying other settings is often recommended instead of relying too much on one or two parameters exclusively. For example, in methods such as *ridge* and *lasso* regression, good cross-validation often involves grid searches with varying ranges of input parameters to determine the best values to minimize error, not simply one or two values that have been “tuned” on the train data. Our select “snippet” discussions of cross-validation in this book usually simplify this process greatly for the purpose of keeping the treatment brief and space-wise manageable. However, this is in no way meant to suggest the process of cross-validation and arriving at the final model is therefore complete. In actual data analysis, *it may be just the beginning*. It would be a mistake for the reader to assume that because our discussion of cross-validation is brief or often omitted that it is therefore not necessary or important for the



given method. It most often and surely *is*; however, extensive cross-validating via computational demonstrations would take us far beyond where we want to go in an introductory text that seeks to strike an ideal balance of *theory*, *method*, and *practice*. In many cases, we provide the requisite references to packages where extensive cross-validation direction can be sought out. As mentioned earlier in the book, most software examples in the book are simply meant as “foot in the door” glimpses into the computational aspects and in most cases are meant only to selectively *demonstrate* the technique; these easy demonstrations are not meant to be exhaustive or in any way complete “cookbook” or necessarily thorough or exhaustive steps to conducting any of the analyses.

### 1.24 Monte Carlo Methods

Monte Carlo methods feature the use of repeated iterative sampling facilitated by the computer. Note that we do not *need* the computer to do such sampling but suffice to say that it would be virtually impossible to sample extensively without the convenience of a computer that can easily and very quickly automate such a process. Any time a method features repeated sampling of some kind by computer, it is, by definition, a *Monte Carlo method*. Of course, the evolution of the computer and advanced computing speed has made the idea of resampling relatively easy to implement. It should be noted, however, that despite a Monte Carlo method being advertised as executing *random sampling*, resampling by computer could never *truly* be random. That is, once we have an algorithm choosing the next “random” number, the true randomness idea is forever lost.

One common use of the technique is in estimating the sampling distribution of a statistic, either because the sampling distribution cannot be derived analytically or simply because we want to compare the Monte Carlo method to its analytical counterpart. When we estimate a quantity and its sampling distribution using such methods, it often goes by the name of *bootstrapping*. We survey the bootstrap in the following chapter. The bootstrap is a special case of the more general Monte Carlo method because it features resampling. However, note that bootstrapping is not *equivalent* to the Monte Carlo method.

Monte Carlo methods are also often used to solve complicated optimization problems via randomized algorithms and hence are often employed in statistics and machine learning. Such approaches can also quite easily be used to estimate statistical power against competing alternatives. For a good example of this, see Rizzo (2019).

### 1.25 Missing Data

Missing data should never be ignored as there is often a reason for the *missingness*. The fact that an individual did not answer a particular question on a survey is not missing data; rather, it is *extremely informative data* and may signal the most important of responses. Perhaps the individual did not want to admit to ever stealing anything? Hence, the question, “Have you ever stolen anything?” is answered by not answering! That is the data that becomes of *most interest*. *Never ignore missing data. Instead, get curious as to why it is missing.*

Missing data is sometimes discussed under the more general topic of “cleaning data” in textbooks. *Cleaning data* typically refers to not only replacing or imputing missing data but also correcting for erroneous entries and detecting other abnormalities in the data file. For these latter uses, data cleaning is fine, but for replacing missing data, as mentioned, the analyst should proceed with caution. Empirical data is *experience*; it is the “stuff” that is of paramount importance in your entire analysis; it should not have to be “cleaned” that much. If you find yourself spending lots of time “cleaning” a data file, you may want to reconsider how your data was collected.

### 1.26 Qualitative Approaches to Data Analysis

The topics of this book are from statistics and machine learning, and hence are considered *quantitative* or *numerical* in nature. Quantitative approaches dominate science. However, *qualitative* approaches can be just as useful and in some cases even more fruitful. Such approaches are often used in the social sciences as well as humanities (e.g., history, anthropology, etc.). A philosophical attempt to distinguish between what is “qualitative” vs. “quantitative” usually turns to “mush” rather quickly and there ends up being more similarity between the approaches than differences (numbers are, after all, *qualitative categories*). As discussed earlier in the book, categorization can be a dangerous thing, epistemologically. Most forms of knowledge and language end up being about *patterns* and *structures* among objects, and whether we measure them as quantitative or qualitative becomes a side issue. Quantitative approaches are usually regarded as more rigorous because numbers and mathematical operations have less flexibility than other epistemological objects or structures.

The qualitative tradition, in general, prefers to not reduce information to numbers. There is a very good reason for this, as numbers may actually mask and detract from underlying processes if not used correctly. As noted by Carrera-Fernández et al. (2012), who discuss qualitative research,

in which the researcher trusts textual data more than numerical data and analyzes this data in its textual form instead of transforming it into numbers for analysis, with the objective of understanding the meaning of human action ... not only textual data but also visual, verbal, bodily, audio and spatial data are used ... data are analyzed in their original dimension, without being previously transformed into numbers.

(p. 3)

Of course, a qualitative approach to determining the proper flap angle on an airplane at take-off is out of the question, unless of course the qualitative categories are precise numerical measurements. Numerical accuracy in this area of investigation is imperative. Note it is still qualitative, only that the qualitative nature is one of precise mathematical categories. In investigations where numbers may not represent underlying essences, for instance the study of poetry, literature, or the study of the soul, history, and so on, quantitative approaches may actually *detract* from such underlying essences. Quantitative approaches, especially in the social sciences, are largely overrated. In certain domains, they give the *appearance* of rigor, but underneath the numbers is too often a weak foundation. Rollo May’s *The Meaning of Anxiety* (2015) may tell us more about anxiety, its *essence*, than this or that statistical model applied to imperfectly measured anxiety data. The important point here is to keep an open mind and not discount qualitative approaches simply because they are not “math heavy.” *Not everything is reducible to numbers*. However, not adopting such a reductionist philosophy can make the communication of qualitative characterizations much more difficult. Perhaps in the end what is needed is a proper mixture of approaches in accounting for complex phenomena.

### 1.27 Chapter Review Exercises

1. Through an online internet search, try to distinguish between *statistics*, *machine learning*, *data science*, and *artificial intelligence*. Is making the distinction easy or hard? Why? Where does *computation* fit into these distinctions?
2. Explain what it means for an algorithm to be *stable*, both conceptually and technically.
3. Distinguish between *inductive* vs. *deductive* logic.
4. In R, try computing  $5 = 5$ , and then  $5 == 5$ . How is the output different in each case?
5. Distinguish between *supervised* and *unsupervised* learning and discuss why the distinction can quickly become artificial.
6. Distinguish between *overfitting* and *underfitting* and give an example in each case, whether of a model fit to data or a more general theory or narrative that seeks to explain empirical events.
7. In reliability theory, under what case would  $\sigma_Y^2 = \sigma_T^2$ ? Do you think such a situation is realistic in measurement and science more generally? Why or why not?
8. Why is something measurable at the ratio scale also necessarily measurable at the nominal scale? Explain.
9. Distinguish between *probably approximately correct* and the *V-C dimension*.
10. Distinguish between *parametric* vs. *nonparametric* statistical and machine learning methods. Do you encounter any difficulties when trying to differentiate between them?
11. What is the *curse of dimensionality*? Why might it be a problem in dimension-reduction methods?
12. Distinguish between *training* vs. *test* error and discuss where the concept of cross-validation “fits” in between these two types of error.
13. If you had to collapse all approaches into quantitative or qualitative, which would you choose? That is, can qualitative approaches be “subsumed” under quantitative, or are quantitative approaches best subsumed under qualitative? Explain.

## 2 Mathematical and Statistical Foundations

In this chapter we provide an overview of the general mathematical and statistical background required for understanding the rest of the book. Most readers of this book will arrive with a varied background; some may be coming to this book having taken prior mathematics, statistics, and methodology courses, while others may only have the bare minimum introductory first- or second-year statistics course as their only preparation that they took likely as a follow-up to introductory courses in linear algebra and calculus which they may or may not have understood. Sadly, in today's climate, it is very common for students to be able to pass a course in calculus, for example, without being able to describe what calculus is even *about* afterward; one can apply the power rule of differentiation to the function  $f(x) = 2x^2$  to get  $f'(x) = 4x$ , but one may not understand at all the work that has been just performed. What does  $f'(x)$  mean here? How should  $4x$  be interpreted? It has been remarked by many that far too much mathematics education is about *computation* rather than truly *understanding*. For such readers who may not have been encouraged to *understand* in prior coursework, some or much of the material surveyed in this chapter may not quite be “review” and will instead essentially be introduced for the first time. Readers with a limited background are encouraged to supplement their study of these topics with references provided throughout and suggested readings at the close of the chapter.

As mentioned in the previous chapter, the foundations of statistics and machine learning are not really in mathematics. Mathematics only serves as the *vehicle* by which such concepts are communicated. For instance, in a simple linear regression problem, the “idea” is to minimize error in prediction. That's the *concept*. To operationalize the concept requires more than words or a general idea; it requires *precise symbolic definition and representation*, that is, *mathematics*. The two are not one and the same. And while statistics and mathematics are challenging in their own right, the more significant challenge as it concerns the applied scientist is to understand *why* the given mathematics is being used in the given context. This helps to motivate the big-picture ideas behind the mathematics.

As an example, *kernels*, while all the rage in modern machine learning, are in fact quite old and not new to linear algebra and can be used in a variety of contexts well beyond machine learning. But why use them at all? What is the general “theme” underlying their use? The general big-picture idea to using them is to both facilitate computation while also providing a shorthand to approximating functions in higher dimensions. The underlying mathematics allows us to define what this looks like *symbolically*, but understanding *how*, *why*, and *when* to use kernels is the more “global” picture that is most important to grasp. “Big-picture” understanding is what is needed most and which is most often lacking from typical undergraduate *and* graduate courses, especially in quantitative courses targeted to applied scientists. Students need to be taught *why*, not just *how*. When one understands the “why” the how either reveals itself or can be dug up. Learning the *how* at the

expense of the *why* is not pedagogically meaningful, yet sadly is often encouraged in the teaching of students. Students are not *robots*; they are *creative minds* and should be educated as such.

The technical “underground” of statistics, multivariate statistics, machine learning, and data science more generally is typically found in what are often regarded as the three “pillar” areas of mathematics: *probability and statistics*, *linear algebra*, and *calculus and optimization*. Indeed, it is not out of the ordinary for a statistician or researcher to simultaneously be somewhat of a “mini expert” in linear algebra or calculus, or both, since so much of linear models and multivariate modeling has its technical basis in these areas. As an example, consider the topic of *eigenvalues* and *eigenvectors*. These objects show up throughout statistics and machine learning, yet one can also study them in their own right, far removed from any applied statistical use. *They are mathematical objects in and of themselves*. As discussed at the outset of this book, the researcher or scientist, or more generally the “user” of these methods must be continuously aware of how much of a “sub-area” to study while also keeping focused on the primary area of their investigation. For example, too much of a focus on linear algebra eventually detracts from their applied experimental science, whatever that specialization may be (e.g., biology, psychology, forestry, chemistry, etc.).

## 2.1 Mathematical Previews vs. the “Appendix” Approach

### 2.1.1 Why Previews Are Better

Too often requisite mathematical underpinnings in books like this are given either as a disconnected review of fundamentals or as an appendix. All the “good stuff” is packed away in the appendix for the reader to look up as needed. This approach is limiting, however, as it fails to connect the material to the more substantive areas of the text. Our approach is to provide an overview of these technical fundamentals but most importantly, *preview where we will see them again throughout the book*. This last point is especially important. It does little good for the reader to learn or review a bunch of seemingly “abstract” mathematics if it is not properly *motivated*. At least not in an *applied* sense. Of course, the reader may not always readily fully appreciate the preview as it will be an incomplete description of the new concept (if it is indeed new to the reader); however, it will still serve the purpose of exposing the concepts to the reader as a preview of where such mathematics are used in statistics and machine learning. These early foundations help make understanding the pillars of statistics and machine learning (probability, linear algebra, calculus) a more *continuous* process rather than a *discrete* one. In this sense, the foundations *are* the method, but one is largely unaware of this until they see the foundations “in motion” or “at work” when studying the method later in the book, then returns back to these earlier pages to better recognize the connections that were being previewed.

As an example, the concept of a *linear combination*, taught in the abstract, may come across as an arcane and pointless symbolic mathematical object, until one realizes that so much of multivariate statistics and machine learning are about these very objects. If the reader is coming to the book with virtually no prior experience, then they will not appreciate our preview of where linear combinations will be used later in the book. On the other hand, if they are coming to the book with such prior experience, they will likely immediately recognize the connection which may help “tie” together concepts for that reader.

As another example, understanding the problems associated with *linear dependence* in a regression problem will only truly make sense once we get to linear regression, even though the concept of linear dependence is introduced and previewed in the current chapter. Books like this one are written in a more or less linear fashion, but *the understanding of these subjects is not always linear*. Chapters later in the book may help make something “click” in earlier chapters, and the reader should be pedagogically open to these types of connections.

### 2.1.2 About Proofs

We avoid formal proofs in this chapter and throughout the book. However, this does not imply that basic proofs cannot be constructed based on the details we do present. Proofs are often a lot “closer” in proximity to the reader’s capacity than they may at first realize, especially if they are truly grasping the material.

As an example, it is well known that the Pearson correlation coefficient is bounded by limits of  $-1$  and  $+1$ . Why this is so we explain in the book, though the explanation would not constitute a “formal proof.” However, if desired, a proof can be constructed from pieces of the explanation, and this typically makes for a great exercise. As gifted educator Morris Kline has said numerous times in reference to teaching calculus, it is better to understand the *intuition* behind the object than to learn a rigorous proof, at least on first reading and exposure. Indeed, historically, physicists and other scientists have relied on this approach quite heavily in the invention of new concepts. The mathematical and statistically specialized communities rightfully have high and very rigorous standards for what denotes a “formal proof,” (they should be applauded for their appreciation of rigor), but it is important for the reader to recognize that they themselves are sometimes not that far off from that “ideal” formal justification in at least some cases. *Approximations are okay*, so long as one realizes they are not the be-all and end-all and may lack formality. So long as the approximation does justice to the concept, you’re fine. If the approximation leads you astray from the concept, then that’s a problem.

Where results are very important, we provide the requisite technical references to more formal proofs and justification. For example, understanding what the *central limit theorem* is and why it works is most important in an applied sense, but if one were adamant on learning “why” via formal proof and had the ambition to work through it, select references are provided that further justify the theorem. Understanding why and how a theorem is “proved” and not the details of the proof may be enough, even for the curious reader. Proofs can be complex, but the *idea* of proof is (generally) not.

What is a proof, exactly? A quote by Abbott (2015, pp. 8–9) on famed mathematician Hardy is helpful here:

As Hardy indicates, there is an artistic quality to mathematics of this type, which may or may not come easily, but that is not to say that anything especially mysterious is happening. A proof is an essay of sorts. It is a set of carefully crafted directions, which, when followed, should leave the reader absolutely convinced of the truth of the proposition in question. To achieve this, the steps in a proof must follow logically from previous steps or be justified by some other agreed-upon set of facts. In addition to being valid, these steps must also fit coherently together to form a cogent argument. Mathematics has a specialized vocabulary, to be sure, but that does not exempt a good proof from being written in grammatically correct English.

Hence, a proof is a way of justifying that a mathematical statement is correct; it is an argument of sorts, just as arguments abound in other areas of inquiry (e.g., science, history, politics, etc.). Mathematical proofs, however, are much more rigorous because they are on objects that we can operationalize as specific quantities and variables.

One final point for now before we begin. Obviously, it is impossible to cover all the mathematics used in statistics or machine learning in one single chapter. It is even not possible to cover it all in one single *book*, or series of books, or encyclopedia! You get the idea. Our primary goal here is to simply survey the most *common*, *relevant*, and *foundationally important* prerequisite mathematics and statistics for an understanding of the contents to follow in the remainder of the book.

A related goal is to provide additional insight for readers who may already have an introductory grasp of these topics and to help such a reader see the bigger picture. For example, has the reader ever appreciated how similar basic neural networks are to other statistical methods? As mentioned at the outset of this book, one of the chief goals is to emphasize such *connections* to augment the student’s understanding of these more global notions.

## 2.2 Elementary Probability and Fundamental Statistics

Probability theory is the quantification of uncertainty and is at the foundation of most statistical and machine learning ideas. As R.A. Fisher (1925, 1935), the “father of statistics” put it, we may not be able to make *certain* predictions, but we can often rigorously quantify the degree to which we are *uncertain*. This is a powerful idea, one that essentially revolutionized how scientific thought and conclusions are “packaged.” We may not know whether it will rain tomorrow, but the extent to which we can quantify our uncertainty regarding whether or not it will rain nonetheless provides us with knowledge and gives us a “handle” on the problem. We have become relatively *certain* about the *uncertainty*. That’s a big idea. Historically, the invention or discovery of probability was somewhat of a paradigm shift, not only in science, but in everyday communication.

Ideas and concepts of probability go back at least to the ancient Greeks and probably much before then, as most important ideas do. However, probability was only mathematically formalized into a system relatively recently beginning in the 1600s. Most mathematical (and other) inventions have histories where the key concepts germinate over time. Few innovators truly work alone and few important ideas are unmotivated by other equally important ideas. The formalized interpretation of probability, considered in proper context, was more of a *culmination* of ideas that germinated for centuries.

## 2.3 Interpretations of Probability

There are generally three schools of probability: *classical*, *frequentist*, and *subjective*. Classical views are, for the most part, not really part of mainstream probability. The probability of getting a “1” on a roll of a die is  $1/6$  since there are six sides to the die and all *appear* to have equal probability. That’s an example of the classical view. Historically, frequentist and subjective notions have dominated statistics and hence also more or less dominate machine learning.

For the *frequentist*, probability is a long-run frequency; that is, if you want to know the probability of something, you need to *experience* the event over repeated trials and take the fraction of times the event occurs to the total number of trials. The typical classic and easy example is that of flipping a coin. If we flip the coin 100 times and get 50 heads, we say the probability of heads is equal to  $50/100 = 0.5$ , even though theoretically this is only an *approximation* to the true probability. The 0.5 is not the true probability. For the true probability under this approach, we would require flipping the coin an *infinite* number of times. From this perspective, true probabilities are only arrived at *in the limit* as sample size grows infinitely large, that is,

$$\lim_{N \rightarrow \infty} \frac{n_H}{N} = p(H) \quad (2.1)$$

Unfortunately, “infinitely large” could never be a reality, and hence one weakness of the relative-frequency approach is that obtaining such a large number of trials is nothing more than a fanciful idea, even if it does serve as a useful working model for defining probabilities. Many scholars have critiqued the relative-frequency interpretation of probability precisely on this basis.



The *subjectivist*, also commonly known as the *Bayesian* (since Bayesians typically adopt a subjective viewpoint of probability), takes probability to be an *opinion*, a *degree of belief* and hence does not require the unrealistic and unattainable infinite number of trials idea that is mandatory for the frequentist to accept. As we will see later however when we survey Bayesian statistics, subjective probability has its problems as well.

The key point to take away from this discussion so far is that the nature of probability has historically, and is still currently, open to debate. You are understanding this point correctly; *nobody knows what a probability actually is!* All that can be offered is an *interpretation* of probability. Which interpretation is correct is not a debate we will have here, and in fact, if you studied and wrote a dissertation on the topic, you'd most likely end up in the same place you started. That is, probability is a relatively elusive concept, and the best we can do is simply adopt a *way* of thinking about it. The frequentist and subjectivist notions of probability dominate modern science.

## 2.4 Mathematical Probability

Regardless of the interpretation of probability adopted, virtually all probability theorists agree on the fundamental mathematics of probability. That is, while the nature of probability (i.e., frequentist vs. subjectivist) is and has been up for debate for literally centuries, the essential mathematics underlying fundamental probability are usually not contested, except perhaps by true rebels and nonconformists! Hence, in what follows in our brief presentation of mathematical probability, there are no true “debates.” Most are “on board” with these basic notions and definitions. Much of modern probability is based on the *theory of sets*, which we do not review here, except as it applies directly to the concepts we do present.

The probability of the event  $A$  is denoted  $P(A)$ , and is a number between 0 and 1:

$$0 \leq P(A) \leq 1 \quad (2.2)$$

The *complement* of  $A$  denoted  $\bar{A}$  is the probability that  $A$  does not occur. Since the probability of the entire sample space, by definition (and convenience) must equal 1, this implies that

$$P(\bar{A}) = 1 - P(A) \quad (2.3)$$

The *joint probability*, that is, of  $A$  and  $B$  occurring at the same time is denoted

$$P(A \cap B) \quad (2.4)$$

If  $A$  and  $B$  cannot occur simultaneously, then

$$P(A \cap B) = 0 \quad (2.5)$$

and we say the two events  $A$  and  $B$  are *disjoint* or *mutually exclusive*. Examples of mutually exclusive events include the events “head” and “tail” on the flipping of a single coin since one cannot obtain a head and a tail on the same flip.

To know if two events are mutually exclusive, one must know about the *context* or *physical model* of the problem. For example, the events “heads” and “tails” are not *intrinsically* and *organically* mutually exclusive events; they are only so under the model of a *single flip*. For successive flips, they are not typically mutually exclusive. The only absolutely necessary criterion for mutually exclusive events is as stated in (2.5), that of  $P(A \cap B) = 0$ , and hence so long as this condition

is satisfied, even in the abstract, the two events are mutually exclusive. Knowledge of temporal conditions of the physical model imposed serves to help us know whether the condition is met.

Why is this point so important? Why are we going on about it? The reason is that physical models (which include *temporal sequences* as a special case) are special cases or circumstances of a more general “real-world” context. Abstract mathematics (or philosophy, for that matter) does not require such a basis in reality. To emphasize the point, if the aliens come one day and have no conception of time, the concept of mutual exclusiveness would still be “communicable” by simply drawing the idea in Figure 2.1 with Venn diagrams on a sheet of paper.

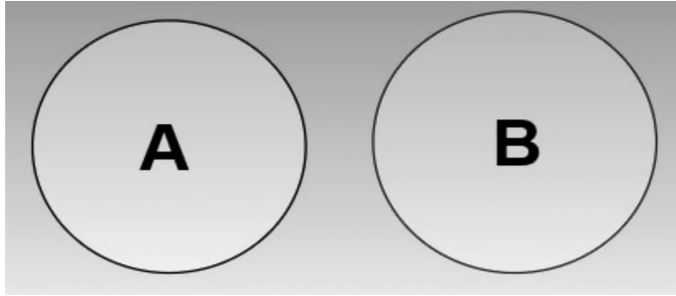


Figure 2.1 Depiction of mutually exclusive events, non-overlapping circles.

The depiction in Figure 2.1 is what mutual exclusiveness *is*, at least from a geometrical point of view. Knowing whether mutual exclusiveness applies to the objects under study must come from the *context*. If there is only one flip of the coin, head and tail are mutually exclusive events. If there are two flips, they are not. *Context matters*. Everything happens in a given context; this includes scientific, mathematical, statistical, and machine learning matters. This “theme” will return time and time again throughout this book and it is worth emphasizing and repeating here:

Context Matters!

#### 2.4.1 Unions and Intersections of Events

The probability of  $A$  or  $B$  occurring on a given trial is denoted by their union,

$$P(A \cup B) \tag{2.6}$$

and is equal to the sum of probabilities of  $A$  and  $B$  minus their intersection,

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{2.7}$$

where  $P(A \cap B)$  defines the probability of the intersection.

It is easy to see that for mutually exclusive events, where  $P(A \cap B) = 0$ , the union of  $A$  and  $B$  is equal to

$$\begin{aligned}
P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\
&= P(A) + P(B) - 0 \\
&= P(A) + P(B)
\end{aligned}$$

This is sometimes referred to as the *special rule for the addition of probabilities*, that is, where the intersection is equal to 0 and we can simply compute the union by adding probabilities. To be on the safe side, it is good advice to always plan on subtracting an intersection from the addition to ensure for yourself that there is no overlap. It is obvious and easily provable that the intersection of a number of sets must be a subset of their union as the overlap, no matter how large, cannot be larger than the sum of all sets. The reader may recall that a *subset* can consist of the entire set, whereas a *proper subset* must be smaller than the full set.

## 2.5 Conditional Probability

Most phenomena in science are not about probabilities; they are instead about *conditional probabilities*. We rarely care about the probability of an event occurring; we care more about the probability of an event occurring, *given something else has already occurred*. This “conditional” statement translates to a reduction in the original sample space from what it originally was for the unconditional probability estimate. The probability of cancer is interesting, but the probability of cancer *given that you do or do not smoke two packs of cigarettes per day* is much more interesting. Virtually every problem and finding in science, no matter how simple or complex the model or lack thereof, can be translated into a statement of conditional probabilities.

Conditional probabilities add the aforementioned “context” to the problem since unconditional probabilities, while often accurate in and of themselves, can be grossly misleading. In passing, this is precisely why some detest statistics, because depending on what other factors or conditions are imposed on a problem, one can draw one of many conclusions. For instance, while the probability of perishing from COVID-19 may be 0.05, the probability of one dying given that one is an otherwise healthy 30-year-old is likely significantly less. Unconditional probabilities are, in a sense, “averages” across objects and conditions, and as we will discuss later, averages can be grossly misleading. Conditional probabilities are specialized *focused slices* within that spectrum. They help us narrow in on the probability estimate of most interest and provide that necessary context to the problem. The difficulty is that in many or most problems, there is no theoretical limit to how much context can be included which makes arriving at the “true” conditional probability in many situations very difficult or even impossible. At worst, a lack of context will simply make the probability estimate unmeaningful and very misleading. The probability of a human being currently eating baby formula is very low. It’s a silly probability estimate because we have not conditioned it on age and thus have not provided proper context to the problem. The probability estimate is accurate. The lack of context to the problem is foolish and makes the unconditional probability estimate *uninteresting*.

The concept of so-called *risk factors* also evolves from the general principle of conditional probabilities. For instance, when a practitioner asks about the past suicidality in your family as a way of estimating your own probability of suicide, she is attempting to narrow the sample space such as to ascribe past suicidality as a risk factor in estimating your own risk of suicide. The concept of risk factors plays a significant role in many branches of epidemiology and science in general. When you hear the word “risk factor,” think “conditional probability.”

Mathematically, the conditional probability of  $A$  given  $B$  is defined as

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (2.8)$$

while the conditional probability of  $B$  given  $A$  is given by

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \quad (2.9)$$

depending on which variable we are conditioning the other on. Using conditional probabilities and simple algebra, we can now define the aforementioned intersection between  $A$  and  $B$  more formally:

$$\begin{aligned} P(B|A) &= \frac{P(A \cap B)}{P(A)} \\ P(A \cap B) &= P(A)P(B|A) \\ &= P(B)P(A|B) \end{aligned}$$

That is, the probability of  $A$  and  $B$  occurring simultaneously (i.e.,  $P(A \cap B)$ ) is equal to the product of an unconditional probability – either  $P(A)$  or  $P(B)$  – with a conditional probability where it is also true that

$$P(A)P(B|A) = P(B)P(A|B)$$

### 2.5.1 Unconditional vs. Conditional Statistical Models

As emphasized in our discussion of unconditional vs. conditional probabilities, much of statistical modeling in the end is about conditional probabilities. In most cases, however, we do not evaluate such probabilities in a vacuum. Rather, we evaluate them relative to a particular *statistical model*. Hence, when we write the unconditional probability of  $P(A)$ , it will now be understood that we are assuming a statistical model, whether that statistical model is specified explicitly or not.

Referring to our previous example, the unconditional probability of cancer is, by itself, though useful, rather uninteresting, and would denote a much too simplistic statistical model similar to the idea of predicting overall suicide rates in a regression context based on no predictors would also be much too simple of a model. The *intercept-only model* is limiting. To enrichen the model, we should want to include *covariates* and variables to provide a meaningful *context* to the evaluation of the probability. This is the very essence of much of statistical modeling, to *provide a context for evaluating and estimating particular probabilities of most interest*. Many statistical models are, in the end, nothing more than generalizations of ideas of conditional probabilities. Of course, that's a simplification, but it's a good start to understanding statistical modeling.

Going from conditional probabilities to statistical models, we need to generalize our notation. Instead of speaking of the unconditional probability  $P(A)$ , we need to speak instead of the unconditional *model*, not simply *probability*. The conditional model is given by the following,

$$p(y|\theta) \quad (2.10)$$

where the response is now  $y$  and the parameter vector is now  $\theta$ . The model is unconditional because there are no covariates. We now add a covariate  $x$  as follows:

$$p(y|x, \theta) \quad (2.11)$$

which yields a conditional model where now  $y$  is “conditioned” on  $x$ , for which the parameters in  $\theta$  will be estimated. Under the circumstance where  $y$  and  $x$  are independent, then  $p(y|\theta) = p(y|x, \theta)$ . Contrarily, when  $y$  and  $x$  are dependent, then it should be obvious that  $p(y|\theta) \neq p(y|x, \theta)$ .

## 2.6 Probabilistic Independence

From the definition of conditional probability, we can quite easily obtain the concept of statistical or probabilistic independence. If  $P(A) = P(A|B)$ , then events  $A$  and  $B$  are said to be *independent*, which implies that knowing  $B$  has occurred tells us nothing about the probability of  $A$  occurring. Note that the qualifier “has occurred” implies a temporal dimension, but as was the case for mutual exclusiveness, we need not assume a temporal context at all. All we really require is a reduction in the sample space so that the event of interest is referenced to that *reduced space* instead of to the original wider space.

We can easily see from the rule for joint events that if  $A$  and  $B$  are independent, then

$$\begin{aligned} P(A \cap B) &= P(A)P(B|A) \\ &= P(A)P(B) \end{aligned} \quad (2.12)$$

since if  $A$  and  $B$  are independent, then  $P(B|A) = P(B)$ . That is, if we can assume events  $A$  and  $B$  are independent, then we can simply multiply their unconditional probabilities in determining  $P(A \cap B)$ . This is known as the *multiplication rule for independent events*. If we cannot assume they are independent, then we cannot simplify the rule and must always invoke the conditional probability  $P(B|A)$  rather than simply  $P(B)$  in the second term. That is, the more complete rule for computing the probability of the intersection is the first above, that of

$$P(A \cap B) = P(A)P(B|A)$$

The multiplication of unconditional probabilities is simply a special case of the above more general rule. But how do you know if events  $A$  and  $B$  are independent or not? We discuss this issue next.

### 2.6.1 Everything Is about Independence vs. Dependence!

Whenever new concepts are introduced in statistics, they are often so fundamental that it is tempting to conclude that the whole discipline revolves around them. We just experienced this idea in our discussion of conditional probabilities, concluding that everything is about conditional probabilities. Even more fundamental is the idea of *independence versus dependence*. The unifying principle of these concepts cannot be overstated. It is what probability and thus virtually all statistics and science are about. On the one hand, statistically, probabilistic independence is at the heart of many probability calculations and often simplifies matters a great deal. On the other hand, in terms of real scientific empirical objects, establishing independence can be extremely difficult (and many times impossible) to demonstrate, and instead is often simply an *assumption* that

must be (conveniently, and even unrealistically in many cases) made, often on reasonable grounds given the experts concluding the independence have knowledge of the system(s) they are working with. Determining empirically that two events are independent can often be next to impossible. Statistical modeling is about making *assumptions*, often lots of them.

As an example of what we are talking about, consider an airplane system. Probabilistic independence is a sought-after goal when conceptualizing and manufacturing aircraft components and running failure analysis scenarios. Engineers purposely design airplane systems to be as independent as possible so that *a failure in one system does not change the probability of a failure in another system*. Why is this so important? Well, consider the possibilities when flying on an airplane; it would be a terrible thing if the probability of a failure of one engine increased the probability of a failure of a second engine, or if the probability of the landing gear retracting was not independent of the probability of the hydraulic system working. In these cases, it would take but one failure to influence other failures, and hence a single failure could, in theory, take down the airplane through a chain reaction of events. Hence, as much as possible, system events on airplanes are designed to be probabilistically independent. The probability of one engine failing should, ideally, not have an effect on the probability that the plane experiences a decompression, for instance.

But how do these experts “know” the systems are independent? They don’t! Well, not for *certain*, anyway. However, rational argument, a *plausible argument*, knowledge of the systems under consideration, increasing empirical evidence, and making assumptions that align with such knowledge typically lead such professionals to conclude that operating the flaps on the wings, for instance, is independent of the ability of the fuel tank to hold fuel, just as a failure of the landing gear to retract is independent of the plane’s engines to generate power. Of course, these seem like “obvious” assumptions to make, but the point is that they are often nonetheless *assumptions* (assumptions often supported by increasing amounts of evidence, in this case). The goal of science is to reject “obvious” assumptions with newly established empirical evidence that potentially counter or challenge existing assumptions.

For instance, it was once assumed that a simple cigarette could never cause any harm, just as it was once assumed that chemicals used in paint (e.g., benzene) could never pose a threat to our health and the health of our pets. Social media was never thought to be associated with depression and anxiety in our children. Science is about cutting through “established” beliefs with evidence that suggests such “establishment” should be rejected. Newtonian mechanics was once assumed correct until Einstein’s “unlikely” theory led us to believe differently. The point is that all scientific “fact” is tentative to new evidence and revision, and the idea of probabilistic independence is no different. We can assume events to be independent to get on with business (and fly the plane, for instance), but we must always allow the possibility that they are not, subject to new evidence. It was once assumed that a thin layer of icing on wings could never influence the capacity of the plane to stay airborne. We now know differently, largely due to experiencing (i.e., data) planes fall from the sky because of a thin layer of such ice. Such has encouraged us to “update” our belief about the impact of ice on wings, which is why that 20-minute delay we are so impatient about as travelers (while de-icing equipment clears our wings) could be saving our lives. The 20-minute delay is a “good thing” and has been informed by prior deaths and tragedies, that is, *experience*.

## 2.7 Marginal vs. Conditional Distributions

As we have seen, the concept of conditional probabilities is vitally important to both statistics and science. However, we have also discussed that conditional probabilities are often applied to statistical *models*, not individual probabilities. With these two elements, we can generalize concepts

of conditional probabilities to that of *joint* and *conditional distributions*. Further, for a continuous distribution, the marginal *probability density function* (PDF) over all possible values of  $y$  can be obtained from the joint PDF by integrating out all other  $p$  variables that may be present and summing (via integration) over all possible values of  $y$ ,

$$f(x) = \int_{-\infty}^{+\infty} f(x, y) dy \quad (2.13)$$

The PDF integrates to 1, that is,

$$f(x) = \int_{-\infty}^{+\infty} f(x, y) dy = 1 \quad (2.14)$$

In the discrete case, we can simply visualize summing up or down a  $2 \times 2$  table across *categories* of the second variable. As was the simpler case for the rule of independence for individual probabilities, random variables are *independent* if the joint PDF is equal to the product of marginal PDFs. If the joint PDF is not equal to the product of marginal PDFs, then the two random variables are said to be *dependent*.

## 2.8 Independence Implies Covariance of Zero, but Covariance of Zero Does Not (Necessarily) Imply Independence

As we will discuss in more detail later, the covariance between variables is a measure of the extent to which the variance of one variable is *linearly* associated with the variance of another. The covariance is given by

$$Cov_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n} \quad (2.15)$$

For any two variables that are statistically independent, this necessarily implies a covariance equal to 0. However, the converse is not necessarily true. That is, a covariance equal to 0 does not necessarily imply that two events are independent. They may turn out to be independent, but the covariance equal to 0 is not enough to imply it. This is because the relationship between the two variables may simply be one that is *nonlinear*. *The covariance between two variables is a measure of linear association or relationship only*. Hence, two variables may be strongly dependent or related to one another, yet still have a covariance (and hence, also *correlation*, though as we will see later the two are not equivalent) equal to 0. This is one reason why interpreting a measure of covariance or correlation should never be done without also inspecting a scatterplot of the association. *Always plot your data!* This is worth repeating:

**Always plot your data!**

Numerical summaries are never enough. As an extreme instance and very insightful example of this effect, we refer to *Anscombe's Graphs in Statistical Analysis* (Anscombe, 1973) in Figure 2.2 which consists of three different data sets that yield identical summary statistics.



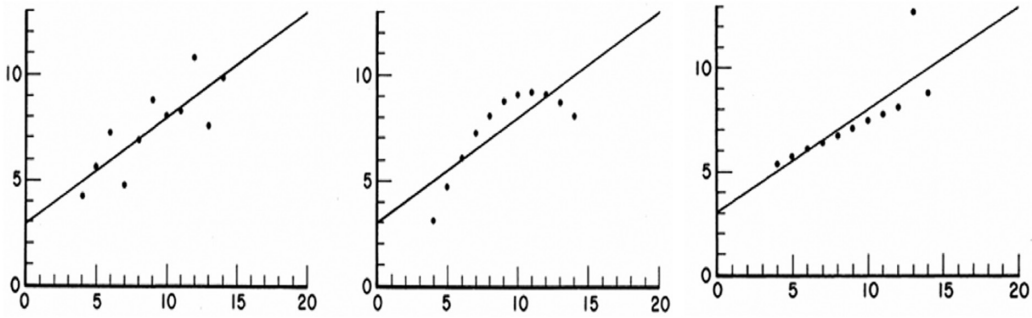


Figure 2.2 From Anscombe, F. J. (1973). Graphs in Statistical Analysis. *The American Statistician*. Copyright © 1973 American Statistical Association, reprinted by permission of Informa UK Limited, trading Taylor & Francis Group, <https://www.tandfonline.com> on behalf of American Statistical Association.

Remarkably, all four plots share the same means on  $x$  and  $y$  ( $\bar{x} = 9.0$ ,  $\bar{y} = 7.5$ ), regression of  $y$  on  $x$  (slope of 0.5), regression equation  $y = 3 + 0.5x$ , sum of squares  $x - x = 110.0$ , regression sum of squares (27.50), residual sum of squares (13.75), estimated standard error (0.118), and  $R^2 = 0.667$ . If such a demonstration does not convince the reader to always plot their data, nothing will!

Other examples of this effect abound (see Murphy, 2022, p. 42 for examples). In short, *visualization is a must for good data analysis and exploration, not an option*. Let's put this another way; if you are not visualizing your data in at least *some* capacity, you are usually conducting a *poor data analysis*. There are special cases where one may be able to circumvent visualization, but the point is that visualization will tell you things about your data that numerical summaries may never. Numerical summaries alone will never provide the widest or most complete picture or understanding of the data, and as we have said already ad nauseam by now, and will continue to repeat throughout the book, all variables exist in a wider *context* of other variables. Visualization helps provide a description of that context, which is why so much effort is put toward developing new visualization techniques in statistics. As we will see later in relation to manifold learning, multivariate visualization can be especially challenging and difficult, and as the dimensionality of the data increases, can be virtually impossible.

We will revisit covariance and correlation in more detail shortly.

## 2.9 Sensitivity and Specificity: More Conditional Probabilities

Another area where conditional probabilities are featured and used is when determining the sensitivity and specificity of diagnostic tests. A diagnostic test is a *sign*. It is a piece of *data*. It is a moment of *experience*. It is not a deterministic indication that the disease or condition is present. A classic and modern example of this is diagnosing COVID-19. A diagnostic test may indicate the presence of the virus, but there is also associated with that a probability that the disease is not present. A test that detects the disease when the disease is present is called the *sensitivity* of the test. A test that fails to detect a disease that is not present is called the *specificity* of the test. Good tests have strong sensitivity and specificity though the two will seldom, if ever, be perfect or equal to one another.

Of course, we desire any test to have maximum sensitivity and maximum specificity. That is the ideal we shoot for. Where things go wrong with diagnostic tests is when a test indicates the virus or condition is present when the virus or condition is not present. This is known as a *false positive*, or more commonly in the statistical literature as a *Type I error* in hypothesis testing. On the other hand, a test that fails to detect a virus when the virus is present has committed a *Type II error*.

Which is the worst error? Which error has the greater consequence? Such, of course, depends, again, on the *context of the problem*. Context matters! Again! Which is most important depends on which type of error the scientist wishes to minimize. Surely, we do not want to misdiagnose the virus. Yet at the same time, diagnosing the virus when it is not present can lead to many problems, such as emotional distress, financial costs, the inconvenience of further testing, etc. Indeed, this has been one of the controversies surrounding habitual mammogram testing, for example; some argue that due to the error rates involved, yearly mammograms may be doing more harm than good. The risks of diagnosing a disease when none exists are real and can have great consequences. At the same time, missing a condition when the condition is present can likewise have serious consequences. Which error is more consequential largely depends on the situation, context, and the inclination of the scientist or subject.

## 2.10 Bayes' Theorem and Conditional Probabilities

There is perhaps no better or more important of an example of conditional probabilities as with Bayes' theorem. Recall that we mentioned at the outset of this chapter that the two primary camps in statistics are those of the frequentist and subjective perspectives. Maximum likelihood estimation, the most common method for estimating parameters, generally stems from and is governed by a frequentist approach in which the probability of the data given a parameter is evaluated, that of  $p(y|x)$ , where here  $y$  is the data and  $x$  is the parameter (in place of the notation  $\theta$  for our discussion here). The conditional probability  $p(y|x)$  is referred to in frequentist probability as the *likelihood*.

In Bayesian reasoning, the probability of the parameter given the data is instead evaluated, that of  $p(x|y)$ . Bayes' theorem inverts conditional probabilities from  $p(y|x)$  to  $p(x|y)$ :

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \quad (2.16)$$

where  $p(y|x)$ , as before, is called the *likelihood*,  $p(x)$  is the *prior probability*, and  $p(x|y)$  is the *posterior probability*. The denominator,  $p(y)$  is the *total probability* of the data, or evidence more generally.

An even more meaningful way to interpret Baye's theorem is by replacing  $x$  with  $T$  (theory) and  $y$  with  $D$  (data):

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

$$p(T|D) = \frac{p(D|T)p(T)}{p(D)}$$

That is,  $p(T|D)$  is the probability of the theory given the evidence or data. The difficulty with Baye's theorem is that one must enter a *prior probability*, which can be quite subjective in nature. Note that no matter the strength of the data, if the prior probability  $p(T)$  is equal to 0, then the posterior probability will also equal 0, since,

$$p(T|D) = \frac{p(D|T)(0)}{p(D)}$$

$$p(T|D) = 0$$

The big idea in Bayesian analysis is that contrary to frequentist approaches where parameters are regarded as fixed constants, Bayesian analysis treats parameters as *random variables*.

### 2.10.1 Bayes Factor

In the case where we have two competing models,  $M_0$  and  $M_1$ , we can form the *Bayes factor*, which is a ratio of likelihoods of the data under these two hypothetical conditions:

$$B = \frac{p(D | M_1)}{p(D | M_0)} \quad (2.17)$$

where  $p(D | M_1)$  is the probability of the data under the condition of the alternative hypothesis,  $M_1$ , and  $p(D | M_0)$  is the probability of the data under the condition of the null hypothesis,  $M_0$ . If the two likelihoods are the same, then we would, obviously, expect  $B$  to equal 1, whereas if  $M_1$  is preferred over  $M_0$  then we would expect  $B > 1$ . The Bayes factor provides a competing alternative to classical frequentist statistics that focuses on the  $p$ -value as a means of making a decision between the two states. According to Jeffreys, a Bayes factor greater than 10 indicates strong evidence for  $M_1$  while a Bayes factor greater than 100 indicates exceptionally strong evidence for  $M_1$  (Murphy, 2022). However, as always, the size of the ratio should be evaluated, at least in part, relative to the context of the investigation. A Bayes factor of 2 in some research settings may be considered relatively large, whereas a Bayes factor of 10 in another research setting may be considered trivially small. Have we mentioned before that context matters?

### 2.10.2 Bayesian Model Selection

Suppose we run a host of competing models and wish to select the “best” one in some sense. The qualifier “in some sense” is necessary here, as selecting a “best” model is in no way a straightforward task and will usually depend on much more than a single criterion for selection.

By a Bayesian approach, that model should be the one that maximizes  $p(m | D)$ , where  $m$  is now the model and  $D$  is, as before, the data:

$$p(m | D) = \frac{p(D | m)p(m)}{\sum_{i=1} p(D | m_i)p(m_i)} \quad (2.18)$$

We recognize in the above the quantity  $p(D | m)$ , which, when unpacked, is equal to

$$p(D | m) = \int p(D | \theta, m) p(\theta | m) d\theta$$

and is called the *marginal likelihood* (Murphy, 2022). If  $p(D | m)$  holds strong under different settings of the parameter  $\theta$ , then  $m$  is likely to be considered a reasonable model. The key to the marginal likelihood is the summing (via integration) across a variety of parameter values ( $\theta$ ). This is why it is called “marginal,” as a result of the summation over all values of  $\theta$ . If it were for a particular value of  $\theta = \theta_i$ , then it would be “conditional” instead of marginal. Under the condition of two competing models  $m_1$  and  $m_2$ , should  $p(D | \hat{\theta}_1, m_1)$  and  $p(D | \hat{\theta}_2, m_2)$  both be large, Occam’s razor requires us to select the model that is simplest. Marginal likelihoods then help us select a suitably complex model, which has analogies to cross-validation and other model selection guidelines in a frequentist context (see Murphy, 2022, pp. 179–180 for details). However,

marginal likelihoods can be difficult to compute in practice because they require integrating over the entire parameter space (Murphy, 2022). The *Bayesian Information Criterion* (BIC) can be considered a kind of approximation to the log marginal likelihood, which, as we will see later in the book, is used in many contexts, including regression and cluster density estimation, among others.

### 2.10.3 *Bayes' Theorem as Rational Belief or Theorizing*

Bayes' theorem is not a trivial idea. Far from it. By itself, Bayes' theorem tells us how, as *rational agents*, we should update our belief about  $T$  in light of new  $D$ , where  $T$  is nothing more than a belief or theory and  $D$  is nothing more than experience in its fullest generality. Bayes' theorem is not a statistical idea; first and foremost it is a *philosophical* one. Bayesian inference essentially tells us how a rational agent *should* update their belief in light of new experience. "Should" equates to being *rational*. It is not equivalent to how human beings *actually* update their beliefs. Indeed, much psychological research has been conducted on human subjects to demonstrate that human beings are far from being Bayesian.

As one might imagine, this idea has made Bayes central to not only statistics, but also other areas where the study of rational behavior is of primary interest. The application of Bayes' theorem, whether formal or informal, is a way of being "intelligent" about updating one's belief. *Game theory*, *psychology of beliefs*, *economic theory*, etc., are all areas that, in the end, study human behavior. Sub-specialties of these domains have interests in learning whether human beings update their beliefs in correspondence with what Bayesian theory would prescribe. The *Bayes estimator* tells us which action to take, if acting *rationally*, to minimize the risk of making a wrong decision. Maximizing expectation, however, is not always equivalent to maximizing *utility*, and hence Bayesian decision theory will often have us maximize the latter instead of the former, except if the problem is relatively simple and both expectation and utility coincide. Utility, simply put, is a measure of how beneficial this or that action is to the agent, not simply evaluated based on probability or mathematical expectation. Not buying that overpriced post-pandemic McDonald's hamburger may maximize budgeting constraints, but buying it because it tastes *so good* maximizes utility. A fiscally poor and irrational choice, but has much utility for the buyer and thus becomes rational from that perspective. Utility theory is a fascinating area of investigation. For an excellent discussion, consult Winkler (2003).

While Bayes' theorem is generally useful, it should be remarked that it could introduce bias and discrimination into the process of updating belief. For example, the high probability of a black person committing a crime may be justified by reference to Bayes' theorem because for a given crime, past crimes may have featured a disproportionate number of blacks, causing the prior probability to skyrocket. Bayes' theorem is rational updating of belief, but it may be considered discrimination to use it in this particular *context*. Likewise, the probability of a mass in a dog being cancerous may be high upon application of Bayes' theorem, given the age and breed of the dog. While Bayes' theorem is doing its work, it can also be terrifying that a dog is put down because of their breed and without the result of a biopsy. The prior probability makes a cancer diagnosis more likely, perhaps even highly probable, but if it's a decision of *life vs. death* for the dog, putting the dog down on the "gamble" that it is cancer is an unwise action. *The cost of making an error in this regard may be too high, even if the posterior probability is exceedingly high. Bayes theorem can only estimate the probability of the theory; it cannot, by itself, tell you how to act, nor is a posterior probability ever equal to unity in most real-world applications.* Further, it must always be remembered that we do not know anything about the current event; we only know about *past events*, and the prior probability in Bayes' theorem, even if estimated correctly, is not about the current event under consideration. *Calculate Bayes' theorem with reckless abandon, but use, and most importantly*

**interpret Bayes' theorem with caution.** As a result of the prior probability potentially having an overwhelming influence on the posterior, work has been done on what are called *noninformative priors* (Bishop, 2006), which, as the name suggests, attempts to reduce the influence of the prior.

We will revisit Bayes' theorem often in this book, such as when determining the optimal decision boundary when discussing *discriminant analysis*, or when considering *Naïve Bayes* as a nonparametric alternative to classification. The idea of including prior information in a model is not new and can be witnessed throughout statistics and science, as well as more modernly with machine learning. For instance, a new paradigm in machine learning called *physics-informed machine learning* has the goal of informing machine learning models using physically relevant prior information (Pateras et al., 2023). That is, the model or algorithm is “informed” by prior knowledge of the object or system that is being modeled.

## 2.11 Law of Large Numbers

The law of large numbers is a fundamental law in statistics that in words essentially says *the larger the sample size, the closer one gets to the true probability of an event*. As a simple example, getting 9 heads out of 10 trials yields an estimate of the probability of heads equal to  $9/10 = 0.90$ . The estimate of the probability of heads is not that great in this instance because it is based on only a few observations, that is, a relatively small sample. As sample size  $n$  increases, the approximation gets better, until in the limit as  $n \rightarrow \infty$ , the true probability is known. The limit, in this case, infinity  $\infty$ , is never achieved. The true probability, at least in this frequentist perspective, will never be known. All we really have are estimates. The bigger the sample size, in general, the better the estimate.

The law can be formalized as follows for the case of the sample mean  $\bar{y}$  when used as an estimator of the population mean,  $\mu$ : for any  $\varepsilon > 0$ , the probability of any deviation of the kind  $y - \mu$  goes to 0 as  $n$  increases, eventually without bound, that is,  $n \rightarrow \infty$ . Alternatively, we could express the same thing by saying for any  $|\varepsilon| > 0$ , the probability that  $y - \mu$  is between  $-\varepsilon$  and  $\varepsilon$  goes to 0 as  $n \rightarrow \infty$ . Incidentally, it is worth noting that infinity,  $\infty$ , is not regarded as a number in mathematics, but is simply a *concept*. The symbol for positive infinity in R is `Inf`.

The law stems from a relative-frequency interpretation of probability and more pragmatically means that *if you want to know the true probability of an event, the more data you collect, in general, the better your estimate of that probability*. Note that this does not necessarily imply that collecting more data is always advisable. A million data points can do no worse than a thousand, unless of course the cost of collecting such extra observations is prohibitive and one can conclude about just as much knowledge from the initial sample of a thousand than the million. While the law of large numbers wants us to collect more data, it is a theoretical argument only. It may not be practical in reality. That thing we call the “real world” and is the unpleasant distraction from the theoretical mathematician often interferes with our capacity to collect more data and if the identical conclusion can be drawn with lesser data points, then collecting more data may simply be a waste of time and resources. Limiting theorems such as the above are attractive, but we must not get lost in their beauty at the expense of practical, real-world considerations.

An implication of the above is that statistical *power curves* tend to asymptote (horizontally) as sample size increases, and hence the cost of collecting more data may simply not be worth it. In the age of big data and the facility by which large data sets can be collected, these truths still hold, but since collecting data can be done rather cheaply these days in many contexts, such costs are unlikely to deter businesses such as *Amazon* from collecting terabytes of data and hence ignore power curves altogether. When “Big Data” is available, power curves are left in the wind. In such contexts, data is far too easy to come by to be concerned with power curves.

However, in other settings, such as smaller experimental ones where collecting data may be very difficult and vastly expensive, establishing just how much information is needed to draw

conclusions is a very worthwhile activity. For example, if your units of observation consist of children with a particular rare disorder, each single data point may be extremely difficult to collect and the difference between 20 and 25 observations may be a huge undertaking. Collecting data may be prohibitively expensive. The development of small-sample statistics in the early 1900s made it possible to study populations without having to sample huge numbers. This was the gift of inferential statistics. We seem to be coming full circle in the current climate where collecting bigger data sets is the norm in some areas (definitely not all), mostly due to the facility by which such data can be collected (i.e., electronically) and easily stored. Early inferential statistical procedures were originally designed to circumvent this activity and make the most out of small data. Will the age of “small data” ever return? Let’s imagine what that might look like in the following fictional academic news report headline:

An amazing new quantitative technology is being popularized for which one only requires very small samples to make generalizations to much wider populations. We call this new age of data analysis, SDA, or “Small Data Analysis.”

Increasing sample size can also have an influence on Bayesian posteriors. For example, in choosing a model, for small sample size, a posterior may indicate one model, whereas as sample size grows, a competing model may have a higher posterior due to the increase in information. *Weak* vs. *strong* laws of large numbers can also be distinguished though not discussed here. For details, see Shao (2023) for a clear but relatively theoretical discussion. Casella and Berger (2002) also feature a discussion (p. 235).

### 2.11.1 *Law of Large Numbers and the Idea of Committee Machines*

One indirect application of the law of large numbers that we will revisit later in the book is contained in the idea that *average prediction functions converge to expected prediction functions*. For instance, ten prediction functions, when averaged, get us closer to the true expected (i.e., in terms of long-run probability) prediction function than any of the single prediction functions. That is, *when we combine many weak learners, we often achieve better predictions than the capacity of any of the weak learners that make up the combination*.

An analogy to sports would be that a team of weak players produces a better outcome than any of the individual teammates. Indirectly, the law of averages is present in this statement, since it implies that the greater the  $n$ , where “ $n$ ” in this case is the number of prediction functions (players), the better we will converge to the true function. This idea of combining so-called “weak learners” to produce a stronger learner generally falls under the general category of *committee machines*.

## 2.12 *Random Variables and Probability Density Functions*

A *random variable* is a mathematical variable that is associated with a probability distribution. More precisely, a random variable is a function from a sample space into the real numbers (Casella & Berger, 2002) such that for any “slice” or interval, that is, for any subset of the sample space is called an *event*. The function must as well be *measurable* (Mohri et al., 2018) though measurability of functions is well beyond the scope of the current text. We simply make mention of the word “measurability” here to denote the fact that not all functions are measurable, and hence saying that random variables are a function onto the real numbers is incomplete. Deeper and more sophisticated definitions of a random variable will first distinguish between *measurable* versus *nonmeasurable* functions (e.g., see Shao, 2023).

As an example of a random variable, consider the variable  $x_i$  having values 0, 1, 2, 3. The variable  $x_i$ , as it currently stands, is often called a *mathematical variable*. The descriptor “mathematical” is usually implicit, but such a variable may often be called such in a statistics text simply to distinguish it from a random variable. As the reader will recall, it is a variable because  $x_i$  can take on different values in the set. If  $x_i$  could only take on a single value (e.g., 0 or any other value), it would not be a variable but instead would be a *constant*. It becomes a *random variable* when we associate probabilities with those values in the set such that they sum to unity. For instance, the aforementioned  $x_i$  is now a random variable in the table below, where each value of  $x_i$  is now associated with a probability of 0.25:

| $x_i$        | $p(x_i)$ |
|--------------|----------|
| 0            | 0.25     |
| 1            | 0.25     |
| 2            | 0.25     |
| 3            | 0.25     |
| $\sum = 1.0$ |          |

Note that there is no requirement that probabilities be equal across values of  $x_i$  for  $x_i$  to be a random variable. The only requirement is that the sum of probabilities is equal to 1.0. That is, we could have just as easily distributed the probabilities as 0.20, 0.40, 0.30, and 0.10 to illustrate the concept.

### 2.13 Convergence of Random Variables

Where a random variable “is going” is often of theoretical interest. That is, we often would like to know what value a random variable *converges to* in the limit. As sample size increases without bound to infinity, that is,  $n \rightarrow \infty$ , we would like to know properties of this convergence.

We say a sequence of random variables *converges in probability* to a random variable  $x$  if:

$$\lim_{n \rightarrow \infty} p[|x_n - x| \geq \varepsilon] = 0 \quad (2.19)$$

In passing, note we have used the word *infinity* above, and do so as well in other places in the book. And while it seems convenient to use it and the spirit of what is meant by statements such as  $n \rightarrow \infty$  is quite easily understood, the concept of infinity itself is quite rich historically and comes with it many philosophical difficulties and challenges, many of which are still ongoing today. Our inclusion of it in no way is meant to trivialize the depth of the object. Indeed, a dissertation with the title *The History and Theory of Infinity in Philosophy and Mathematics* would be no easy undertaking! Some hold that concepts of infinity, coupled with abstraction and proof, open the gateway to advanced mathematics (Stillwell, 2016). Make no mistake as well, by our mention of *philosophy of mathematics*, we are in no way meaning to communicate anything less than extremely rigorous. An in-depth study of infinity would not only require deeper mathematics, but also deeper philosophy. Whether such philosophy utilizes symbols (as does mathematics) has no bearing on the depth required to investigate such a topic.

### 2.14 Probability Density Functions

A *probability density function* is a function  $f$  that is associated with a continuous random variable  $x$  and that integrates to one:

$$p(a \leq x \leq b) = \int_a^b f(x)dx = 1$$

The probability of any particular value in a *continuous probability density* has a limiting value equal to 0. That is,  $p(x = x_i) = 0$ . Continuity is defined very precisely and rigorously in mathematics using epsilon-delta definitions; however, intuitively, continuity can be thought of as drawing a curve without lifting one's pencil from the page. Because it is a probability density function, its CDF is also regarded as *absolutely continuous*, where absolute continuity implies, among other things, that we can integrate for all values of  $a, b$ . The distinction here between absolutely continuous vs. simply continuous has no bearing on the work we do in the current book. However, to the real analyst in mathematics, such a distinction is very important.

At first consideration, this may seem like an odd result. *How can any particular value in a continuous distribution have (limiting) probability equal to 0?* To make this more intuitive, consider the probability between any two particular values,  $a$  and  $b$ , to equal a number greater than 0. As the distance  $a - b$  decreases, the probability of that interval gets smaller and smaller, until in the limit the distance  $a - b$  is such a thin slice that the probability of that thin slice is, theoretically, equal to 0. Numerically, one can think of this as the impossibility of grabbing hold of a number with an infinite number of decimals; only an *interval* of numbers can be seized; any particular number in that interval is elusive. One can also liken the idea to that of *Zeno's paradox*, in which as one approaches an object, the distances are continuously “halved” on route to that object, until one gets so close to that object that the distance between the person and the object has a limiting value of zero.

If the probability distribution is of a *discrete* rather than a *continuous* variable, then it usually goes by the name of a *probability mass function*. In such distributions, we can assign a positive probability of a given value of the random variable,  $p(x = x_i) \neq 0$ , unless of course the probability of a particular value of  $x$  is, in actuality, zero. The point, however, is that the probability of any particular value does not *need* to be zero as was the case for a continuous distribution. In a discrete distribution, we can assign probabilities to specific events. In a continuous distribution, we can only assign probabilities to *intervals*. That is, in a continuous density, the “events” are, in actuality, the interval “cuts” along the real line.

An example of a discrete vs. continuous probability distribution is given in Figure 2.3.

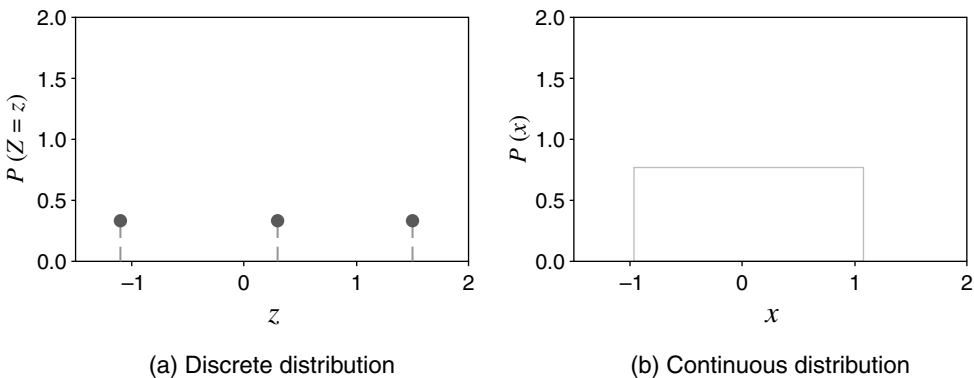


**Figure 2.3** (a) Example of a discrete distribution where each value of  $z$  can be assigned a unique probability (i.e.,  $p(Z = z_i)$ ). (b) Example of a continuous distribution where only ranges of values of  $x$  can be assigned probabilities. Used with permission of Cambridge University Press, from *Mathematics for Machine Learning* by Marc Peter Deisenroth et al, © 2019; reproduced with permission of the Licensor through PLSclear.



## 2.15 Normal (Gaussian) Distributions

Normal distributions and densities in statistics are everywhere. They are often called “Gaussian” because of their innovator, Friedrich Gauss. Not only do many physical phenomena approximate normality, but assuming normality theoretically provides a great deal of convenience in deriving new statistics and sampling distributions. Normality of a single variable is referred to as *univariate normality*, for two variables is *bivariate normality*, and for greater than two, *multivariate normality*. We survey each of these densities below.

### 2.15.1 Univariate Gaussian

The univariate normal density is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad (2.20)$$

There are two parameters in the univariate density, that of  $\mu$  and  $\sigma^2$ , where  $x$  is simply the “input” value and  $\pi$  and  $e$  are constants. The density (or crudely, “area”) under the normal distribution integrates to 1,

$$\int_{-\infty}^{\infty} N(x | \mu, \sigma^2) dx = 1$$

If we set  $\mu = 0$  and  $\sigma^2 = 1$ , the normal density is called the *standard normal density* and is the classic  $z$ -distribution that appears at the back of most introductory statistics texts and serves to reveal various areas (proportions) under the curve. Having a single “standard” distribution to refer to provides a single “scale” from which other distributions with different  $\mu$  and  $\sigma^2$  can be transformed via  $z$ -scores. When we compute a  $z$ -score,

$$z = \frac{x - \mu}{\sigma} \quad (2.21)$$

we are transforming the given distribution to have a mean equal to 0 and standard deviation equal to 1 where  $(x - \mu)^2 / \sigma^2 = (x - \mu)(\sigma^2)^{-1}(x - \mu)$ . Hence, for a given  $\mu$  and  $\sigma^2$ , the “action” of the univariate density is a function of its exponent. If we take the reciprocal of  $\sigma^2$ , that is,  $1/\sigma^2$ , we obtain the *precision* of the distribution (Bishop, 2006, p. 24). As we will see later, writing the exponent in the way of  $(x - \mu)(\sigma^2)^{-1}(x - \mu)$  will translate nicely in going from univariate to multivariate dimensions.

The transformation to  $z$  is an example of a *linear transformation* (the meaning of which we review shortly). It being an example of a linear transformation means that conversion to  $z$  does not change the shape of the distribution of the original data. This is very important! At first glance it can seem that conversion to  $z$  should *normalize* a distribution and hence give it a normal shape. But this intuition is incorrect and is a perfect example where what one feels *should* be correct is, in actuality, not. Intuition in mathematics and statistics can be a boon but at times a dangerous thing which is why proofs are usually required before a statement is considered true.

### 2.15.2 Mixtures of Gaussians

Mixture models are those that combine different distributions, typically *linearly*, at arriving at a probability estimate. Oftentimes those individual distributions are Gaussian, and hence *Gaussian mixture models* are quite common. We write a mixture of Gaussians as

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k N(\mathbf{x} \mid \mu_k, \Sigma_k) \quad (2.22)$$

The components  $k = 1$  through  $K$  of a mixture model are the different  $k$  distributions that are “mixed” into a single distribution, each having its own mean  $\mu$  and covariance matrix  $\Sigma$  and where  $\pi$  is the mixing parameter or “weight” applied for each distribution. Combined, we wish to model the probability of  $\mathbf{x}$ ,  $p(\mathbf{x})$ . The sum of mixing proportions must equal 1, that is,  $\sum_{k=1}^K \pi_k = 1$  and where  $0 \leq \pi_k \leq 1$ . We can also impose equality constraints in a mixture distribution such that certain parameters are set to be equal across distributions, such as that of equal covariance matrices in a multivariate setting (Goodfellow et al., 2016).

A Gaussian mixture model is a *universal approximator*, which, as we will see when we survey neural networks, implies that any smooth function (or density) can be approximated to any small degree of error given enough components (Goodfellow et al., 2016). Hence, given enough mixtures of distributions, function approximation becomes a certainty just as with neural networks; if we are given enough hidden layers and nodes, they too become universal approximators. Mixture models, as we will see, are also useful in methods ranging from random effects models to cluster analysis to neural networks and beyond and are much more general than simply mixing normal densities. Our example here is of normal densities only for the reason that we are currently surveying the normal distribution. Mixtures for discrete variables, for instance, might include mixing Bernoulli distributions (Bishop, 2006).

### 2.15.3 Evaluating Univariate Normality

Many methods and techniques have been proposed for evaluating the normality of distributions. Perhaps the most useful are simple graphical and visualization techniques; however, inferential methods exist as well to estimate how likely it is the current sample distribution was selected from a population distribution that is normal. Tests for univariate normality include the *Kolmogorov-Smirnov test*, the *Shapiro-Wilk test*, the *Anderson-Darling test*, and many others. According to Bilodeau and Brenner (1999), the Shapiro-Wilk test has been found to be the best omnibus test. We assume the reader has some experience with interpreting these tests, especially the Kolmogorov-Smirnov and Shapiro-Wilk tests. Further details can be found in Denis (2021a).

We can demonstrate the Shapiro-Wilk test very easily in R:

```
> shapiro.test(rnorm(100, mean = 5, sd = 3))

Shapiro-Wilk normality test

data: rnorm(100, mean = 5, sd = 3)
W = 0.99085, p-value = 0.7331
```

The null hypothesis is that the population from which the sample data were drawn is normally distributed. Since the  $p$ -value is not even close to a traditional cut-off, such as  $p < 0.05$  ( $p = 0.7331$ ), we do not have evidence to reject the null. Note that this in no way confirms the population to be normal, but it does not give us evidence to suggest it is *not* normal.

There are many ways and packages available for plotting all kinds of densities in R. For example, we can easily obtain a plot of the normal distribution in R by the following (Figure 2.4):

```
> x <- seq(-3, 3, length = 100)
> y <- dnorm(x)
> plot(y)
```

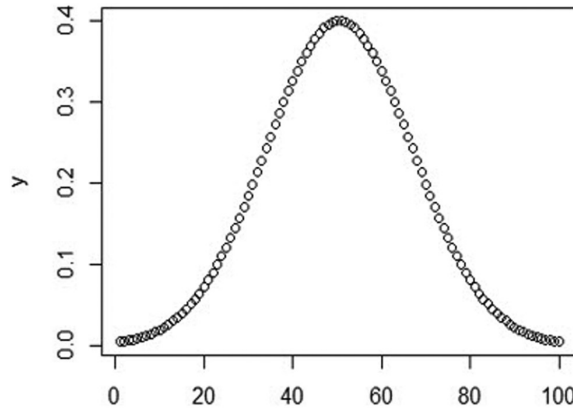


Figure 2.4 Plot of a normal distribution.

#### 2.15.4 Multivariate Gaussian

As we have just seen, in the univariate density,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

the exponent featured  $-(x-\mu)^2/2\sigma^2$ . When we generalize the univariate Gaussian to the multivariate setting, we replace observations  $x$  with vectors  $\mathbf{x}$ ,  $\mu$  with  $\boldsymbol{\mu}$ , and individual variances,  $\sigma^2$  with covariance matrices  $\Sigma$ :

$$f(\mathbf{x}) = \frac{1}{(\sqrt{2\pi})^p |\Sigma|^{1/2}} e^{-(\mathbf{x}-\boldsymbol{\mu})'\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})/2} \quad (2.23)$$

For  $\boldsymbol{\mu} = \mathbf{0}$  and  $\Sigma = \sigma^2 \mathbf{I}$ , where  $\sigma^2 = 1$ , this is the parallel setting for the standard normal distribution with  $\mu = 0$  and  $\sigma^2 = 1$ , where now  $\mathbf{I}$  is the identity matrix. But how exactly is  $\Sigma = \sigma^2 \mathbf{I}$  the analogous condition to  $\sigma^2 = 1$ ? Recall that when  $\Sigma = \mathbf{I}$ , this means that all values along the main diagonal are equal to 1 and all elsewhere is equal to 0 indicating that there is zero pairwise covariance between all variables. This implies that all variances in the multivariate density for  $\Sigma = \mathbf{I}$  are equal to  $\sigma^2$ , where  $\sigma^2 = 1$  in this case, just as  $\sigma^2 = 1$  is true for the univariate density under a standard normal distribution.

We can easily compare respective univariate and multivariate densities. In the univariate density, note that the density begins with

$$\frac{1}{\sigma\sqrt{2\pi}}$$

In the multivariate setting, this is generalized to

$$\frac{1}{(\sqrt{2\pi})^p |\Sigma|^{1/2}}$$

and now reflects *volume* as opposed to *area* as was the case in the univariate setting due to the now multidimensional space occupied by the multivariate density. The reason why it reflects volume instead of area is because we are now working in greater than two dimensions (i.e., think of a cube or cuboid, for example, instead of a square). The univariate distribution is often referred to as a *one-dimensional distribution statistically*, but such refers to the *number of variables* being studied, which is one. On paper, *geometrically*, the distribution occurs in *two* dimensions.

But what of the bivariate density? The bivariate density is simply a special case of the multivariate density for the case of two variables and hence, we find it more convenient to simply regard it as a multivariate density. However, it can also be written in the following form:

$$f(x_1, x_2) = \frac{1}{2\pi} \frac{1}{\sigma_1\sigma_2(1-\rho^2)^{1/2}} \exp \left\{ -\frac{1}{2(1-\rho^2)} \left[ \left( \frac{x_1 - \mu_1}{\sigma_1} \right)^2 - 2\rho \left( \frac{x_1 - \mu_1}{\sigma_1} \right) \left( \frac{x_2 - \mu_2}{\sigma_2} \right) + \left( \frac{x_2 - \mu_2}{\sigma_2} \right)^2 \right] \right\} \quad (2.24)$$

where  $\rho^2$  is the squared correlation between variables  $x_1$  and  $x_2$ . For further details on the bivariate density, see Johnson and Wichern (2007, p. 151).

The multivariate density is significant for many reasons, most of which parallel the reasons for the popularity of the univariate density in statistics and science more generally. However, there is an added feature of importance as regards the multivariate density, and that is, if we can assume multivariate normality, we can be assured that *any linear combination of the random vector  $\mathbf{X}$  is also normally distributed*. While this may seem like an intuitive result, it has huge implications for statistical inference in multivariate models. The theorem also implies that conditional distributions will be multivariate normal as well.

### 2.15.5 Evaluating Multivariate Normality

Assessing multivariate normality is exponentially more difficult than assessing univariate normality due to it being a higher-dimensional problem. The best advice for such evaluation is probably given by Johnson and Wichern (2007, p. 177):

It will become clear that our investigations of normality will concentrate on the behavior of the observations in one or two dimensions (for example, marginal distributions and scatter plots). As might be expected, it has proved difficult to construct a “good” overall test of joint normality in more than two dimensions because of the large number of things that can go wrong. To some extent, we must pay a price for concentrating on univariate and bivariate examinations of normality: We can never be sure that we have not missed some feature that is revealed only in higher dimensions ... for most practical work, one-dimensional and two-dimensional investigations are ordinarily sufficient. Fortunately, pathological data sets that are normal in lower dimensional representations, but nonnormal in higher dimensions, are not frequently encountered in practice.

Still, there are tests that can be used to evaluate normality, even in the multivariate domain. A generalization of the Shapiro-Wilk test can be used, as well as a more recent test called the *Henze-Zirkler test*. See Zhou and Shao (2014) for details. The consequence of not ensuring multivariate normality is that the asymptotic distribution of many statistical tests becomes much more complicated (Bilodeau & Brenner, 1999). Recall that the asymptotic distribution is that obtained as  $n \rightarrow \infty$ .

In practice, researchers often use visualization and graphical techniques to verify univariate (e.g., *Q-Q plots*, *histograms*, etc.) and bivariate normality (e.g., scatterplots), and if such are more or less *approximately* satisfied, they then make the inferential “leap” to assuming that multivariate normality holds as well. However, it needs to be emphasized that while multivariate normality implies bivariate and univariate normality, the reverse is not necessarily true. That is, univariate and bivariate normality does not automatically imply multivariate normality. Hence the aforementioned “leap” required. Univariate and bivariate normality may increase the probability of multivariate normality, but it does not necessarily imply it.

## 2.16 Binomial Distributions

When a variable has a binary outcome, such as pass vs. fail, and whose probability of a success is constant or *stationary* over a series of trials, the binomial distribution is a plausible distribution for modeling such data. A random variable  $x$  is binomially distributed if  $x$  can take on one of two binary-valued numbers. For convenience, we denote these numbers 0 and 1; however in theory they could be any two numbers. We assume that 1 occurs with probability  $\pi$  and 0 occurs with probability  $1 - \pi$ , where of course  $\pi$  is bound by limits 0 and 1 as is true of all probabilities.

The binomial density is given by

$$p(x; n, \pi) = \frac{n!}{x!(n-x)!} \pi^x (1-\pi)^{n-x} \quad (2.25)$$

where  $x$  is the number of outcomes that we are modeling,  $n$  is the total number of trials, and  $\pi$  is the probability of a success on any given trial. A classic example of the binomial in practice is modeling the probability of getting 2 heads out of 5 flips on a fair coin (i.e., where “fair” implies  $\pi = 0.5$ ). This probability is easily computed,

$$\begin{aligned} p(x; n, \pi) &= \frac{n!}{x!(n-x)!} \pi^x (1-\pi)^{n-x} \\ p(2; 5, 0.5) &= \frac{5!}{2!(5-2)!} (0.5)^2 (1-0.5)^{5-2} \\ &= 0.3125 \end{aligned}$$

Knowing that the probability of getting 2 heads out of 5 flips of a fair coin is equal to 0.3125, or about 31% of the time, such can help us evaluate the null hypothesis that a given coin is fair. Surely, if the coin is producing 2 heads out of 5 flips, this would give us no reason to suspect the coin is *not fair*. On the other hand, had the coin produced, say, 0 heads out of 5 flips, such would have yielded a probability of 0.03125. That is, the probability of getting zero heads out of five flips on a fair coin is quite low. It is so low that it might suggest we reject the null hypothesis that the coin is fair and infer the alternative that it is not, or at minimum, that the probability of heads is *not* equal to 0.5.

We can compute binomial probabilities very easily in R. For the case of 2 heads out of 5 flips with a stationary success set at 0.5, the probability is computed as

```
> dbinom(2, 5, 0.5)
[1] 0.3125
```

The mean of a binomial distribution is  $n\pi$ , that is, the number of trials multiplied by the probability of success on any given trial. For the above data, this number is  $5(0.5) = 2.5$ . That is, on 5 flips, we would expect an average of 2.5 heads. Of course, since the variable is a discrete one, this interpretation should be made with caution as it is not possible to obtain “fractions” of heads. Recall that averages do not care about fractions. The mean number of persons per household may very well be 3.4, impossible as this number is for human beings (or any discrete objects, for that matter).

The variance of a binomial random variable is equal to  $n\pi(1-\pi)$ . For example, the variance of a binomial distribution with probability  $\pi = 0.5$ , and as before  $n = 5$ , is equal to  $n\pi(1-\pi) = 5(0.5)(1-0.5) = 2.5(0.5) = 1.25$ , with standard deviation (square root) equal to 1.118.

### 2.16.1 Approximation to the Normal Distribution

The binomial distribution approximates the normal distribution for large number of trials, and in the limit, converges to the normal distribution. We can easily demonstrate in R how increasing  $n$  approximates normality via the plots in Figure 2.5.

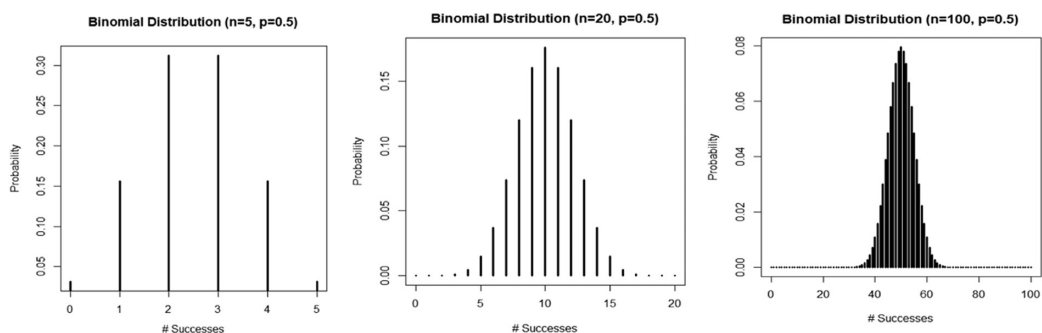


Figure 2.5. Binomial distributions; as the number of successes increases from 5 to 100, the distribution increasingly takes on a smooth and normal shape.

Notice that as  $n$  increases for the same probability  $p = 0.5$ , the distribution increasingly resembles a smooth normal distribution. This is just one example, and perhaps the simplest and easiest to demonstrate, of how virtually all distributions in statistics are related. Many distributions converge to others under sufficient conditions. The Poisson distribution, to be surveyed shortly, in the limit converges to the normal distribution as well.

## 2.17 Multinomial Distributions

The binomial distribution just surveyed is actually a special case of the more general multinomial distribution:

$$p(x_1, \dots, x_k) = \frac{n!}{x_1! x_2! \dots x_k!} \pi_1^{x_1} \pi_2^{x_2} \dots \pi_k^{x_k} \quad (2.26)$$

where now  $\pi_1^{x_1} \pi_2^{x_2} \dots \pi_k^{x_k}$  are the different probabilities associated with individual events  $x_1, x_2, x_k$ . That is, instead of having only two possibilities on the discrete variable (binomial), the variable now has several possibilities, each having their own respective probabilities,  $\pi_1, \pi_2, \dots, \pi_k$ .

## 2.18 Poisson Distributions

The Poisson density is given by

$$p(x / \mu) = \frac{e^{-\mu} \mu^x}{x!} \quad (2.27)$$

where  $e$  is a constant and  $x$  is the number of events occurring in a fixed interval of time. The expectation (i.e., mean) of the Poisson density and that of its variance are both equal to  $\mu$ . For further details on the Poisson density, see Casella and Berger (2002, p. 92).

## 2.19 Chi-Square Distributions

The chi-square density is given by

$$f(x) = \frac{1}{2^{v/2} \Gamma(v/2)} x^{[(v/2)-1]} e^{-x/2} \quad (2.28)$$

where  $\Gamma$  is the gamma function and  $v$  are the degrees of freedom of the density. The chi-square distribution of a random variable is equal to the sum of squares of  $n$  independent and normally distributed  $z$ -scores (see Fisher, 1922 for details):

$$\chi_n^2 = \sum_{i=1}^n z_i^2 = \sum_{i=1}^n \frac{(x_i - \mu)^2}{\sigma^2}$$

A final thought on statistical distributions and densities. Not surprisingly, there are many more than we have highlighted here in our brief review. Most times, the type of problem you are modeling will quite naturally dictate or strongly suggest the density you should use to model the given phenomenon. The key is recognizing from the problem you are facing which statistical distribution might be most *appropriate*. Such insight can be difficult and often depends on relying on one's experience or expertise in the given substantive area. It is not unusual, however, for an analyst to spend some time contemplating which density would be best in modeling this or that phenomenon, especially when the problem is difficult.

## 2.20 Expectation and Expected Value

When we talk about “expectation” in statistics, we are generally referring to the long-run average behavior of a random variable. Since we are dealing with a random variable, this idea necessarily invokes and implies the *probability* concept. The expected value of a function  $g$  is the sum (or

integral) of that function multiplied by its corresponding probability, for a continuous random variable,

$$E_x[g(x)] = \int_x g(x)p(x)dx \quad (2.29)$$

For a discrete random variable, the expectation is

$$E_x[g(x)] = \sum_x g(x)p(x) \quad (2.30)$$

The expectation for a discrete variable is different from that of a continuous variable since, with a discrete one, we can assign individual probabilities to values of the variable and sum up products,  $g(x)p(x)$ . With a continuous variable, products are technically also summed, but they require integral calculus to do the summation and not simply sigma as in the discrete case.

We can compute expected values of single random variables as above, but we can also compute expected values of elements of vectors and matrices as well. The expectation of a random matrix is the expectation of each of its random variables,

$$E(\mathbf{X}) = \begin{bmatrix} E(x_{11}) & E(x_{12}) & \cdots & E(x_{1p}) \\ E(x_{21}) & E(x_{22}) & \cdots & E(x_{2p}) \\ \cdot & \cdot & & \cdot \\ E(x_{n1}) & E(x_{n2}) & \cdots & E(x_{np}) \end{bmatrix} \quad (2.31)$$

Since  $E(x_{11})$  through to  $E(x_{np})$  is equal to  $\mu_{11}$  to  $\mu_{np}$ , we can write:

$$E(\mathbf{X}) = \begin{bmatrix} E(x_{11}) & E(x_{12}) & \cdots & E(x_{1p}) \\ E(x_{21}) & E(x_{22}) & \cdots & E(x_{2p}) \\ \cdot & \cdot & & \cdot \\ E(x_{n1}) & E(x_{n2}) & \cdots & E(x_{np}) \end{bmatrix} = \begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1p} \\ \mu_{21} & \mu_{22} & \cdots & \mu_{2p} \\ \cdot & \cdot & & \cdot \\ \mu_{n1} & \mu_{n2} & \cdots & \mu_{np} \end{bmatrix} \quad (2.32)$$

### 2.21 Measures of Central Tendency

Observing every data point in a data set, especially for large data sets, provides a weak summary of that data and we usually wish instead to summarize the data with one or two measures that, in some sense, “represent” the data. We emphasize “in some sense,” because no single measure could ever summarize any data *perfectly*. To understand what central tendency means with regard to a particular measure used, one must understand in some depth the measure, that is, the abstract “idea” of what central tendency might mean. Of course, if the data set consists of only say, 2–3 data points, then inspecting the data set *point by point* is well advised and it makes little sense in trying to reduce such data into a smaller set of summary statistics. Usually, however, our data sets are medium-sized to very large, and graphical or numerical summaries are required in getting to know our data.

Measures of central tendency are *data reduction measures*; they take a set of observations and reduce such data down to a single measure.



### 2.21.1 The Arithmetic Mean (Average)

The arithmetic mean is without question the most popular measure of central tendency. The arithmetic mean is given by

$$\bar{x} = \frac{1}{n} \sum_{n=1}^N x_n \quad (2.33)$$

Note that the arithmetic average consists of summing all observations, and then dividing by how many numbers went into that sum. Hence, the mean is often regarded as the *center of gravity* in a distribution. Since it is the center of gravity, it is also true that the sum of deviations from the mean is equal to 0; that is,

$$\sum_{i=1}^n (x_i - \bar{x}) = 0 \quad (2.34)$$

The fact that deviations from the mean sum to zero is one reason why such deviations are typically squared (or absolute values taken under some circumstances) when producing the variance or standard deviation.

For a random variable, the mean is given by

$$\mu = \int_{-\infty}^{+\infty} x f(x) dx \quad (2.35)$$

where  $x$  will denote a specific value of the variable and  $f(x)$  is the density under the curve for that value, and for discrete random variables,

$$\mu = \sum_{i=1}^n x p(x) \quad (2.36)$$

where  $p(x)$  will be the probability of the specific value  $x$  since with discrete random variables we can assign an individual probability to  $x$ , whereas with continuous variables recall, we cannot.

The arithmetic mean is only one of many means that can be computed on a set of data. Other means include the *geometric mean* and the *harmonic mean*. The geometric mean is given by

$$\bar{x}_g = \left( \prod_{i=1}^n x_i \right)^{\frac{1}{n}} = \sqrt[n]{x_1 \cdot x_2 \cdots x_n} \quad (2.37)$$

Note that in the geometric mean, we are taking products of values (via the product operator,  $\prod_{i=1}^n$ ) instead of sums as in the arithmetic mean and taking the root equal to the number of observations  $n$  (i.e.,  $1/n$ ).

The harmonic mean is given by

$$\bar{x}_h = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}} \quad (2.38)$$

The important point is that when someone reports computing the mean on a set of data, be sure to inquire *which* mean is being computed. Typically, when the word “mean” is used, it refers to the arithmetic mean, but not always. In the reporting of scientific research, it would be irresponsible of a researcher to not specify which mean is being reported if it were *not* the arithmetic mean. Further, though the arithmetic mean is often advertised as a measure of *central* tendency, it only reveals the center of a distribution in the sense of  $\sum_{i=1}^n (x_i - \bar{x}) = 0$ , and in no other sense. In a perfectly normal distribution, the mean will be at the center of the distribution with symmetrical and equal areas on each side of the distribution. In a skewed distribution, however, the mean will no longer be at the “center” of the distribution and will simply fall where the condition  $\sum_{i=1}^n (x_i - \bar{x}) = 0$  is satisfied. For example, consider the location of the arithmetic mean in the distribution in Figure 2.6.

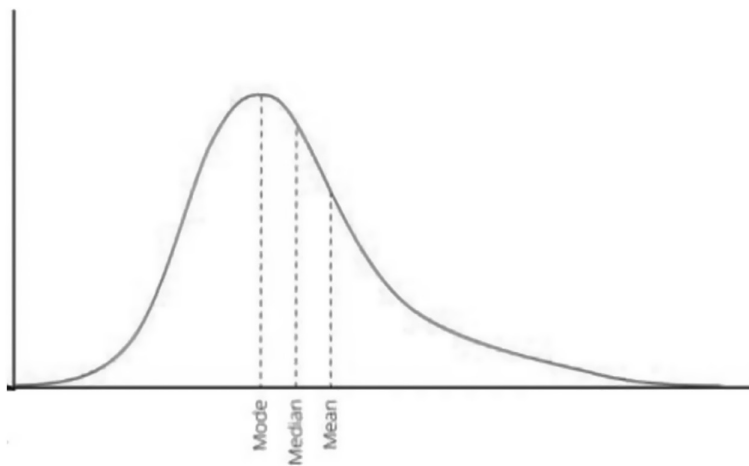


Figure 2.6 A distribution with a slight positive skew, where the mean is being “pulled” by larger scores toward the upper end of the distribution, where  $\sum_{i=1}^n (x_i - \bar{x}) = 0$  is satisfied. Key to note is that  $\sum_{i=1}^n (x_i - \bar{x}) = 0$  does not necessarily imply  $\bar{x}$  is in the geometric “middle” of a distribution, nor does meeting this condition necessarily imply the distribution is normal in shape.

Note that the mean is *not* at the center of the distribution, however it is still defined as a measure of “central tendency.” If you memorize “*mean = central tendency*,” you will never truly understand the mean! The only way to understand it is to inspect its structure and appreciate what it actually *does*. This is true of all equations and formulas in statistics and machine learning. *Never rely on “verbal” definitions to describe them; only their equations and formulas reveal what they actually accomplish.*

It is a good computing exercise to compute the mean the “long way” in R or Python (try devising a function to generate the mean). In Python, we can easily compute the mean by using `numpy.mean()`:

```
import numpy as np
x = [10, 20, 30]
np.mean(x)
Out [56]: 20.0
```

The mean of the observations 10, 20, and 30 is, not surprisingly, equal to 20.

#### 2.21.1.1 Averaging Over Cases: Why Thinking in Terms of Averages Can Be Dangerous

Means and averages are much more than about the above computations. Though averaging incorporates all deviations from a center point, it simultaneously ignores individual information. If averaging is held as a *policy* to govern by, then it will naturally and necessarily ignore important individual cases. The “spirit” of an average need not be numerical either; it can be entirely conceptual as the following example illustrates.

Consider the case of canine vaccines. Many states require all dogs to be vaccinated routinely every one to three years. However, there is increasing evidence, some published, and some based on clinical observation and experience, that over-vaccinating may have detrimental health effects. If this is the case, then why have a blanket policy of vaccinating all dogs every one to three years? The answer is an easy one, because of *convenience and a lack of attention paid to each particular case*. Since the goal of vaccination is to not only protect any individual dog but also to offer protection to all human beings, the policy of vaccination is very general. But this greatly ignores individual cases. Not all dogs are the same and not all dogs have the same behavior patterns. For the dog whose owner doesn’t know where the dog is at 11 p.m. as the dog roams about the neighborhood searching for mischief and wrestling with potentially rabid foxes and raccoons, the one-to-three-year vaccination policy is a good one! However, for the dog who is primarily an “inside dog” who goes outside only periodically and is in virtually no danger of contracting the disease due to hyper-vigilant supervision by his owner, the one-to-three-year policy is overkill *for that particular dog*. Averaging makes things “easy” and allows a *blanket policy*, and essentially discriminates against cases that do not “fit” into requiring the policy. The law says your 11-year-old senior dog requires vaccination, yet science may suggest doing so is a bad idea because the dog is elderly, has medical conditions, and in all probability the prior vaccination is probably still providing immunity (a titer test for immunity may be a better option). Given such risk factors, vaccinating again would be somewhat reckless, yet by the “easy” and convenient policy of averages, the law often does not take into account these individual cases.

The moral of the story is that while thinking (and computing) in terms of averages can be productive, it can also be very dangerous. *Individual case analysis* is almost always better, despite how more expensive and time-consuming such an analysis may be. *Means and averages are balance points, yet often, “balancing” should not be of primary interest*. Case studies and more focused analysis should be the priority when the data, resources, and time allow it. Indeed, this is one criticism against numerical “summative” statistical research in general, that it focuses too heavily on the *common person* rather than the *individual person*. In clinical settings, case studies will typically do away with averages and instead focus on the individual.

Averages can also be dangerous when it comes to interpreting data. For instance, our interpretation of the mean implicitly assumes *linear interpolation* between values. For example, consider the mean of two numbers, 2 and 4. The mean is, of course, 3. Denoting 3 as the “center of mass” assumes that the distance and importance between 2 and 3 and 3 to 4 is the same, which in this case is linear. Linear interpolation, in general, can be risky because the intermediate values may not be

linear, but are often assumed to be out of convenience. This is a simple idea but can have drastic and important consequences when interpreting real data.

### 2.21.2 The Median

Another measure of central tendency is that of the *median*. The median is the data point that divides the ordered data set into exactly two equal parts. The computation of the median will depend on how many observations are in the data set. For instance, if  $n$  is odd, then the location of the median  $\tilde{y}$  is equal to  $(n+1)/2$ , yielding a median value equal to

$$\tilde{y} = y_{\left(\frac{n+1}{2}\right)} \quad (2.39)$$

If  $n$  is even, then the location of the median is between  $(n/2)$  and  $(n/2)+1$ , yielding a median of

$$\tilde{y} = \left[ \left( y_{n/2} \right) + \left( y_{(n/2)+1} \right) \right] / 2 \quad (2.40)$$

While means can be said to be sensitive to extreme observations, medians are not. Medians, and other median-like statistics (such as quartiles), are said to be *robust* to extreme or outlying observations. Because medians only consider the “middle” value of the distribution, one can easily change other values in the data set while still retaining the same median. For example, the median of the data set 9, 10, 11 is 10, while the median of the data set 9, 10, 100 is also equal to 10. If a researcher reports the median of their data, ask to see a plot! Resist engaging in the cognitive bias that the median (or the mean, for that matter) implies a normal distribution. “Normality” is a theoretical notion. Empirical normality never truly exists; it can only be *approximated* by the theoretical ideal, that is, by the mathematical definition of a normal density.

Recall that the arithmetic mean is not robust to extreme scores or outliers. That is, since the mean is a weighted sum of deviations, even changing one value in the distribution will alter the mean. Is this a benefit or drawback? Such will depend on what you want the measure to accomplish. The scientist is, of course, interested in the data, but without knowledge of the functions (e.g., mean, median, etc.) *imposed* on such data, statistics computed on that data become amenable to misunderstanding and misinterpretation.

## 2.22 Measures of Variability

Measuring the variability in an empirical or theoretical distribution can take many forms, some better than others depending on the context of the situation and the characteristics of the data. Measures such as the *range*, *interquartile range*, which is equal to the 75th minus the 25th percentile, *semi-interquartile range*, equal to the 75th minus the 25th percentile divided by 2, and others are useful as “median-like” measures of variability. Being median measures, they are insensitive and robust to outlier observations, and hence, if such influence is not desired when summarizing a data set, these types of measures come highly recommended. Most commonly, however, we wish to calculate more sensitive measures that utilize the arithmetic mean as their locus or center from which we take deviations. For this reason, the variance and standard deviation dominate statistics as measures of variability, which we now review.

### 2.22.1 Variance and Standard Deviation

The sample variance is defined as

$$S^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n} \quad (2.41)$$

where  $\bar{y}$  is the mean of the distribution and  $y_i$  any particular data point in that distribution. The sample size is given by  $n$ . The sample variance requires us to sum squared deviations from the mean and take the average of that sum. The sample variance with  $n$  in the denominator is a *biased estimator* of the population variance,  $\sigma^2$ ,

$$\sigma^2 = \frac{\sum_{i=1}^n (y_i - \mu)^2}{n} \quad (2.42)$$

where now  $\mu$  is the population mean to replace  $\bar{y}$  in the aforementioned sample variance. Formally,  $S^2$  is a biased estimator because its expected value does not equal the population parameter,  $\sigma^2$ . More succinctly, we write  $E(S^2) \neq \sigma^2$ . To make it an *unbiased estimator*, we can correct the denominator by subtracting a *degree of freedom*, which yields,

$$s^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1} \quad (2.43)$$

and now makes it true that  $E(s^2) = \sigma^2$ . The positive square root of the sample variance is the *sample standard deviation*  $s$ ,

$$s = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1}} \quad (2.44)$$

which provides a measure of average deviation from the mean, now no longer in squared units as was true for the variance. As an example, if the standard deviation for a distribution is computed to be equal to 5, we can conclude that *on average, scores deviate from the mean by five units*. However, without knowing the characteristics of the distribution and whether it is normal or not, we cannot use the normal density to compute proportions under the curve. That is, we cannot automatically assume that nearly 68% of cases fall within +1 or -1 units (i.e., a single standard deviation on either side of zero). We need to know more about the *shape* of the distribution in order to make such statements.

The variance and standard deviation so far discussed are for empirical variables, not theoretical random variables. For a continuous random variable, the variance is given by

$$\sigma^2 = \int_{-\infty}^{+\infty} (x - \mu)^2 f(x) dx \quad (2.45)$$

If the variable is *discrete*, then we can associate a probability  $p_i$  with any particular value  $x$  in the distribution, yielding

$$\sigma^2 = \sum_{i=1}^n (x_i - \mu)^2 p_i(x) \quad (2.46)$$

In Python, the variance can be computed by `numpy.var()`. In R, the `var()` function will return the variance.

### 2.22.2 Mean Absolute Deviation

The reader may wonder why squared deviations of the kind  $(y_i - \bar{y})^2$  are taken when computing the standard deviation. Indeed, we must take at least *some* function of the deviations since taking the sum of deviations,  $\sum_{i=1}^n (y_i - \bar{y})$ , will always result in a value of zero, no matter the extent of variability or dispersion present in the distribution. That is, recall it can easily be shown that  $\sum_{i=1}^n (y_i - \bar{y}) = 0$  for any conceivable distribution, which thus makes it a poor measure of dispersion. Why do we always get zero? The reason is because deviations from the mean are made up of both positive and negative deviations which *balance out* when taking the sum. By squaring deviations, we eliminate the negatively-signed differences and hence obtain a value greater than zero *only if there is indeed variability present in the distribution*. If there is no variability, then  $\sum_{i=1}^n (y_i - \bar{y})^2 = 0$  will be true but only because there is indeed no variability, not because it is a constant for any distribution of data we may consider.

Squaring deviations is not the only operation that will negate negative deviations. We could have also taken a simpler function, that of absolute values of deviations. That is, taking absolute-valued deviations,

$$\sum_{i=1}^n |y_i - \bar{y}|$$

would have also generated a value greater than zero only in the situation where there is variability in the distribution. So why take squares? Why not take absolute deviations instead? It turns out that taking squared deviations is mathematically more tractable and facilitates differentiation easier than would absolute values. However, it should be emphasized that taking absolute-valued deviations is permissible and is done in many contexts, especially those in which outliers in the distribution may make the squared sum especially large. That is, squaring a deviation magnifies its deviation much more than taking an absolute value. For example, a deviation of 4 has a squared value of 16, whereas it only has a value of 4 in absolute value. Hence, if you wish to minimize the influence of outlying data, then taking absolute deviations may be preferable in some cases to taking squared deviations. There is a whole literature on taking absolute-valued deviations in a variety of statistical and machine learning methods. Such a procedure is usually called *least-absolute-value (deviation) regression* in a linear model context.

### 2.23 Skewness and Kurtosis

The *skewness* of a distribution is a measure of the general shape of that distribution away from the typical bell-shaped normal distribution and is given by

$$\gamma = \frac{E[(y_i - \mu)^3]}{(E[(y_i - \mu)^2])^{3/2}} \quad (2.47)$$

A distribution that is positively skewed has a tail that stretches out toward the right (i.e., higher scores), while a distribution that is negatively skewed has a tail that stretches out toward the left (i.e., lower scores). A distribution with an absence of skew may have a single mode (i.e., the classic normal distribution) or may have several modes (e.g., a bimodal distribution). *A lack of skew in a distribution should not automatically be associated with or imply a normal shape.* For example, the distribution in Figure 2.7 exhibits virtually no skew.

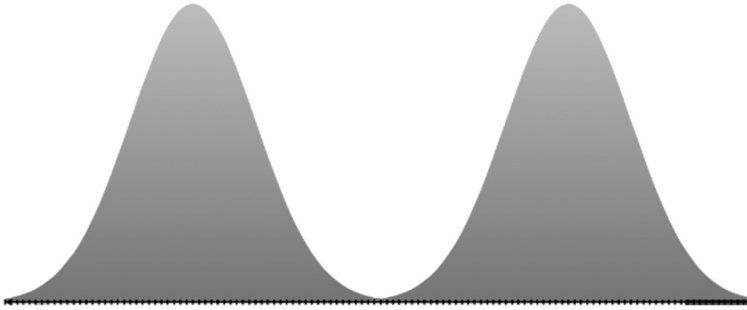


Figure 2.7 Bimodal distribution exhibiting virtually no skew. It is a mistake to assume that a distribution with no skew necessarily implies a unimodal normal distribution.

To reiterate, it can be easy to assume that upon hearing a distribution has no skew to visualize a normal distribution with a single mode. However, this could be in error, as the plot in Figure 2.7 reveals. *An absence of skewness does not necessarily imply a single mode.* This is one more reason to always plot data. Numerical summaries are usually never enough to draw strong conclusions about distributional characteristics.

The *kurtosis* of a distribution measures the general “tailness” of that distribution (Upton & Cook, 2014) and more succinctly the behavior of the distribution’s tails with regard to how “heavy” they are, meaning how much mass exists in the tails (DeCarlo, 1997). It is given by

$$k = \frac{E[(y_i - \mu)^4]}{(E[(y_i - \mu)^2])^2} \quad (2.48)$$

A distribution is said to be *mesokurtic* if the distribution exhibits kurtosis typical of a bell-shaped curve, *platykurtic* if the distribution has lighter tails and is often flatter toward the center of the distribution when compared to the normal, and *leptokurtic* if the distribution exhibits heavier tails and is often more narrow in the middle of the distribution, typically making it more “peaked” than what would be expected in the normal density. While degree of “peakness” of a distribution is not the formal definition of kurtosis, the heaviness of tails often to some extent (though definitely not *always*) corresponds to whether the distribution is wider or more narrow toward the center of the distribution. However, associating kurtosis entirely with the peakness of a distribution has largely been jettisoned.

### 2.24 Coefficient of Variation

The *coefficient of variation* is yet another measure of dispersion in a data set, and essentially “scales” the standard deviation  $\sigma$  by the value of the mean,  $\mu$ . It can be useful in this regard to know how the standard deviation for a set of data (sample or population) relates to the size of that data’s mean. Hence, if we report the standard deviation equal to 2, knowing what the mean is can help us assess the extent to which a standard deviation of 2 is considered “big” or “small.”

The coefficient of variation is given by

$$CV = \frac{\sigma}{|\mu|} \quad (2.49)$$

where  $\sigma$  is the population standard deviation and  $|\mu|$  is the absolute value of the population mean. As an example, for  $\sigma = 5$ , the coefficient of variation when  $\mu = 5$  is much smaller (i.e., 1) than if  $\mu = 1$  (i.e., 5). Hence, comparing standard deviations of distributions without incorporating respective means may, in some cases, be misleading, and a useful practice when comparing variability in distributions is to also incorporate the coefficient of variation into the discussion.

### 2.25 Statistical Estimation

Classical (and Bayesian, for that matter) statistics is all about *estimation*. Unless we have entire populations, we cannot know, for certain, the parameters of those populations and instead must estimate them using sample data. Recall that if you have the population data at your disposal, there is nothing to estimate, as one can simply compute descriptive statistics on the parameters of interest. This is sometimes misunderstood by newcomers to statistics, many of whom, upon learning about significance tests and  $p$ -values become in a hurry to conduct such tests on every data situation they encounter, all too proud to report  $p$ -values! However, a significance test on population parameters provides no additional insight beyond observing the parameters themselves. Significance tests are only meaningful when performed on sample data and when we wish to know something about the population parameters from which the sample data was presumably collected.

We say that  $\hat{\theta}$  is an estimator of the population parameter  $\theta$ . More precisely, it is a *point estimator* because it seeks to estimate one particular and specific value of the parameter,  $\theta$ . In general, an *estimator is any function of the sample data that is used to estimate a population parameter*. When we apply the estimator to some data and obtain a value for it, for instance, applying the mean estimator and obtaining a numerical mean, we say that the *estimator* has produced an *estimate*. Point estimators are almost always *statistics* (since they are computed on samples), but statistics are not always point estimators. As an example, if we compute the mean salary of a university department, such a mean is a descriptive statistic on the sample; however, it is not a point estimator because we are not using it to estimate any parameter.

An estimator is *unbiased* if the expectation of  $\hat{\theta}$  is equal to  $\theta$ , that is,

$$E(\hat{\theta}) = \theta \quad (2.50)$$

In contrast, an estimator is *biased* if the above equality does not hold,

$$E(\hat{\theta}) \neq \theta \quad (2.51)$$



with the amount of bias being equal to the difference between the expectation and the actual parameter value:

$$\text{Bias} = E(\hat{\theta}) - \theta \quad (2.52)$$

Many estimators exhibit bias in samples but are not *asymptotically* biased. That is, as sample size grows indefinitely, bias eventually vanishes. More succinctly, an estimator is asymptotically unbiased if bias tends toward zero as sample size increases toward infinity. We can express this more formally as,

$$\lim_{n \rightarrow \infty} (E(\hat{\theta}) - \theta) = 0$$

where in the above recall that  $E(\hat{\theta}) - \theta$  is the degree of bias present. As  $n$  increases, bias increasingly disappears, and is nonexistent in the limit. That is,  $E(\hat{\theta}) = \theta$  in the limit as  $n \rightarrow \infty$ .

The *variance* of an estimator is a measure of how much the estimate would change if we changed or perturbed the data. That is, in repeated sampling where we experience random fluctuations in sample data, the variance of an estimator captures how much the estimate changes as a result of this fluctuation. We define the variance of an estimator as:

$$\text{Var}_{\hat{\theta}} = E(\hat{\theta}^2) - (E(\hat{\theta}))^2 \quad (2.53)$$

It seems intuitive to conclude that favorable estimators are those that have minimal to no bias while simultaneously having minimal variance. In most cases, this is the ideal to shoot for. However, as we will see later in the book, such estimators are not always most favorable for all situations. There are times where purposely allowing bias into an estimator is actually a “good thing” if it helps reduce variance. Of course, the most ideal situation would be to both reduce variance and reduce bias, but this is usually an unattainable goal. As bias decreases, we usually experience an upshoot in variance, and as variance decreases, an upshoot in bias. This *trade-off* is well known in statistical circles and not surprisingly goes by the name of the *bias-variance trade-off*. Biased estimators include those of the *ridge* and *lasso*, which are detailed later in the book. For now, we unpack in more detail the bias-variance trade-off.

## 2.26 Bias-Variance Trade-Off

We just mentioned that two things we need to be concerned about regarding an estimator are its bias and variance. We can unpack this idea into the following fundamental identity:

$$E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\varepsilon) \quad (2.54)$$

where  $E(y_0 - \hat{f}(x_0))^2$  is the expected test mean squared error (MSE) at the value of the predictor,  $x_0$ . It is important to realize here that the variance and bias in this context refer to the *statistical learning method or model*, not the data. That is, we are not interested in the variance of the sample data in this context; we are interested instead in the variance generated by using such and such statistical *methodology or model estimate*. Variance is how much the estimated function,  $\hat{f}$ , would

change if a different training data set was used, or if the current one was “perturbed,” and bias is how “off” the learning method is in actually capturing the true function,  $f(x_0)$ . The more flexible the method, the higher the variance, while the lower the bias. The relative change of these two quantities will influence test MSE. As flexibility increases, bias initially decreases faster than the variance increases (James et al., 2013). However, beyond this initial state, an increase in flexibility impacts the bias much less but instead increases the variance, at which point test MSE likewise increases. The objective is to have both low variance and low bias as much as possible; however, a reduction in bias often results in an increase in variance, and vice versa.

### 2.26.1 Is Irreducible Error Really Irreducible?

It is typically held that variance of a statistical method,  $Var(\hat{f}(x))$ , can be mitigated or reduced; however,  $Var(\varepsilon)$  is considered to be *irreducible variance*, which implies it cannot be mitigated and we are essentially “stuck” with it. That is,  $Var(\varepsilon)$  is sometimes regarded as a *constant* that cannot be decreased. But is  $Var(\varepsilon)$  truly irreducible? No! Yes, we can find a better function to fit, that is,  $\hat{f}(x)$  to better approximate the “truth” in the form of  $f(x)$ , which will get us closer to  $E(y - \hat{f}(x_0))^2 = 0$ , but we can also reduce  $\varepsilon$  via *measurement*, or at minimum, make  $\varepsilon$  less *variable*. It is only “irreducible” in the sense that it may be independent of the function one is fitting, but it is not irreducible in the sense of simply reducing the overall squared error. That is,  $\varepsilon$  exists not solely as a result of not fitting the “correct” function or using a better statistical learning technique, but can decrease simply by an improvement in *measurement*. Reducing  $Var(\varepsilon)$  becomes a realistic possibility.

Are we theoretically able to get  $\varepsilon$  completely to zero? Probably not, but such will depend on what we consider  $\varepsilon$  to actually “be” at a deeper philosophical level. If one believes in *true chance* or *true error*, then it may be impossible to get error down to zero and we will always have at least *some* fluctuation. As noted in James et al. (2013, p. 19), “the irreducible error will always provide an upper bound on the accuracy of our prediction for  $Y$ . This bound is almost always unknown in practice.” While this is true, our point is merely that this does not take into consideration whether error can *theoretically* ever go to zero. More formally, we may write,

$$\lim_{Var(\varepsilon) \rightarrow 0} = E(y - \hat{y})^2 = [f(x) - \hat{f}(x)]^2$$

That is, as  $Var(\varepsilon) \rightarrow 0$  in the limit, we simply obtain the squared difference between the observed  $f(x)$  and estimated  $\hat{f}(x)$  functions (note that we dropped  $Var(\varepsilon)$ ). This “ideal” is theoretically possible as we find ways to decrease  $\varepsilon$  and its  $Var(\varepsilon)$ . While this is a hugely interesting topic on a psychometric and philosophical level, we will not explore the issue further here.

## 2.27 Maximum Likelihood Estimation

Maximum likelihood is, historically, the most common method of estimating parameters in a variety of statistical models and contexts. It is due primarily to Fisher (1922) though in concept at least, its origins and germination span several hundred years before Fisher (Stigler, 1986). The idea of maximum likelihood is very simple. Suppose you have observed data in the form of a sample. Maximum likelihood asks the question:

*What values of parameters should we choose, that would maximize the likelihood function of such data?*

Note carefully that the data have already been observed. The question we are asking is what value for parameters of interest *would have made such data most likely*?

As a simple example, consider flipping a coin ten times and observing five heads. The five heads constitute the observed data. What parameter value for the probability of heads would have given this observed result a “maximum likelihood” of occurring? The answer, of course, is the probability of heads equal to 0.5, or, more succinctly,  $p(H) = 0.5$ . An alternative parameter value such as  $p(H) = 0.4$  would have made the data *less likely* than  $p(H) = 0.5$ , and hence  $\hat{\theta} = 0.5$  is the maximum likelihood estimator in this case. The estimates for the parameter values (e.g.,  $p(H) = 0.5$ ) are referred to as *maximum likelihood estimates*.

More formally, the maximum likelihood estimator, or MLE, is defined by

$$\hat{\theta}_{mle} \triangleq \arg \max_{\theta} p(D | \theta) \quad (2.55)$$

which simply expresses the argument value (i.e.,  $\arg$ ) that makes the data  $D$  most likely given the parameter value,  $\theta$ . That is, maximum likelihood (ML) provides which value for  $\theta$  makes  $p(D | \theta)$  (the *likelihood function*) most probable. The logarithm of a function is a monotonically increasing function, which means that whether we take the likelihood or the log of the likelihood essentially amounts to the same thing in terms of assessing the size of likelihoods. For technical reasons, and for convenience, taking the log is useful, and hence the log-likelihood is given by

$$\log L(\theta) = \log p(D / \theta) \quad (2.56)$$

In practice, it will often be the negative of the log-likelihood that is *minimized* instead of the log being maximized,

$$\hat{\theta}_{mle} \triangleq \arg \min_{\theta} -\log p(D / \theta) \quad (2.57)$$

which is generally known as the *error function* (Bishop, 2006).

All of the above is quite general. For instance, we can unpack these elements for the multivariate situation specifically in which we desire maximum likelihood estimators. Hence, suppose we seek maximum likelihood estimates of  $\mu$ , the multivariate mean, and  $\Sigma$ , the covariance matrix, that will maximize the likelihood of the data. In the multivariate setting, the aforementioned estimates are found by

$$\hat{\mu} = \bar{y}$$

for the multivariate mean vector, and

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})(y_i - \bar{y})'$$

which can also be given by  $\frac{n-1}{n}\mathbf{S}$ , where  $\mathbf{S}$  is the sample covariance matrix. For justification for these particular maximum likelihood estimators, see Rencher and Christensen (2012, p. 99). Maximum likelihood requires us to select the parameters that *maximize* the joint density evaluated at the observations (Johnson & Wichern, 2007). The maximizing parameter values, as before, are called *maximum likelihood estimates*. That is, the maximum likelihood estimates of  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  are those values,  $\hat{\boldsymbol{\mu}}$  and  $\hat{\boldsymbol{\Sigma}}$  that maximize the likelihood function  $L(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ . In practice, as in the univariate case, often the negative of the log of the likelihood is computed. As mentioned, maximizing the likelihood is equivalent to minimizing error (Bishop, 2006).

### 2.27.1 Why ML Is So Popular and Alternatives

Maximum likelihood estimators possess many favorable properties. It is the default estimation for many if not most statistical and machine learning problems. For instance, it is commonly used in neural networks (Goodfellow et al., 2016).

Despite its popularity, maximum likelihood is by no means the *only* method of estimation in statistical modeling and machine learning. Numerous other methods have been advanced to accommodate data situations for which assuming multivariate normality is unrealistic. For instance, *quasi-maximum likelihood* (QML) is a method of estimation advanced in structural equation modeling featuring nonlinear terms such as quadratic effects (Klein & Muthén, 2007). *Two-stage least-squares* estimation for nonlinear SEM models has also been advanced (Holst & Budtz-Jorgensen, 2020). *Adaptive Gaussian Quadrature* (AGQ) has also been proposed for estimating nonlinearity in SEM models. The *method of moments* is an alternative approach to ML estimation when the solution is either unavailable or computationally difficult (Murray, 2022). *Online recursive estimation* is yet another possibility (Murray, 2022). All of these estimation methods are complex and discussions of them could fill an entire chapter or even book.

The key takeaway from this discussion is that when it comes to estimation methods, though maximum likelihood is probably the most popular method, there are numerous alternative methods which are already in existence and others which are being developed in statistics and machine learning. Hence, when one comes across the term “estimation,” one should be curious to learn what *type* of estimation is being referred to as it will not always be maximum likelihood.

### 2.27.2 Estimation and Confidence Intervals

In classical statistics, we can either attempt to estimate a particular point in a distribution, such as the arithmetic mean, or we can attempt instead to estimate an *interval* of values. Theoretical statistics tells us that the expected value of the sample mean,  $\bar{y}$ , is equal to the population mean,  $\mu$ . That is,  $E(\bar{y}) = \mu$ . However, we also know that the probability of “hitting” the precise value of the population mean has a limiting value of zero. Hence, we seek usually to build an *interval* of values around the sample estimator and instead of seeking *point predictions*, we will use an *interval estimator*, the most common of which is the confidence interval.

The *confidence for the mean* is given by

$$\bar{y} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}} \quad (2.58)$$

where  $z_{\alpha/2}$  is the critical value of  $z$  given the desired accuracy for the confidence interval. For a 95% interval, that value is 1.96, and so we have

$$\bar{y} \pm 1.96 \frac{\sigma}{\sqrt{n}}$$

Recall that seldom do we actually know the value of  $\sigma$  and must instead estimate it based on sample data. When we use the sample standard deviation  $s$  as an estimator of the population standard deviation  $\sigma$ , the mean is no longer distributed as  $z$ , but instead is distributed as a  $t$ -statistic, and hence the confidence interval becomes:

$$\bar{y} \pm t_{\alpha/2} \frac{s}{\sqrt{n}} \quad (2.59)$$

Confidence intervals are by no means limited to means. Interval estimators can be used for a variety of statistics. In many cases, analytical solutions tell us how to construct the standard errors for such intervals. In other cases, standard errors are intractable and difficult or even impossible to calculate. When we cannot compute standard errors directly, one option is to *bootstrap* samples of observations and determine interval bounds via repeated sampling. We discuss the bootstrap shortly.

Be sure to note that as was true for the point estimator, an interval estimator is the random variable. The parameter that is being estimated is not the random variable. While there are a few different ways of interpreting confidence intervals in practice, the most correct way is to refer to the *procedure* for computing such intervals of which one interval has been calculated. Hence, the most correct interpretation of a computed 95% interval on data becomes

*The probability that the given interval, based on the procedure used to compute the interval, covers the parameter, is 0.95. Repeating the procedure of calculating intervals this way would result in 95% of them covering the true parameter.*

There is a whole literature on the correct interpretation of confidence intervals and how researchers presumably always get it wrong. Yes, formality matters here, but the above is not the only way a confidence interval can be conceptualized, especially in a real research setting. The above interpretation is rather awkward when it comes to interpreting real research, but is arguably near the most theoretically correct interpretation. For a brief discussion on this point and a brief argument for why competing ways of interpreting confidence intervals may be useful in practice, consult Denis (2020).

## 2.28 The Bootstrap: A Way of Estimating Nonparametrically

Perhaps more than any other statistical technique, the bootstrap is the one that has benefited the most from the computer age revolution and the speed and facility that such computation provides. Though theoretically the bootstrap would still be possible without computers, it would be so laborious and impractical computationally that it is fair to say the technique would have likely forever been abandoned and have never risen to fame. Thanks to computation, however, the bootstrap has, in the world of statistics and machine learning, become a “household name.”

The origins of the bootstrap are usually credited with Efron (1979) and is actually a special case of what is generally known as the *jackknife* which came before it, but is far more applicable and generalizable than the jackknife. The jackknife itself can be thought of as a *linear expansion method* (called the *delta method*) for approximating the bootstrap (Efron, 1979). As noted by Efron and Hastie (2016), “Before computers, applied statisticians needed to be Taylor series experts in

laboriously pursuing the accuracy of even moderately complicated statistics” (p. 155). The jack-knife, originating in 1957, was a first step toward a computational approach to standard errors.

The basic idea of the bootstrap is to capitalize on computing power to draw repeated samples from a sample that, for the given bootstrap scenario, serves as a *pseudo-population*. We include the bootstrap here in our discussion of estimation as it is yet another way to estimate parameters of a distribution; while the methods we have just surveyed do so *parametrically* through time-tested analytical formulae, the bootstrap is a *nonparametric* way of estimation through simulations of resampling using the computer.

Even more generally, bootstrap methods are a special case of the wider family of *Monte Carlo methods*, which is a general name given for *resampling*. To be clear, the bootstrap is not *equivalent* to Monte Carlo methods by any stretch, as many other techniques that utilize Monte Carlo sampling exist. However, in applied statistics and machine learning circles the bootstrap is undoubtedly one of the more popular *applications* of the general Monte Carlo technique. As a result of the bootstrap typically being used for estimation, we have included a discussion of it here instead of under Monte Carlo techniques, but including it under a general discussion of Monte Carlo techniques would not have been wrong.

### 2.28.1 Simple Examples of the Bootstrap

As discussed, *the bootstrapped sample is to the original sample as the original sample is to the population*. Why would we ever want to sample from a sample? There are a few reasons, but one primary reason is that obtaining things like standard errors and such *analytically* is sometimes not feasible and in many cases, actually impossible. True, we know what the standard error is for the mean, for instance, equal to

$$\frac{\sigma}{\sqrt{n}}$$

but we are less sure, for instance, what the standard error is for the *median* (i.e., the standard error for the median is less *exact*). That is, the standard error of the median is less *analytically available*. However, thanks to the bootstrap, we can assess “empirically” the standard deviation of medians (i.e., the standard error of the median). This is a chief benefit to being able to bootstrap, that of being able to “simulate” statistics empirically via resampling that have no analytical sampling distribution or that deriving such sampling distributions proves very difficult. In theory, we could “invent” a new statistic tomorrow that has no known sampling distribution and simply bootstrap the statistic via resampling to determine its “behavior” over the long run. This is perhaps the biggest reason why the bootstrap is such a “big deal” in statistics.

More formally, assume we have a training set, denoted by  $\mathbf{Z} = (z_1, z_2, \dots, z_N)$ , where any given  $z_i$  corresponds to data on  $x, y$ , that is,  $z_i = (x_i, y_i)$ . Imagine now we resample from  $\mathbf{Z}$  a total of  $B = 100$  samples. We will refer to these samples as the *bootstrapped samples*. To use our example of the standard error, we would compute standard deviations on each of the bootstrapped samples and observe standard deviation behavior over the 100 samples. This would denote the standard error we want to compute. However, we need not necessarily be computing standard errors; we could be computing virtually *anything* on these bootstrapped samples. As mentioned, bootstrapping is widely applicable to virtually any statistic, even those for which deriving sampling distributions is impossible.

As an example, we could compute linear regressions on the bootstrapped samples and observe the behavior of the linear equations across the samples, or any other models or statistics. This is the

power of the bootstrap and Monte Carlo simulation in general; *it allows us to resample repeatedly and build a distribution of functions of the data, whatever those functions may be*. The bootstrap in this regard is very flexible and has wide application. It is *nonparametric*, meaning it assumes nothing about the population, making it extremely flexible and well-suited for creative ventures pertaining to whatever you wish to calculate on the repeated samples. Bootstrapping *allows you to create an actual empirical distribution of computed statistics instead of deriving a theoretical one analytically*.

More formally, consider once more  $\mathbf{Z} = (z_1, z_2, \dots, z_N)$ . Suppose we compute some quantity on  $\mathbf{Z}$ , that is, some function of  $\mathbf{Z}$  which we will appropriately denote  $f(\mathbf{Z})$ . Suppose that function  $f(\mathbf{Z})$  is the variance of  $\mathbf{Z}$ , that is,  $Var(\mathbf{Z})$  defined as

$$f(\mathbf{Z}) = Var(\mathbf{Z}) = \frac{\sum_{b=1}^B (\mathbf{Z}^{*b} - \bar{\mathbf{Z}}^{*b})^2}{B}$$

where  $\mathbf{Z}^{*b}$  is a particular computed bootstrapped sample and  $\bar{\mathbf{Z}}^{*b}$  is the *mean* of all bootstrapped samples. Notice carefully that  $Var(\mathbf{Z})$  here is just an ordinary variance but computed on the  $B$  bootstrapped samples. All of this looks familiar based on our earlier study of the variance. For instance, if we wished for the more typical estimator, we could use  $B-1$  instead of  $B$  in the denominator,

$$f(\mathbf{Z}) = Var(\mathbf{Z}) = \frac{\sum_{b=1}^B (\mathbf{Z}^{*b} - \bar{\mathbf{Z}}^{*b})^2}{B-1}$$

It is easy to see then that  $Var(\mathbf{Z})$  is yielding a measure of variance around bootstrapped samples.

### 2.28.2 Why Not Bootstrap Everything?

As we have seen, the bootstrap is quite universal and general and can be applied in many different contexts. Indeed, we can literally bootstrap virtually any function on a sample if we have sufficient reason for doing so. It does not need to be something familiar, such as a mean, median, or other well-known statistic. This is precisely the advantage of the bootstrap, in that it allows us to estimate statistical quantities for which we may have literally no theory to guide us and for which there may be no chance of ever knowing the sampling distribution without resampling.

So, why isn't the bootstrap used more often? Why don't we bootstrap virtually *everything*? The reason is that except when assumptions are violated, analytically-derived solutions may produce more favorable results than resampling. Computer simulations can be inexact when compared to timeless analytical derivations. Hence, when statistical theory is available and analytical solutions possible, *classic formulas may dominate bootstrapped approaches*. As a general guideline as to when or not to bootstrap, first check to see if an analytical solution is available and use it if it is a valid one that is strongly supported theoretically. Second, try the bootstrap. Of course, there is never any harm in trying out both approaches and comparing results.

When bootstrapping, the question that also naturally arises regards how many replicates are sufficient to get a reasonable estimate of the distribution of the bootstrapped statistic. Though there is some debate on this, a general recommendation is in the neighborhood of  $B = 50$ , and rarely greater than 200 (Rizzo, 2019). Bootstrapping also allows for replacement, and hence one bootstrapped sample might contain the same sampling element as another bootstrapped sample. The

probability that a data point is not included in a sample is equal to  $1 - (1/n)$  which implies that the probability that the given data point is not included in  $n$  samples is equal to  $(1 - 1/n)^n$ , which is a well-known approximation to  $1/e$  in the limit as  $n \rightarrow \infty$  (Aggarwal, 2015).

Bootstrapping in Python can be easily performed using *SciPy*. Bootstrapping in R can be performed several ways. One way is by using the *bootstrap* package (Original et al., 2019).

### 2.28.3 Variations and Extensions of the Bootstrap

There are numerous variations on the bootstrap. Among these include the *leave-one-out bootstrap*, where the accuracy of an instance is computed using all the data except that instance (Aggarwal, 2015). Kleiner et al. (2014) proposed a sub-sampling method called *bag of little bootstraps* (BLB) to help deal with the massive data that may be too large for computer memory or take too long to analyze efficiently. In the BLB approach, one draws a number of subsamples of a given size on each sample from the original collected data set. For each of the subsets, a number of bootstrapped samples are chosen and we compute statistical methods on those subsets (e.g., computation of the mean). These are thus, the point estimates. These measures are then combined by averaging to yield the overall point estimates and quality measures.

The *fractional bootstrap*, also known as the *Bayesian bootstrap*, draws random weights and instead of applying such to resampling units, applies them to the actual data. For details and implementation in R, see the *fwb* package (Greifer, 2023).

## 2.29 Elements of Classic Null Hypothesis Significance Testing

For better or for worse, much of classical statistics is based on the method of *null hypothesis significance testing*, or “NHST” for short, which provides the basis for rejecting a hypothesis if the data are unlikely to have occurred under the presumed model. This yields the classic “*p*-value” from which a scientist decides “yay” or “nay” whether to retain or discard the null. The null hypothesis, contrary to misconception, is not necessarily the hypothesis of a null result or zero effect, nor is it necessarily the hypothesis of equality in a comparison scenario of two populations. R.A. Fisher, the inventor of ANOVA, coined the null hypothesis to be the hypothesis to be *nullified* if observed data contradict it. More specifically, if the probability of an observed statistic is relatively small in light of the null, the null is rejected and a *statistical alternative* hypothesis is inferred. The designation of “statistical” in *statistical alternative* is important. A statistical alternative hypothesis is a statement of “not the null,” and is *not* a statement confirming the researcher’s substantive alternative hypothesis. Classical null hypothesis significance testing, or really any *statistical* theory, does not have the ability to confirm a research hypothesis. Misunderstanding this point can easily lead to misunderstanding the results of a significance test.

As an example to illustrate the point, suppose we posit a null hypothesis that the mean IQ of a population is equal to 100. That is,

$$H_0 : \mu = 100$$

The statistical alternative hypothesis is that the mean is not equal to 100; that is,

$$H_1 : \mu \neq 100$$

But why would it not be equal to 100? Perhaps because the investigator treated the group with a magic pill that would increase their IQ. In this context, that is the *research alternative hypothesis*.



Note that *rejection of the null and inference of the statistical alternative hypothesis is not equivalent to inference of the research alternative*. The null hypothesis does not know “why” it is being rejected; it is simply being rejected based on the low probability of the test statistic. Hence, a rejection of the null could “happen” under a lot of conditions, especially if the study or experiment were done poorly. If the null is rejected under a properly done experiment, then inference of the statistical alternative naturally leads to a strong inference of the research alternative, but that “leap” is not automatic, and making it so can easily lead to inference of an incorrect research alternative. For instance, under a poorly performed study, perhaps it wasn’t the magic pill at all that was the effective agent; perhaps the increase in IQ was due to increased sunlight due to a change in seasons or a change in diet among participants that simulated an increase in IQ because of greater cognitive processing power.

To emphasize the abstraction that exists in the classical null hypothesis significance test, we can express the parameter value (or vector, in the case of multivariate models) as belonging to a particular parameter space under the null,  $\Theta_0$ , from which the parameter value is a member of that set,

$$H_0 : \theta \in \Theta_0 \quad (2.60)$$

against an alternative hypothesis that the parameter value is a member of an alternative set,

$$H_1 : \theta \in \Theta_0^c \quad (2.61)$$

where  $\Theta_0^c$  is the complement to  $\Theta_0$  and whose membership is that of *not the null*. Why the abstraction? To make things unnecessarily abstract and mathematical? No! This symbolism demonstrates the most *general framework* of the null hypothesis significance test and emphasizes that the complement to the null hypothesis *is* an abstraction. It is not “concrete.” Otherwise said, rejection of the null hypothesis does not necessarily mean anything other than inferring the complement to the null in terms of an abstract parameter space! Allowing this idea to sink in will prevent the reader from making egregious scientific inferences when rejecting a null hypothesis that are habitually made even by professionals in various fields. A rejection of the null does not imply the research alternative; it merely indicates the complement to the null *statistically* (not *substantively*).

Classical significance levels used in most scientific investigations include 0.05 and 0.01, though there is absolutely no statistical or philosophical grounds for using these levels. Fisher himself simply denoted the 0.05 level as “usual and convenient” in the establishment of evidence to contradict the null. Still, there is a long history of misunderstanding what 0.05 actually means, and often researchers incorrectly treat 0.05 as a “yes” vs. “no” decision on the presence of an *effect* in the experiment, which is a serious misunderstanding. Not only does 0.05 not necessarily indicate the presence or absence of a meaningful effect, but there is simply nothing “special” about 0.05. As many have noted before, surely God loves the 0.06 level as much as the 0.05. Determining an appropriate significance level should be done carefully with regard to the risk tolerances of making type I and type II errors depending on the given context of the situation. Once more, *context matters*!

A rejection of the null does not necessarily imply that the null is false. Of course, logically, the null can be said to be false even before collecting the data. Do we really expect population means to ever be *exactly* equal? Hardly ever! Even with a rejection, we do not know if the null is false; we are simply using the available sample data to *hedge our bet*. Science is a gamble! A false rejection of the null is called a *type I error* and is indicated by the level at which we set our statistical test, that of  $\alpha$ . A *type II error* is the error committed when we fail to reject a null hypothesis that is, in actuality, false. The probability of such an error is indicated by  $\beta$ .

*Statistical power* is equal to the balance of the type II error rate, computed as  $1 - \beta$ . Power is the probability of rejecting the null hypothesis given that it is false. Ideally, power for a study or experiment should be at least .80 or higher, where more power than not is usually a good thing if our goal is to reject a null.

### 2.29.1 *One-Tailed vs. Two-Tailed Tests*

A *one-tailed test* is when we commit all of our rejection region on one tail of the distribution and none on the opposite tail. A *two-tailed test* is when there are rejection regions on both ends of the sampling distribution. It is worth emphasizing that while typically two-tailed tests constitute an even “splitting up” of the alpha region, there is no requirement of such equality when performing a two-tailed test, so long as one adjusts their test appropriately. That is, one can allot 4% of the rejection region in one tail and 1% in the other if that is what substantive interest or theory indicates. In practice, however, for a 5% test, the researcher will usually allocate 2.5% to each tail. Since the event of the statistic falling in one tail or the other constitutes a sum of two *mutually exclusive events*, these areas can be added (i.e.,  $0.025 + 0.025$ ) to produce the overall type I error rate.

### 2.29.2 *Effect Size*

A *p-value* essentially tells us the probability of the data given the null hypothesis. Because the standard error of a statistic is influenced heavily by sample size, it is entirely possible to reject a null hypothesis simply as a function of sample size, where the “effect” is extremely small. This can be very easily illustrated in the case of a one-sample *z*:

$$z = \frac{\bar{y} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Note that as  $n \rightarrow \infty$ ,  $z \rightarrow \infty$ . More formally,

$$\lim_{n \rightarrow \infty} z = \infty$$

which reads that in the limit as  $n \rightarrow \infty$ ,  $z$  will get increasingly larger and likewise go to  $\infty$ . Since its inception, it has long been argued that *p-values* do not tell the researcher or scientist what they actually want to know. What they want to know is a number characterizing the mean difference or the extent to which the independent variable “explains” variance in the dependent variable. *The p-value will not tell you this information, no matter how small it is! Stop reporting  $p < 0.05$ , thinking that it alone justifies a scientific finding! Only effect sizes will yield explained variance and a true characterization of a scientific effect.*

### 2.29.3 *Cohen’s d Measure of Effect Size*

Cohen’s *d* is a popular measure of effect size in either the one-sample case or two-sample case. Below is *d* for the one-sample case, where  $\bar{y}$  is the sample mean,  $\mu_0$  is the population mean under the null hypothesis, and  $\sigma$  is the population standard deviation,

$$d = \left| \frac{\bar{y} - \mu_0}{\sigma} \right| \quad (2.62)$$

Note that  $d$  is effectively a ratio of quantities, the ratio of one deviation,  $\bar{y} - \mu_0$  to another,  $\sigma$ . That is, the ratio answers the question “How large is the observed deviation relative to the *standard* deviation?” If the deviations are approximately the same, then  $d \simeq 1$ . Unlike  $R^2$ , which is bound by 1,  $d$  does not have such an upper limit.

Note that  $d$  is not a function of sample size, as is the  $p$ -value. True, sample size  $n$  does appear in the denominator of  $d$  in the standard deviation,

$$\sigma = \sqrt{\frac{(y_i - \mu)^2}{n}}$$

however, note that the standard deviation is not a function of  $n$  as is the standard error in the one-sample  $z$ -test for a mean,  $\sigma / \sqrt{n}$ . That is,  $\sigma$  will not necessarily increase or decrease simply as a function of  $n$  as is true of the standard error. Hence, though sample size appears in the denominator, it serves a different function. This is the precise difference between a  $p$ -value and an effect size: *p-values are a (partial) function of sample size; effect sizes are (generally) not.*

There are a whole host of effect sizes that can be computed in statistics. Sometimes the effect has to be gleaned from the research paper as the authors of the paper will not outwardly identify their effect size measure or mistakenly believe that the effect size is contained in the  $p$ -value. For example, they may report the odds of an effect occurring, which in itself serves as a useful effect size estimate, but will not identify the odds specifically as the effect size. It is up to the reader in many cases to understand and appreciate the nature of an effect size so they can identify it in the paper they are reading (and possibly compute or otherwise estimate it if it is omitted).

Transformations between effect sizes are well documented. For example, if a research paper reports a  $d$  statistic, one can convert from  $d$  to  $r$  via the following transformation:

$$r = \frac{d}{\sqrt{d^2 + 4}} \quad (2.63)$$

The conversion of  $d$  into a log odds ratio is also easily obtained:

$$\log \text{ odds ratio} = \frac{d\pi}{\sqrt{3}} \quad (2.64)$$

For example, suppose Cohen’s  $d$  is equal to 0.22. As the reader may verify, this translates into an approximate log odds ratio of 0.40. A wealth of other transformations between effect sizes exists in the literature, including many “conversion” calculators online that aid in translating effect sizes from one form to another.

#### 2.29.4 Are P-values That Evil?

As mentioned, the misconception behind  $p$ -values, historically, is nothing short of shameful. The legacy of misusing such indicators in the establishment of scientific evidence is nothing short of embarrassing, and what is perhaps worst, *it continues on today!* While effect sizes are being

increasingly emphasized in research, there are still numerous studies across various fields that report  $p$ -values as a sole indicator in the establishment of scientific evidence, almost as if  $p < 0.05$  is reason enough on its own to somehow establish evidence or success in an experiment or study. As we have emphasized, without the inclusion of an effect size, the reporting of scientific evidence based on  $p$ -values alone is incomplete. *Any good statistics course one takes should dramatically emphasize the distinction between  $p$ -values and effect sizes. From a scientific perspective, it is probably the most important piece of knowledge one can acquire in learning applied statistics.*

Based on this discussion, it may seem as though  $p$ -values are evil and should forever be avoided. Indeed, some have even argued for a ban on  $p$ -values. Meehl (1978) has even accused Fisher of almost purposely “beffudling” researchers into believing in  $p$ -values! However,  $p$ -values are really not that bad and serve a purpose. *Interpreting  $p$ -values incorrectly* is where the real problem lies.

The  $p$ -value, by itself, serves a useful function. It allows for an inferential statement based on a rejection of the null hypothesis. This is not a bad thing. And though it may sound borderline criminal that one can obtain a small  $p$ -value based simply on increasing sample size  $n$ , this too is not a bad thing. An increase in sample size leading to an increasingly smaller  $p$ -value implies that the estimation of the parameter value is improving. That is, the strength of the evidence against the null should increase given the increase in  $n$ . In the limit, as  $n$  increases to infinity, the rejection of the null is guaranteed and assured, assuming at least *some* effect. An estimator is *consistent* if with increasing  $n$ , we obtain a better and more precise estimate of the parameter based on our sequence of estimates. That is, a consistent estimator is one such that

$$\lim_{n \rightarrow \infty} p(|\hat{\theta} - \theta| < \varepsilon) = 1 \quad (2.65)$$

where  $\hat{\theta} - \theta$  is the difference between the estimator,  $\hat{\theta}$ , and the parameter value,  $\theta$ . Consistency of an estimator assures us that as the distance  $\hat{\theta} - \theta$  gets smaller, less than some arbitrarily small number  $\varepsilon$ , the probability of that distance being smaller than  $\varepsilon$  is equal to 1 as  $n \rightarrow \infty$ . As Casella and Berger (2002, p. 468) note, an equivalent statement to the above is

$$\lim_{n \rightarrow \infty} p(|\hat{\theta} - \theta| \geq \varepsilon) = 0 \quad (2.66)$$

which essentially says the same thing; in this case, the probability of any difference being equal or greater than some small magnitude  $\varepsilon$  is equal to 0 as  $n \rightarrow \infty$ .

The  $p$ -value is a reflection of this consistency and hence it getting smaller as  $n$  increases is what the  $p$ -value *should* be doing. What the *user* should not be doing is attributing the decreasing  $p$ -value as increasing support for the research or scientific alternative hypothesis. That is where the problem lies. The movement to ban  $p$ -values from particular journals is akin to banning firearms entirely 100% due to misuse. An argument can be made on both sides of the debate, but the gun ( $p$ -value) itself, as it sits there not harming anyone, is innocent. As with all innocent mathematics and statistics (and guns), *it takes a human being to truly mess it up.*

### 2.29.5 *Absolute vs. Relative Size of Effect Size: Context Matters*

An  $R^2$  effect size of 0.10 is *relatively* small numerically, since the maximum theoretically attainable effect size is 1.0. However, 0.10 may be quite large and impressive relative to the given research area. Effect sizes should be contextualized within the research area to better appreciate how large or small they really are. Cohen (1988) provided “rules of thumb” for effect sizes in social research of 0.2 small, 0.5 medium, and 0.7 or higher “large.” But it should be emphasized that these are nothing more than *arbitrary guidelines* and Cohen may have been better off not

providing them at all, since too often researchers simply compare their obtained effect size to these arbitrary guidelines instead of comparing their obtained effect size to the effect sizes in competing research.

How large is an effect of  $d = 0.7$  or so? That converts to an  $R^2$  of approximately 0.11, which means we're only explaining approximately 11% of the variance in the response variable. Is that "large?" Not in absolute terms; it's very small, given  $R^2$  has a theoretical upper limit of 1.0. But, for many areas of social research (which is the area Cohen was referring to when he published his guidelines of 0.2, 0.5, and so on), explaining 11% is usually pretty decent as much social research will typically explain much less. Hence, in that *context*, 0.7 or so is considered relatively large. *But only in that context*. If the effect size of raising or lowering the flaps on the response variable of flight on an airplane were that low, we'd all be taking the bus. In that context, explaining 11% of the variance is comical, negligibly small, and outright dangerous. We could easily devise a social study where 11% would be negligible as well, yet we could also imagine a social study where 11% is whoppingly large and impressive. Imagine if the way people smiled or did not smile explained 11% of the variance in their lifespan; that would probably be considered ground-breaking and extremely large! Yet the same effect size explaining flap angles and flight has the plane crashing into the ground way too often.

Hence, effect size without context is very difficult to interpret other than in a purely absolute sense (e.g., 11% of the variance explained is better than 0% but still a long way to maximum variance, which is 100%). In an *absolute* sense, regardless of the research area, 11% is small. Or, we could say, "Relative to the range of theoretically possible effect sizes, 11% is small, because the theoretical maximum is 100%. We're only explaining approximately one tenth of what is theoretically possible to explain in terms of variance." But when we step away from the abstract and say things like "This drug explains variance in blood pressure by 30%," that may have tremendous *practical value* and save many lives, especially if given most other drugs explain only up to 5%. In other words, the drug is *clinically relevant*. It is *clinically significant*. In this way, that drug's effect size is extremely impressive and large, because we're comparing it to the research area and also thinking about its practical implications. Summer tires vs. winter tires probably explains a reasonable amount of variance in traffic deaths in winter, let's say 5%. That 5% might represent thousands and thousands of lives saved and may be the most significant thing we can do to curb traffic deaths in winter because all other "treatments" (like brighter headlight bulbs or other alternatives) explain only 0.3%, for instance. Hence, the effect size of 5% is very big in the particular area. Are we going to compare that to Cohen's guidelines? No, because his guidelines were specific to a narrow area of research and may at best be arbitrary in relation to other areas.

### 2.29.6 Comparability of Effect Sizes Across Studies

The question often arises as to whether effect sizes are comparable across experiments and studies. The simple answer is that some effect sizes, such as Cohen's  $d$ , are comparable, but with many caveats. First, the comparison will rarely if ever be perfect, as the standard deviation for each study can differ from study to study, as well as what measuring instruments were used, the reliability of those measures, and so on. Cohen's  $d$  is the standardized mean difference within a given study, but *across studies those differences are not necessarily standardized*. This is the key point.

For example, suppose that  $d = 0.5$  in one study vs.  $d = 0.6$  in another. We are choosing these effect sizes to be relatively close on purpose. The second study has the higher effect size, but are we prepared to say the second study has truly found a larger effect? Perhaps the effect size in the one study is larger or smaller simply as a function of variability and not mean difference. For this

reason and others, some (e.g., Baguley, 2009) even suggest reporting *simple effect sizes* without standardization. A simple effect size does not incorporate the standard deviation and features only the mean difference. The problem is that even a few outlying cases can push the variability up so that all of a sudden the mean difference in the numerator does not look quite as impressive (since we are dividing by an increasingly large denominator).

For the above reasons, for most comparisons of effect sizes, it may be best to simply report general *ranges* of effect sizes and focus less on *specific* effect sizes, especially if the material under study is also expected to exhibit much measurement error. For example, if we were working with pristine measures that are reliable and repeatable to multiple decimal places (e.g., speed of an engine or angle of descent of an airplane), then a difference of 0.6 to 0.5 would be more meaningful than with say, a social measure that may be more prone to measurement error and error due simply to the way the data were sampled. Of course, if you suspect that the multiple issues that could affect the size of effect are more or less *constant* across studies, then perhaps you could conclude *0.6 is greater than 0.5 and all the factors which may influence this difference we assume are randomly distributed or more or less equivalent across the studies*, which allows for a more blanket and general statement. However, barring such equivalency, there may be good reasons why one study reported the 0.6 vs. 0.5 even if in reality both effect sizes are in fact, and for all purposes, virtually *equivalent*.

Cohen's  $d$ , or any other effect size, is a function of the specific data that went into it, so in that sense it is *unique* to the data set. If a second data set exhibits the same (or very similar) numerical properties as the first data set, then  $d$  values are much more comparable. Differences in scaling can also mean tremendously different effect sizes to the point where we may be not even sure of what is being compared when we compare  $d$  values. However, if a first study had an effect size of 3 and another 0.5, one might be, in general, fairly confident that the first study exhibits more of an effect than the second *if both studies were similar in many characteristics, including scaling*. As that difference gets smaller and smaller, the confidence in making such a conclusion decreases. If a researcher reports 0.51 vs. 0.52 in Cohen's  $d$  values and concludes, "Therefore, the second study (0.52) found a larger effect," yes, numerically it did, but each effect size computation is but an estimate of the true underlying effect and is riddled with all the measurement errors and random influences that went into the computation, not the least of which is the way the scale is set for the variables under study. A Likert scale with a range of 1 to 7 for responses is likely to elicit a much different distribution than a Likert scale of 1 to 3, even though in both cases the same variable may be being measured. This issue relates to the *operationalization* of the independent or predictor variables, which we now briefly discuss.

### 2.29.7 Operationalizing Predictors

How one chooses to operationalize the independent or predictor variable(s) can have a significant influence on whether an effect is found in statistical modeling. For example, consider  $\eta^2$  from ANOVA. Suppose that  $\eta^2 = 0.80$  in one study vs.  $\eta^2 = 0.70$  in another. Does such imply that the first study has a truly larger effect size? Without knowing any more details of each study, it is entirely possible that  $\eta^2 = 0.80$  has resulted more from the way the independent variable was operationalized than anything else. The following principle should be heeded:

*Where each experiment or study "comes from" is just as important as the numerical measure one is choosing to contrast and compare studies because that numerical measure (e.g., Cohen's  $d$ ,  $\eta^2$ , etc.) is a function of all those underlying influences.*

The above is key. When one traces the roots of the equation for  $d$  or  $\eta^2$ , etc. one gets into the very numbers and characteristics of the data. Effect sizes are not simply *summary statistics*; they are general *reports* of the *data*. Just as comparing the sample mean from one study to another depends on what went into each sample mean, so it is also the case with effect sizes. To emphasize the point, consider the following statement: “The mean grade in one class is a bit higher than another.” What does that tell us? It tells us that *the mean grade in one class is a bit higher than in the other!* To go further into analysis would require us to examine what went into each mean, and so on, which is a much deeper investigation such that in some cases could even reverse the direction of the effect. The takeaway point here is that summary statistics never tell the full story. The deeper story is told in the actual data points, and *the even deeper story is told in how the data were obtained and measured*. “The student’s GPA is 4.0” means nothing unless we know more about what courses were taken, how courses were graded, difficulty of each course, and so on. Data is *experience*, and unless you know the experience intimately, summary statistics can only provide crude approximations.

### 2.30 Central Limit Theorem

There are many situations where theoretical results are facilitated by assuming a normal sampling distribution. For example, when we obtain a sample mean and wish to compare it to the sampling distribution of the mean, we need to assume the sampling distribution takes on a normal (or at minimum, *approximately* normal) shape. If the original population distribution from which we have sampled is known to be normal, then it stands that the sampling distribution will likewise be normal. However, in most cases we do not know the shape of the original population distribution and hence we need a way nonetheless to assure the normality of the sampling distribution. The *central limit theorem*, or CLT, allows us to know the shape of the sampling distribution, whatever the shape of the original parent population.

The *univariate* central limit theorem of the mean holds that if we let  $x_1, x_2, \dots, x_n$  be independent observations from any population with mean  $\mu$  and finite variance  $\sigma^2$ , then  $\sqrt{n}(\bar{x} - \mu) / \sigma$  has an approximate normal distribution,  $N(\mu, \sigma^2)$  for large sample sizes. In the limit, as sample size increases, the sampling distribution of the mean takes on an increasingly normal shape.

Likewise, we can state the *multivariate* central limit theorem more formally as if we let  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$  be independent observations from any population with mean  $\boldsymbol{\mu}$  and finite covariance matrix  $\boldsymbol{\Sigma}$ , then  $\sqrt{n}(\bar{\mathbf{X}} - \boldsymbol{\mu})$  has an approximate normal distribution,  $N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  for large sample sizes (Johnson & Wichern, 2007). As in the univariate case, if the parent population is normal, the multivariate distribution of the sample will be normal. The extent to which the parent population is normal speeds up the process for normality of the sampling distribution as sample size increases. For a more formal definition and characterization of the central limit theorem, see Casella and Berger (2002, p. 236). Note that in general, the central limit theorem is about the behavior of *random variables*, not specifically the mean  $\mu$  or mean vector  $\boldsymbol{\mu}$ . In this sense, there are many different “versions” of the central limit theorem. Most students come across the CLT in the context of the behavior of the *mean*, but such should not imply the CLT is *uniquely* about the mean.

### 2.31 Covariance and Correlation

A mathematical *relation*, or simply a *relationship* between two variables, is a subset of the Cartesian product (Figure 2.8). That is, if any pairs are not represented in a given correspondence between two (or more) variables, this, by definition, denotes a relationship between the two (or more) variables.

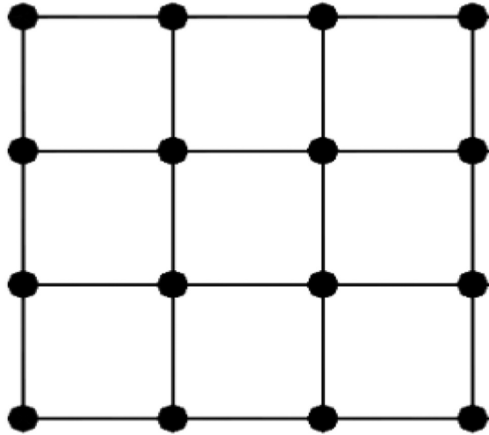


Figure 2.8 Example of a  $4 \times 4$  Cartesian product. Any subset of points would denote a mathematical relationship.

Take a moment and note how general and nonspecific the idea of a relationship is between variables. *If we delete even one pair out of the totality of pairs, we have illustrated a relationship!* Why are we emphasizing this matter? For the reason that the next time someone says to you, “*There is a relationship between  $x$  and  $y$  in my data,*” it should be answered with “*Well, of course there is, but more importantly, is the relationship in any way interesting?*” The point we are making here is that between any two or more variables, there is virtually *always* a relationship that exists among them in the sample and in the population. How could there not be? Do you really think all pairs in any Cartesian plot could ever be represented for any empirical pairing of  $x$  and  $y$ ? Of course not! Hence, *the naming of a relationship between two or more variables is usually an obvious and uninteresting statement.* Now, usually, professing a relationship means more. Let’s consider what it *could* mean now.

A relationship between variables can take on many forms. For two continuous variables, a literal infinite number of curves can be drawn. Said differently, a literal infinite number of *pair deletions* could occur in the Cartesian plot such as to generate a relationship. Often, of most interest (and convenience) is that of *linear* relationships. Both the *covariance* and *correlation* are measures of the *linear association or relationship* between two variables.

The covariance between two random variables  $x$  and  $y$  is given by

$$\text{Cov}(x, y) = E[(x_i - \mu_x)(y_i - \mu_y)] \quad (2.67)$$

which we can also express as the integration of areas for the continuous case,

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x_i - \mu_x)(y_i - \mu_y) f(x_i, y_i) dx dy \quad (2.68)$$

and



$$\sum_{x_i=1}^n \sum_{y_i=1}^n (x_i - \mu_x)(y_i - \mu_y) p(x_i, y_i) \quad (2.69)$$

for discrete random variables.

The sample covariance between variables  $x$  and  $y$  is given by

$$s_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1} \quad (2.70)$$

or, less commonly, can also be written as

$$s_{xy} = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{n-1}$$

We can see that the covariance, of course, requires variation in both  $x$  and  $y$  since if either  $(x_i - \bar{x})$  or  $(y_i - \bar{y})$  are equal to 0, then

$$\begin{aligned} s_{xy} &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1} \\ &= \frac{0}{n-1} = 0 \end{aligned}$$

That is, if you have no variation, you cannot quantitatively assess relationship. Relating age in years to income level won't work if everyone in your sample has the same income. Statistical relationships do not work well with *constants*. You need *variability* in *both* variables.

The covariance is in natural or original units of both  $x$  and  $y$ , and hence it is possible that a small or large covariance is mostly reflective of the scale on which  $x$  and  $y$  are measured. That is, a scale on which there is little variation on either  $x$  or  $y$  or both might reflect a data set with small covariance even if the degree of linear relationship is relatively strong. For this reason, it is useful to standardize the covariance. We do this by dividing the covariance by the product of standard deviations on  $x$  and  $y$ . Doing so gives us the *Pearson product-moment correlation coefficient*,  $r_{xy}$ ,

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{s_{x_i}^2 \cdot s_{y_i}^2}} = \frac{\text{cov}_{xy}}{\sqrt{s_{x_i}^2 \cdot s_{y_i}^2}} \quad (2.71)$$

which has a theoretical range from  $-1$ , a perfect negative linear relationship, to  $+1$ , a perfect positive linear relationship. A value of  $r_{xy} = 0$  indicates no *linear* relationship between variables. Again, the word “linear” here is emphasized, since it is entirely possible that there still is a relationship between variables, only that it is not linear. For example, the scatterplot in Figure 2.9 will likely yield a correlation close to 0, yet the relationship is still relatively apparent.

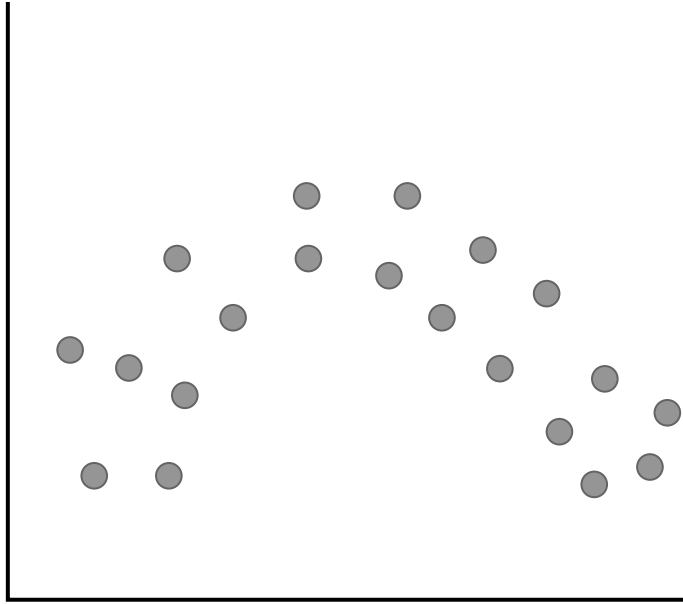


Figure 2.9 A plot of data in which the Pearson Product-Moment correlation is likely near 0, yet there still exists a relationship between  $x$  and  $y$ . The word “relationship” should never be taken or assumed to be necessarily equivalent to or synonymous with the word “linear.”

It is easy to see as well that if either  $s_{x_i}^2$  or  $s_{y_i}^2$  or both equal 0, indicating no variation in variables, then  $r_{xy}$  is undefined. Again, just as was the case for the covariance, the correlation coefficient is a measure of (standardized) co-variation in variables. If there is no variation in one or both variables, the correlation coefficient is not defined. The correlation coefficient  $r_{xy}$  is a *consistent* estimator, that is,  $r_{xy} \rightarrow \rho$  as  $n \rightarrow \infty$ . There are multiple ways to interpret the correlation coefficient. For an excellent discussion, see *Thirteen Ways to Look at the Correlation Coefficient* (Rodgers & Nicewander, 1988).

### 2.31.1 Why Does $r_{xy}$ Have Limits $-1$ to $+1$ ?

The limits of  $-1$  to  $+1$  on  $r_{xy}$  are such as a result of the product of variances,  $s_{x_i}^2 \cdot s_{y_i}^2$  in the denominator, which implies that the numerator  $\text{cov}_{xy}$  can be no larger in absolute magnitude than the denominator  $\sqrt{s_{x_i}^2 \cdot s_{y_i}^2}$ . The denominator in  $r_{xy}$  essentially provides the maximum “variability” that is available to be accounted for in the numerator, and hence the numerator  $\text{cov}_{xy}$  can never be larger, in absolute value, than the denominator. The sign of the covariance will indicate whether  $r_{xy}$  is positive or negative, but the magnitude of  $r_{xy}$  can be no greater than  $|1|$ . A more formal proof of this can be found in Casella and Berger (2002, p. 172).

### 2.31.2 Covariance and Correlation in R and Python

We can quite easily compute covariance and correlation in R via the `cov()` and `cor()` functions. As an example, consider the following two vectors of data:

```
> x <- c(0, 4, 7, 9, 10)
> y <- c(3, 4, 6, 7, 8)
> cov(x, y)
[1] 8.25
> cor(x, y)
[1] 0.9794398
```

We see above that the covariance between the two vectors is 8.25, while the standardized covariance, or *correlation*, is equal to 0.979. Note that the covariance is not that large, yet the correlation has a very large magnitude. This is due to the standardization of the covariance. *Never assume that simply because the covariance is relatively small (or large) that the correlation will likewise be small (or large). They are distinct numerical measures. A large covariance does not necessarily imply an equally large correlation.*

In Python, we can compute the correlation very easily, where we first produce data vectors on *number of coins in your pocket* and *money in your bank account* using `scipy.stats`:

```
coins = [0, 3, 4, 5, 5, 6, 7, 10, 11, 12, 15, 20, 25, 28, 30, 35]
money = [20000, 30000, 40000, 45000, 55000, 60000, 80000, 100000, 130000,
150000, 200000, 230000, 250000, 300000, 350000, 400000]
import scipy.stats as stats
corr, _ = stats.pearsonr (coins, money)
corr
0.9923
```

The Pearson correlation between coins and money is estimated to be 0.9923.

### 2.31.3 Correlating Linear Combinations

Just as we can covary or correlate random variables, we can also covary or correlate *linear combinations of random variables*. For example, consider the following two linear combinations,

$$z = \mathbf{a}'\mathbf{x} = a_1x_1 + a_2x_2 + \cdots + a_px_p$$

and

$$w = \mathbf{b}'\mathbf{y} = b_1y_1 + b_2y_2 + \cdots + b_py_p$$

The covariance between these linear combinations is given by

$$s_{zw} = \frac{\sum_{i=1}^n (z_i - \bar{z})(w_i - \bar{w})}{n-1} \quad (2.72)$$

and likewise, the sample correlation between  $z$  and  $w$  is

$$r_{zw} = \frac{s_{zw}}{\sqrt{s_z^2 s_w^2}} \quad (2.73)$$

Why would we want to covary or correlate linear combinations? For the same reason why we would want to correlate any two random variables. A linear combination is just a *composite* of variables. What is a single variable? It is also a composite variable, *only a very simple composite made up of only itself*. To illustrate, consider once more the linear combination for  $w$  :

$$w = \mathbf{b}'\mathbf{y} = b_1y_1 + b_2y_2 + \cdots + b_py_p$$

Now, simplify  $w$  so that it becomes only a function of  $y_1$ , that is,  $w = \mathbf{b}'\mathbf{y} = b_1y_1$ . If  $b_1 = 1$ , then  $b_1y_1 = y_1$  and hence,  $w = y_1$ . Thus, the linear combination is simply composed of  $y_1$ . That is, the motivation for correlating linear combinations is the same as that for single variables, only that linear combinations are usually more complex than single variables; they are “made up” of more “stuff” (i.e., variables) than a single random variable. However, the concept is the same. Many multivariate methods, in one way or another, correlate linear combinations, though it may not appear this way on the surface. As an example, analysis of variance and discriminant analysis do so, which are all versions of an underlying multivariate correlational methodology called *canonical correlation*. We discuss canonical correlation more thoroughly later in the book.

#### 2.31.4 Covariance and Correlation Matrices

Many statistical procedures, such as principal components analysis or factor analysis, do not operate on individual variables. Rather, they operate on *matrices* of variables. That is, the object of analysis is not a variable but instead a *matrix*. As an example, principal components analysis performs (linear) dimension reduction on the covariance matrix directly. The covariance matrix for the population is given by

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \cdot & \cdot & & \cdot \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix} \quad (2.74)$$

where  $\sigma_{12}$ , for instance, is the covariance between variables 1 and 2, and  $\sigma_{11}$  is the covariance of a variable with itself, which is the variance, since

$$s_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1} \rightarrow \frac{\sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})}{n-1} \rightarrow \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1} = s_x^2$$

Of course, as usual in statistics, we rarely if ever have access to the population covariance matrix, and hence instead of  $\Sigma$ , we usually only have  $\mathbf{S}$ , the sample covariance matrix, which, when corrected by  $n/(n-1)$ , is an unbiased estimator of  $\Sigma$ . That is,  $E\left(\frac{n}{n-1}\mathbf{S}\right) = \Sigma$ .

The sample correlation matrix is the standardized covariance matrix, given by

$$\mathbf{R} = \begin{pmatrix} 1 & r_{12} & \cdots & r_{1p} \\ r_{21} & 1 & \cdots & r_{2p} \\ \cdot & \cdot & & \cdot \\ r_{p1} & r_{p2} & \cdots & 1 \end{pmatrix} \quad (2.75)$$

Both the covariance and correlation matrices are *symmetric* matrices, which, as we will see in our review of linear algebra, implies that for an arbitrary matrix  $\mathbf{A}$ ,

$$\mathbf{A} = \mathbf{A}^T$$

That is, the original matrix is equal to that of its transpose, or, otherwise said, if we exchange rows for columns and columns for rows, we will obtain the same matrix. Why is this fact important? It is important because there are special properties associated with symmetric matrices, including the fact that *eigenvectors are guaranteed to be pairwise orthogonal only for symmetric matrices whose eigenvectors correspond to distinct eigenvalues*, which will have significant implications for methods such as principal components. Symmetric matrices are also necessarily square matrices (i.e., same number of rows as columns), which allows for eigen decomposition to be performed on such matrices, whereas for nonsquare and asymmetrical matrices the more general SVD (*singular value decomposition*) is typically preferred.

### 2.32 Z-Scores and Z-Tests

Recall the definition of a *z-score*, or *standardized score*,

$$z = \frac{y_i - \mu}{\sigma}$$

A *z-score* is actually a comparison of two deviations, one in the numerator,  $y_i - \mu$ , to one in the denominator,  $\sigma$ , the standard deviation. Thus, one can interpret the *z-score* as a comparison of how large the current raw score deviation is to what is “standard” in the given distribution we are working with. A *z-score* of 0 arises only if  $y_i - \mu = 0$ , which means the raw score does not deviate at all from the mean. Should the distribution have no dispersion, then  $\sigma$  will equal 0 and the *z-score* will be *undefined*. It is also worth noting that the numerator represents a *centering* of  $y_i$ . That is, whenever we subtract the mean, we are centering the given observation. Hence, the *z-score* can also be defined as the *standardized centered observation*. Centering plays a significant role in statistical modeling and will be revisited when we survey regression analysis and other methods in the book. Many times, we wish to center predictors by subtracting the mean so that the predictor value at zero is more meaningful. Centering often makes the intercept of a model more interpretable.

We can easily demonstrate centering and computing *z-scores* in R. Consider the following distribution of raw scores:

```
> x <- c(0, 4, 8, 9, 10)
> mean(x)
[1] 6.2
```

We now center the distribution by subtracting the mean,

```
> x.center <- x - mean(x)
> x.center
[1] -6.2 -2.2 1.8 2.8 3.8
> mean(x.center)
[1] -1.775489e-16
```

Note that the mean of the new centered distribution is equal to 0 (within rounding error). To produce  $z$ -scores, we can then divide by the standard deviation, while also confirming that the mean of the new  $z$ -scores is equal to 0 and the standard deviation is equal to 1:

```
> z = (x - mean(x)) / sd(x)
> z
[1] -1.4949527 -0.5304671 0.4340185 0.6751400 0.9162614
> mean(z)
[1] -2.218278e-17
> sd(z)
[1] 1
```

$Z$ -scores form the basis for the *standard normal distribution* which has a mean equal to 0 and a standard deviation equal to 1. Pay notice to the word “standard” here. Most empirical distributions that are normal (or approximately so) do not have a mean of 0 and standard deviation of 1.

### 2.32.1 *Z-Tests and T-Tests for the Mean*

Recall the one-sample  $z$ -test for the mean,

$$z_M = \frac{\bar{y} - E(\bar{y})}{\frac{\sigma}{\sqrt{n}}} = \frac{\bar{y} - \mu_0}{\frac{\sigma}{\sqrt{n}}} \quad (2.76)$$

The reader will recall from introductory courses that we typically do not know the standard error of the mean,  $\frac{\sigma}{\sqrt{n}}$ , and instead require an estimate of it,

$$\hat{\sigma}_M = \frac{s}{\sqrt{n}} \quad (2.77)$$

When we use  $s$  as an estimate of  $\sigma$ , the resulting statistic is no longer a  $z$  statistic; it is instead distributed as  $t$ , which gives rise to the  $t$ -test for one sample,

$$t = \frac{\bar{y} - E(\bar{y})}{\frac{s}{\sqrt{n}}} \quad (2.78)$$

It is a fact of mathematical statistics that the  $t$ -distribution converges to the  $z$ -distribution as degrees of freedom increase without bound. This is the case because the variance of  $t$  is defined as

$$\sigma_t^2 = v / (v - 2) \quad (2.79)$$

and so as  $v \rightarrow \infty$ , the above ratio goes to one, that is,

$$\lim_{v \rightarrow \infty} \left( \frac{v}{v - 2} \right) = 1$$

For two samples, the  $z$ -test for means is given by

$$z_M = \frac{E(\bar{y}_1) - E(\bar{y}_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \quad (2.80)$$

The justification for summing the variances in the standard error is due to the *variance-sum law* that holds that if  $y_1$  and  $y_2$  are normally distributed and independent random variables with means  $\mu_1$  and  $\mu_2$  and variances  $\sigma_1^2$  and  $\sigma_2^2$ , the difference between the random variables has a mean equal to  $\mu_1 - \mu_2$  with variance  $\sigma_1^2 + \sigma_2^2$ . That is, the standard error of the difference between means is equal to,

$$\sigma_{\bar{y}_1 - \bar{y}_2} = \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} \quad (2.81)$$

What is more, due to the central limit theorem, the sampling distribution of  $\bar{y}_1 - \bar{y}_2$  is approximately normal for increasingly large samples and is theoretically normal for  $n \rightarrow \infty$ .

As for the one-sample situation, we rarely, if ever, know the values of  $\sigma_1^2$  and  $\sigma_2^2$  and hence estimate them with  $s_1^2$  and  $s_2^2$ ,

$$t = \frac{E(\bar{y}_1) - E(\bar{y}_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{\mu_1 - \mu_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad (2.82)$$

For unequal sample sizes (*not unequal variances*), we pool variances, which means we will require an estimated standard error not of the mean, but of the *difference between means*,

$$\begin{aligned} \hat{\sigma}_{diff} &= \sqrt{\hat{\sigma}_{pooled}^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)} \\ &= \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{n_1 + n_2}{n_1 n_2} \right)} \end{aligned} \quad (2.83)$$

The classic two-sided null and alternative hypotheses for the  $t$ -test is

$$H_0 : \mu_1 = \mu_2 \quad H_1 : \mu_1 \neq \mu_2$$

With the pooled variance estimator, the  $t$ -test becomes

$$t = \frac{E(\bar{y}_1) - E(\bar{y}_2)}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{n_1 + n_2}{n_1 n_2} \right)}} \quad (2.84)$$

evaluated on  $(n_1 - 1) + (n_2 - 1)$  degrees of freedom. It should be noted that sometimes the numerator of these difference tests are stated with an assumed value  $D_0$  under  $H_0$  for the difference,  $E(\bar{y}_1) - E(\bar{y}_2) - D_0$ . In theory at least, this allows the researcher to specify a value for  $D_0$  other

than zero; however, zero will typically be taken as the default value unless there is good substantive reason to set it unequal to zero.

### 2.33 Unequal Variances: Welch-Satterthwaite Approximation

Recall that the two-sample  $t$ -test assumes equal population variances, that is,  $\sigma_1^2 = \sigma_2^2$ . If there is evidence to suggest that variances are unequal, then an approximation can be used. The approximation adjusts for the degrees of freedom of the test where instead of degrees of freedom equaling  $(n_1 - 1) + (n_2 - 1)$ , they are computed as

$$df = \frac{(n_1 - 1)(n_2 - 1)}{(1 - c)^2(n_1 - 1) + c^2(n_2 - 1)} \quad (2.85)$$

where,

$$c = \frac{\frac{s_1^2}{n_1}}{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Likewise, approximate confidence intervals may be computed via

$$(\bar{y}_1 - \bar{y}_2) \pm t_{\alpha/2} \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \quad (2.86)$$

### 2.34 Paired Data

When working with paired data, we require making inferences about  $\mu_1 - \mu_2$  just as in the independent-samples  $t$ -test, only that now, observations are not independent; they are *matched* or naturally *paired*. What is interesting about paired data on relatively simple data sets is that the concept of paired data generalizes to much more complex models (e.g., elaborate longitudinal models, blocking designs, and multilevel models with repeated measures). The numerator of the  $t$ -test is the same, since  $E(y_1 - y_1)$  is still  $\mu_1 - \mu_2$ . However, the standard error of the difference in means incorporates the fact that the samples are paired, and is equal to

$$\sigma_{\bar{y}_1 - \bar{y}_2} = \sqrt{\frac{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\rho}{n}} \quad (2.87)$$

where  $\rho$  is the correlation between variables  $y_1$  and  $y_2$ . Note that for  $\rho = 0$ , the numerator under the radical sign is simply  $\sigma_1^2 + \sigma_2^2$  as it was in the unpaired situation, that is,  $\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2(0) = \sigma_1^2 + \sigma_2^2$ . However, for  $\rho \neq 0$ , the standard error for paired data has an expectation *less* than that of independent data, since  $2\sigma_1\sigma_2\rho$  is being subtracted from the sum,  $\sigma_1^2 + \sigma_2^2$ . In practice, the  $t$ -test usually takes the form of computing mean difference  $\bar{d}$  between columns, and like in the independent-samples  $t$ -test, subtracting a null value of  $D_0$ , which yields,

$$t = \frac{\bar{d} - D_0}{s_d / \sqrt{n}} \quad (2.88)$$



The resulting  $t$  is evaluated on  $n-1$  degrees of freedom since there is now only a single column of differences, in effect, it is now a one-sample test. A confidence interval for the mean difference can be computed,

$$\bar{d} \pm t_{\alpha/2} \frac{s_d}{\sqrt{n}} \quad (2.89)$$

where  $s_d$  is the standard deviation of the column of difference scores.

This concludes our brief overview and review of fundamental statistics typically taught in introductory courses. If any of the information in this chapter seemed entirely “new” to you, it is suggested that you review a relatively deep introductory statistics text to review the essentials of statistical inference. Ott and Longnecker (2016) is a very good choice in this regard.

We now move on to the second major pillar of statistics and machine learning, that of linear algebra.

### 2.35 Review Exercises

1. Precisely define the *frequentist* approach to probability.
2. Distinguish between *unconditional* and *conditional* probability.
3. Distinguish between *statistical independence* and *mutual exclusiveness*. Can two events that are independent simultaneously be mutually exclusive? Why or why not?
4. Discuss some of the benefits and dangers of adopting a *Bayesian* approach to updating belief, specifically in reference to *Bayes' theorem*.
5. Discuss what happens to the normal density as  $\sigma^2 \rightarrow 0$ . In the limit, as variance becomes equal to 0, what shape does the normal density take on?
6. Using the *Shapiro-Wilk test* in R, evaluate the null of normality for a random sample of 10,000 observations having mean equal to 0 and standard deviation equal to 1.
7. Compare the univariate density to that of the multivariate density. How are they similar? Different?
8. Using the binomial distribution, compute the probability of obtaining 10 heads out of 10 flips of a presumably fair coin, where “fair” implies a probability of heads equal to 0.5.
9. How can *irreducible error* become *reducible* when measurement is taken into account in the bias-variance trade-off? How can the variance of error be reduced?
10. Clearly distinguish between a *p-value* and an *effect size*.
11. Distinguish between *covariance* and *Pearson's correlation*. Under what condition(s) would they be equivalent?

### 2.36 Linear Algebra and Matrices

Linear algebra is at the heart of virtually all multivariate statistics and machine learning. It is largely the vehicle by which many concepts in these domains are communicated and operationalized. Without linear algebra, though we could still in theory conceptualize what might be meant by a *principal component* or other multivariate objects for example, we would not be able to operationalize or define it narrowly or *precisely*. Linear combinations and other objects from linear algebra, in this case, allow us to do so. They allow for precise and rigorous definition. Through the language of linear algebra, we can define exactly what we mean by a host of statistical objects. Linear algebra is the technical “bedrock” of much of multivariate statistics and machine learning.

We proceed rather briskly and review and recap the most relevant aspects of linear algebra for a general understanding of multivariate statistics and machine learning and preview topics featured in the book. Readers having a prior course in the subject will undoubtedly recognize most, if not all, of this material. Readers new to the area will find the topics informative, yet may wish to supplement this section with one or more of the excellent books that have been written on the subject for a slower and more inclusive gradual introduction. As promised, we do not simply present a bunch of concepts and expect the reader to figure out their relevance later. Instead, we take great care in foreshadowing and previewing where many of these concepts will find their place in later sections in the book.

As an example, the idea of *orthogonality* is meaningful to the specialist in linear algebra in its own right, but for the learner of statistics and machine learning it will be helpful to recognize how the concept is featured in multivariate and machine learning models. We will need to survey the actual models later to fully appreciate its relevance, but the foreshadowing of how it (and other concepts) *will* be used will help make these concepts a bit more “applied” in the sense of appreciating where they belong in the big multivariate/machine learning universe.

### 2.36.1 Vectors

A vector is an array or listing of numbers, where the following is a column vector with elements  $x_1$  through  $x_n$ ,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Because we are working in the real numbers, we denote  $\mathbf{x} \in \mathbb{R}^n$  and the listing of numbers  $x_1$  through  $x_n$  denotes the elements of the vector  $\mathbf{x}$ . In this book, if not indicated specifically, it can be assumed that we are working in the field of *real numbers*, and hence where “ $\in \mathbb{R}^n$ ” is not indicated, it can be generally assumed. None of the work featured in this book features the use of complex numbers ( $\mathbf{x} \in \mathbb{C}^n$ ).

In R, we can create a simple vector by:

```
> x <- c(0, 2, 5, 8, 10)
> x
[1] 0 2 5 8 10
```

Creating the same vector in Python is just as easy:

```
import numpy as np
x = [0, 2, 5, 8, 10]
x
Out[294]: [0, 2, 5, 8, 10]
x_1 = [[0], [2], [5], [8], [10]]
x_1
Out[300]: [[0], [2], [5], [8], [10]]
```

### 2.36.1.1 Vector Spaces and Fields

A *vector space* is generally a space where vectors can be added and lengthened or shortened by scalars, which otherwise implies multiplication and obeys a series of rules featuring these two operations. That is, vector spaces generally describe spaces where the operations of addition and multiplication occur. A *field* is an algebraic structure that satisfies a list of properties such as *commutative* and *associative* laws of addition and many more. Many algebraic structures are special cases of others when certain properties are removed. For the purposes of this book, many of the mathematical operations that underlay methodologies occur in vector spaces because they feature vector and matrix operations.

### 2.36.1.2 Zero, Unit Vectors, and One-Hot Vectors

A vector of all zeroes is called the *zero vector*, while the unit vector has length or magnitude equal to one. Vectors that contain all zeroes except for a single value of “1” are also known as *one-hot vectors*, a term arising from electrical engineering where a circuit is either “on” or “off.” *One-hot encoding* can also mean changing status from text to a number, for instance, “red = 0” which is essentially similar to dummy coding variables.

### 2.36.1.3 Transpose of a Vector

The *transpose* of a column vector is obtained by writing columns as rows, in the case of

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

we have

$$\mathbf{x}^T = [x_1, x_2, x_3, \dots, x_n] \quad (2.90)$$

where  $\mathbf{x}^T$  denotes the *transpose* of  $\mathbf{x}$ . Often (as we do in this book), the transpose is written as  $\mathbf{x}'$ . More formally, the transpose of a *matrix* is obtained by exchanging rows for columns and columns for rows in a matrix. Some properties of transposes include  $(\mathbf{A}^T)^T = \mathbf{A}$ ,  $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ ,  $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ ,  $(c\mathbf{A})^T = c\mathbf{A}^T$ ,  $|\mathbf{A}^T| = |\mathbf{A}|$ , among others.

Transposing vectors (and as we will see, matrices) is a useful operation in the computation of many statistics and quantities used in machine learning. For one, and as we will see, it allows for the multiplication of vectors and matrices. In R, the transpose function is `t()`. In Python, the relevant function is `numpy.transpose()`.

Vectors can be conceived as simply a listing of numbers as above, which is often useful in a computer science context in which we simply would like to identify a listing of numbers. However, a geometrical or visual representation of a vector is often extremely useful, especially in scientific

or physical contexts. As an example, consider a visual representation of the vector  $\mathbf{x}' = [3, 2, 1]$  in Figure 2.10.

Note that the first element of the vector, 3, is plotted along the  $x$ -axis, the second element (2) is plotted along the  $y$ -axis, and the third element, 1, is plotted along the  $z$ -axis. As a result of the vector being defined as a specific location in space defined by its coordinates, in a physical context a vector is often said to have *magnitude* and *direction*.

It is also worth noting that the plotting of the vector in Figure 2.10 generates a new geometrical object, in this case a *cuboid*. Many objects in linear algebra have geometrical counterparts. We will return to this point later in our survey of determinants.

#### 2.36.1.4 Vector Addition and Length

Only vectors of the same dimension can be added or subtracted. We add vectors element by element,

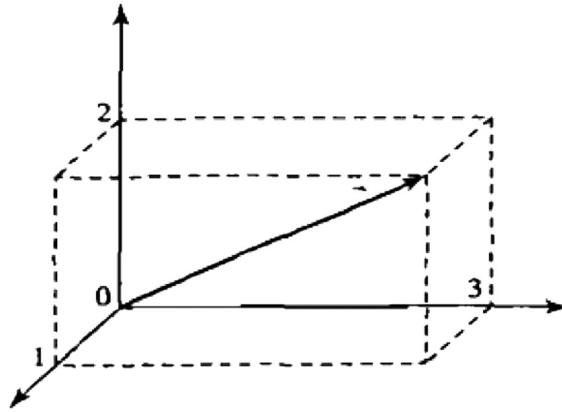


Figure 2.10 Vector with coordinates 3, 2, 1 ( $x, y, z$ )

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix} \quad (2.91)$$

The analogous operation holds for the subtraction of vectors as well. The *length* of a vector is the square root of the sum of squared terms (or “components”),

$$L_x = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad (2.92)$$

For reasons that will become clearer later, in many settings we often desire the length of a vector to be unity, which means the sum of squared elements is equal to 1, that is,

$$L_x = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = 1 \quad (2.93)$$

The length of a vector is also referred to as the *Euclidean norm*, also known as the  $\ell_2$  norm of that vector more generally defined as

$$\begin{aligned} \|x\|_2 &= \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \\ &= \sqrt{\sum_{i=1}^n x_i^2} \\ &= \sqrt{\mathbf{x}^T \mathbf{x}} \end{aligned} \quad (2.94)$$

where  $\mathbf{x}^T \mathbf{x}$  is the dot product (to be reviewed shortly). The  $\ell_1$  norm, on the other hand, instead of summing squares, sums *absolute values*,

$$\|x\|_1 = \sum_{i=1}^n |x_i| \quad (2.95)$$

As we will see later, *eigenvectors* will typically have length set (or *constrained*) to unity in methods such as *principal components analysis*. This is done mostly for convenience so that the variance accounted for by the component cannot grow simply as a function of its length. Setting lengths of vectors equal to unity is common practice in many other statistical methods as well. The important point for now is that what matters most in many of these procedures is not the actual length of the vector, but the *relative distance between the values*. “Normalizing” a vector to unit length does not disrupt these distances. That is, the *ratios* between values remain the same.

We can lengthen or shorten vectors by multiplication by a scalar or constant  $c$ , as in the following where each element of the vector  $x_1, x_2, x_3, \dots, x_n$  is being multiplied by the constant  $c$ :

$$c\mathbf{x} = \begin{bmatrix} cx_1 \\ cx_2 \\ cx_3 \\ \vdots \\ cx_n \end{bmatrix}$$

In Figure 2.11 is an example of multiplying a vector by a constant of 2, which has the effect of lengthening (or “stretching”) the vector. In the same picture we multiply a vector by  $\frac{1}{2}$  (from the origin), forcing it into the opposite direction.

In R, we can easily add two vectors:

```
> x <- c(0, 2, 5, 8, 10)
> y <- c(3, 6, 8, 10, 12)
> x + y
[1] 3 8 13 18 22
```

Multiplying a vector by a constant is also easily accomplished in R. For example, we will assign

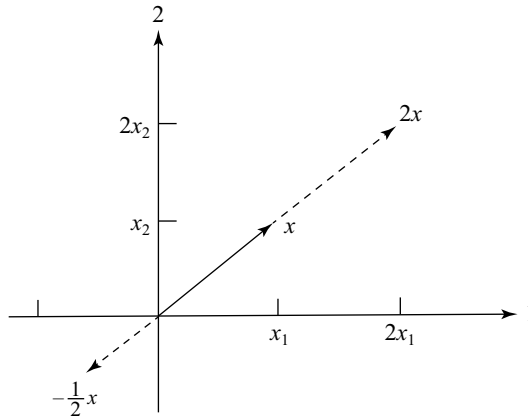


Figure 2.11 Lengthening or “stretching” of a vector by a constant of 2 and multiplying the vector by  $\frac{1}{2}$ .

$z = x + y$  and multiply by a constant  $c = 2$ :

```
> z = x + y
> 2*z
[1] 6 16 26 36 44
```

Vectors can be added in Python as follows:

```
import numpy
a = numpy.array([1, 2, 3])
b = numpy.array([4, 5, 6])
a + b
Out [7]: array([5, 7, 9])
```

We can easily multiply a vector by a constant by:

```
x1 = np.array([1, 2, 3, 4])
x2 = 2 * x1
x2
Out [318]: array([2, 4, 6, 8])
```

### 2.36.1.5 *Eigen Analysis and Decomposition*

The idea of multiplication by a constant may seem trivial in this abstract context but is very significant in multivariate settings because many transformations by a matrix can be equated to the lengthening of a vector. For instance, in *eigen analysis*, as we will see later, the product of a matrix with a vector is equated to simply a lengthening of that vector by multiplication by a constant,

$$\mathbf{Ax} = \lambda \mathbf{x} \quad (2.96)$$

where  $\mathbf{A}$  is some matrix, often a covariance or correlation matrix, post-multiplied by  $\mathbf{x}$ , a vector. The equality states that this product  $\mathbf{Ax}$  is equal to the pre-multiplication of  $\mathbf{x}$  by a constant,  $\lambda$ . The constant  $\lambda$  is referred to as an *eigenvalue* of the transformation and  $\mathbf{x}$  an associated *eigenvector*. As we will see, eigenvectors are unique only up to multiplication by a constant. Many seemingly complicated multivariate methods have eigen analysis at their core and feature this very simple idea of multiplying a vector by a constant, the constant in this case being the eigenvalue of the transformation.

To solve for  $\lambda$  and  $\mathbf{x}$  we can start by rewriting  $\mathbf{Ax} = \lambda \mathbf{x}$  as

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = \mathbf{0}$$

When we set the determinant equal to 0, that is,  $|\mathbf{A} - \lambda \mathbf{I}| = 0$ , we can then find values of  $\lambda$  to then solve for  $\mathbf{x}$ .

As an example, consider the following matrix and the computation of the characteristic polynomial on matrix  $\mathbf{A}$  to obtain eigenvalues:

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

We calculate  $|\mathbf{A} - \lambda \mathbf{I}|$ :

$$|\mathbf{A} - \lambda \mathbf{I}| = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & \lambda \end{bmatrix} = \begin{bmatrix} 3-\lambda & 2 & 1 \\ 1 & 3-\lambda & 2 \\ 1 & 2 & 3-\lambda \end{bmatrix} = -(\lambda-6)(\lambda-2)(\lambda-1)$$

or in unfactored form, simply  $-\lambda^3 + 9\lambda^2 - 20\lambda + 12$ . The corresponding eigenvalues are 6, 2, and 1 because they are the values for which  $|\mathbf{A} - \lambda \mathbf{I}| = 0$  is true. The corresponding eigenvectors, as the reader may verify, are  $\mathbf{v}_1 = (1, 1, 1)$ ,  $\mathbf{v}_2 = (-3, 1, 1)$  and  $\mathbf{v}_3 = (2, -3, 2)$ .

The reader may recall that the fundamental theorem of algebra states that *every polynomial equation of degree  $n$  with complex coefficients has exactly  $n$  roots in the complex numbers*. The proof of the theorem is generally due to Gauss (1799). A number is called *algebraic* if it is the root of a nonzero polynomial with integer (or more generally, *rational*) coefficients. Numbers that are not algebraic are called *transcendental*. For some functions, solving for the roots of an equation is no easy matter. The root of an equation can be *approximated* by techniques such as *Newton's method*. See Tanton (2005) and Hubbard and Hubbard (2002) for more details.

#### 2.36.1.6 Points vs. Vectors

Before we leave the topic of vectors, it may be tempting from the above discussion and visualization of a vector to conclude that a vector is nothing more than a point in dimensional space. This would be a misunderstanding, however. *A vector is not equivalent to a point*. A point has no direction or magnitude, or as Euclid once said, “that which has no breadth” (Tantom, 2005).

The distinction is that a point typically represents a *position* in space, whereas a vector typically represents a *displacement* or *increment*. Points are typically not added or subtracted from one another, but vectors often are. Hubbard and Hubbard (2002, p. 37) give the example of two points corresponding to two cities (e.g., Boston and New York) such that we would typically not “add” cities. However, if we consider the distance (i.e., vector) from Chicago to New York and the vector from New York to Boston, it makes sense to want to add those vectors.

### 2.37 Matrices

A *matrix* is a square or rectangular array of elements and can be thought of as an object in its own right, or, is sometimes best conceived as a collection of vectors. Just as for vectors, the matrices we work with all come from the field of real numbers. Hence, if  $\mathbf{A}$  is a matrix, we write  $\mathbf{A} \in \mathbb{R}^{m \times n}$  and denote the matrix by

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

The *dimension* of a matrix is the number of *rows* by the number of *columns*. If there are  $m$  rows and  $n$  columns, the dimension of the matrix is  $m \times n$ . A square matrix has  $m = n$ , that is, the same number of rows as columns. Note that the dimension of a matrix is not the same thing as the dimension of its *basis*, which is the dimensionality of the largest number of linearly independent rows or columns of the matrix, to be discussed later. We note that a matrix has a two-dimensional structure. If we increase this to three dimensions or greater, the object then becomes a *tensor* (Murphy, 2022). The study of tensors is well beyond the scope of this book.

There are several ways of building matrices in R. A popular way is to specify the number of rows and columns via the `matrix()` function:

```
> A <- matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow=TRUE)
> A
```

```
      [,1] [,2] [,3]
[1,]     1     2     3
[2,]     4     5     6
```

The dimension of the matrix is revealed by the following, having 2 rows and 3 columns (2 3):

```
> dim(A)
[1] 2 3
```

Creating a matrix in Python is just as easy:

```
A = [[1, 4, 5],
      [-5, 8, 9]]
```



A

```
Out [320]: [[1, 4, 5], [-5, 8, 9]]
```

Note that in the above matrix (where each row is a *list*) constructed in Python, `[[1, 4, 5], [-5, 8, 9]]` corresponds to the matrix

$$\begin{bmatrix} 1 & 4 & 5 \\ -5 & 8 & 9 \end{bmatrix}$$

Later, when we sample code for carrying out statistical and machine learning applications, data will often appear by row as above (i.e., `[[1, 4, 5], [-5, 8, 9]]`), but it is important to realize what is being communicated, which is illustrated by the lists or matrix above. If the data were simply a single row vector (or list, more generally), then it would have been represented without inner square brackets, as in the following:

```
[[1, 4, 5, -5, 8, 9]]
```

as we can readily verify in Python,

```
X = [[1, 4, 5, -5, 8, 9]]
```

X

```
Out [145]: [[1, 4, 5, -5, 8, 9]]
```

A more efficient approach to building vectors in Python is to use `numpy`:

```
import numpy as np
x = np.array([1, 4, 5, -5, 8, 9])
x
Out [125]: array([ 1, 4, 5, -5, 8, 9])
```

### 2.37.1 Identity Matrix

The *identity matrix* is a very important *square* matrix that reappears several times in statistical settings and is denoted by having only a series of “1” values along the main diagonal and zeroes everywhere else. As an example, consider the following  $3 \times 3$  identity matrix:

$$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where  $\mathbf{I}_3$  indicates a matrix having 3 rows and 3 columns. More generally, an identity matrix  $\mathbf{I}_n$  has  $n$  rows and  $n$  columns. A *zero matrix* is one for which all entries in the matrix have a value of 0.

A basic result involving identity matrices includes  $\mathbf{IA} = \mathbf{AI}$  where  $\mathbf{A}$  is a matrix of the same dimension as  $\mathbf{I}$ .

Identity matrices can be easily built in R by the following via the `diag()` function:

```
> A <- diag(x = 5)
> A
```

|      | [,1] | [,2] | [,3] | [,4] | [,5] |
|------|------|------|------|------|------|
| [1,] | 1    | 0    | 0    | 0    | 0    |
| [2,] | 0    | 1    | 0    | 0    | 0    |
| [3,] | 0    | 0    | 1    | 0    | 0    |
| [4,] | 0    | 0    | 0    | 1    | 0    |
| [5,] | 0    | 0    | 0    | 0    | 1    |

In Python, we can create an identity matrix in the following way:

```
np.identity(5)
Out [321]:
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

### 2.37.2 Transpose of a Matrix

The transpose of a matrix is analogous to the transpose of a vector, which we denote by

$\mathbf{A}^T$  or  $\mathbf{A}'$

As an example, consider the following square matrix  $\mathbf{A}$  once more:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

We obtain the transpose by exchanging the first row for the first column and the second row for the second column:

$$\mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

We obtain the transpose of a matrix in R via `t()`:

```
> A <- matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow=TRUE)
> A
```

|      | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1    | 2    | 3    |
| [2,] | 4    | 5    | 6    |

```
> t(A)
```

```
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

In Python, there are many ways of obtaining a transpose, such as the following:

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6]])
transposed_matrix = np.transpose(matrix)
print(transposed_matrix)
[[1 4]
 [2 5]
 [3 6]]
```

### 2.37.3 Symmetric Matrices

Symmetric matrices are of fundamental importance in linear algebra and by consequence also form a group of matrices of significant importance in statistics. A symmetric matrix is one for which the *transpose of the matrix is equal to the original matrix*. That is, a symmetric matrix is one for which  $\mathbf{A} = \mathbf{A}^T$ . Symmetric matrices feature many desirable properties, including the fact that the inverse of a symmetric matrix is also symmetric. Further, if a matrix is not symmetric, there is no guarantee that its eigenvectors will be orthogonal. Hence, when we speak of eigenvectors in general being orthogonal in the context of this book, it is usually implied that we are working with a symmetric matrix.

Two very important classes of symmetric matrices in statistics are those of *covariance* and *correlation* matrices, discussed earlier in our survey of essential probability and statistics.

### 2.37.4 Matrix Addition and Multiplication

For any two matrices of the same dimension, as was the case for vectors, we add and subtract matrices element by element. Consider the following two matrices,  $\mathbf{A}$  and  $\mathbf{B}$ , each of dimension  $m \times n$ :

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdot & \cdot & & \cdot \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \cdot & \cdot & & \cdot \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

The addition of matrices  $\mathbf{A}$  and  $\mathbf{B}$  occurs element by element:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \cdot & \cdot & & \cdot \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix}$$

Note that matrix addition is *commutative*, such that  $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$ . Note that this parallels scalar algebra in which for two numbers  $a + b = b + a$ . It is also associative, which means for matrices  $\mathbf{A}, \mathbf{B}, \mathbf{C}$ ,  $(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$ .

We add matrices in Python as follows:

```
a
Out [330]:
array([[1, 2, 3],
       [4, 5, 6]])
b
Out [331]:
array([[1, 2, 3],
       [4, 5, 6]])
a + b
Out [332]:
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

To multiply matrices  $\mathbf{AB}$ , we compute along the first row and down the first column, then the first row, second column, etc., then the second row, first column, then the second row, second column, and so on, as the following example illustrates (Fox, 2016):

$$\begin{aligned}
 & \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 & \quad (2 \times 3) \quad \quad (3 \times 3) \\
 & = \begin{bmatrix} 1(1) + 2(0) + 3(0), & 1(0) + 2(1) + 3(0), & 1(0) + 2(0) + 3(1) \\ 4(1) + 5(0) + 6(0) & 4(0) + 5(1) + 6(0) & 4(0) + 5(0) + 6(1) \end{bmatrix} \\
 & = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}
 \end{aligned}$$

Recall that in scalar multiplication in basic algebra,  $a \cdot b$  yields the same product as  $b \cdot a$ . With matrices, however,  $\mathbf{AB}$  does not, in most circumstances, equal  $\mathbf{BA}$ . That is,  $\mathbf{AB} \neq \mathbf{BA}$ , which means contrary to scalar addition and subtraction, *matrix multiplication is generally not commutative*. That is, multiplication for matrices is *noncommutative*. What is more, not just any two matrices can be multiplied. For matrices to be conformable for multiplication, the number of *columns* in the first (left) matrix must equal the number of *rows* in the second (right) matrix. For intuition as to why matrix multiplication is the way it is, see Fox (2016).

### 2.37.5 *Meaning of Matrices: Matrices as Data and Transformations*

First and foremost, matrices are mathematical objects in and of themselves. The mathematical treatment of matrices is actually quite recent, having their modern origin in the 1800s. When you think about this, realize that in the history of mathematics this is almost like saying that the modern concept of matrices was invented *yesterday* since the history of mathematics is so vast. In pure

mathematics, matrices are studied for their own sake. It may be challenging for the applied reader to appreciate how one can study them for their own purpose, but this is exactly what motivates pure mathematicians (and philosophers): the study of abstract (but far from arbitrary) objects.

In a more applied context, matrices are generally used to denote two ideas. The first is that of a *transformation*; the second is that of *data*. The transformation idea is closer to pure linear algebra, while the data idea is closer to statistics. In fact, it can be shown that any matrix corresponds to a linear transformation such that

$$T(\mathbf{v}) = \mathbf{A}\mathbf{v} \quad (2.97)$$

where  $T$  is a mapping from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ ,  $\mathbf{v}$  is a vector, and  $\mathbf{A}$  is a matrix. The above reads that a linear transformation on the vector  $\mathbf{v}$  is equivalent to pre-multiplying that vector by the matrix  $\mathbf{A}$ . In the simpler case where  $\mathbf{v}$ , in theory, is not a vector but instead a scalar  $b$  (or simply a vector with a single element), then we would have,

$$T(b) = \mathbf{A}b$$

which as was the case for vectors, communicates that in order to (linearly) transform the scalar  $b$  we multiply it by the matrix  $\mathbf{A}$ . If  $\mathbf{A}$  is a very simple matrix, say of one element such that we can simply call it a scalar as well, then the above reduces to

$$T(b) = ab$$

That is, to linearly transform  $b$ , we multiply it by the scalar  $a$ .

To appreciate the second interpretation, that of a matrix as a *data matrix*, consider the following matrix,  $\mathbf{A}$ :

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1' \\ \mathbf{x}_2' \\ \vdots \\ \mathbf{x}_n' \end{bmatrix}$$

Each column of the above matrix might represent a variable, for instance, IQ, GPA, etc., where on the right side of the equality are the corresponding variables  $\mathbf{x}_1'$  through  $\mathbf{x}_n'$  where ' indicates transpose. Linear and matrix algebra is largely the study of different kinds of matrices and their behavior under a variety of conditions. We will see matrices “at work” throughout this book. The matrices we survey here and in this book are, relatively speaking, very small. In practice, matrices can have thousands or even millions of rows and columns. Indeed, in the age of “Big Data,” matrices can be so large that they can challenge the capacities of modern-day computing power.

### 2.37.6 Kernel (Null Space)

The *kernel*, or *null space*, of a linear transformation is the set of vectors such that  $T(\mathbf{x}) = \mathbf{0}$ . For example, the vector  $\mathbf{x}$

$$\begin{bmatrix} -2 \\ -1 \\ 3 \end{bmatrix}$$

is in the kernel or null space of  $\begin{bmatrix} 1 & 1 & 1 \\ 2 & -1 & 1 \end{bmatrix}$  as a result of  $\begin{bmatrix} 1 & 1 & 1 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ . That is,

$T(\mathbf{x}) = \mathbf{0}$ . In words, the kernel of a linear transformation is the set of vectors that maps the transformation to zero.

### 2.37.7 Trace of a Matrix

The *trace* of a matrix  $\mathbf{A}$ , denoted  $\text{tr}(\mathbf{A})$ , is the sum of elements along the main diagonal:

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii} \quad (2.98)$$

where  $a_{ii}$  denotes the values along the main diagonal. Traces only exist for square matrices. Among the properties of traces include the following:

$$\begin{aligned} \text{tr}(\mathbf{A}) &= \text{tr}(\mathbf{A}^T) \\ \text{tr}(\mathbf{A} + \mathbf{B}) &= \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B}) \\ \text{tr}(\mathbf{AB}) &= \text{tr}(\mathbf{BA}) \end{aligned}$$

The trace of a matrix multiplied by a constant  $c$  is also equal to that constant times the trace of the matrix, and we write,

$$\text{tr}(c\mathbf{A}) = c\text{tr}(\mathbf{A})$$

All of these relations can be easily verified by the reader. Traces play a meaningful role in multivariate statistics. For instance, at times we are only interested in values along the main diagonal of a matrix such as when we wish to consider only variances in the covariance matrix and ignore covariances in the off-diagonal. Instead of communicating “sum all the diagonal entries of the matrix to get the total variance,” we simply say “compute the trace of the matrix.” The trace will be featured when we consider multivariate test statistics, for instance, later in the book.

The trace of a matrix in R can be obtained by `sum(diag())`. In Python, `matrix.trace()` in `numpy()` will generate the trace.

## 2.38 Linear Combinations

If there is one mathematical element or object that is common to virtually all of statistics and machine learning, it is that of a *linear combination*, which is defined quite simply as

$$\ell_i = a_1 y_1 + a_2 y_2 + \cdots + a_p y_p \quad (2.99)$$

The above is very generic. Usually in statistics we will see linear combinations with alternate identifiers. We use the symbol “ $\ell_i$ ” to stand for “linear combination.” We can see that a linear combination is actually a sum of products,  $a_1 y_1$  through  $a_p y_p$  depending on how many variables we are summing. In the sum,  $a_1$  through  $a_p$  are scalars, which are usually real numbers. We can see that each variable  $y_1$  through  $y_p$  is being “weighted” by a given scalar. Our first impression here might be that variables receiving a greater weight have more “importance” (in some sense) than variables receiving less weight.

Where do we see linear combinations in statistics and machine learning? Everywhere! As an example, as the reader will surely already recognize, recall the simple linear regression model,

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon \quad (2.100)$$

We can see that the above is nothing more than a linear combination, comprising of an intercept,  $\beta_0$ , which is a constant scalar, and a variable  $x_i$  that is weighted by the scalar  $\beta_1$ , then trailing off with an error term,  $\varepsilon$ .

Linear combinations are utilized all over the place in ANOVA-type models as well. Consider the following linear combination between population means,

$$C_i = c_1 \mu_1 + c_2 \mu_2 + \cdots + c_J \mu_J = \sum_{j=1}^J c_j \mu_j \quad (2.101)$$

The linear combination becomes a *linear contrast* when we impose the constraint that the weights,  $c_1$  through  $c_J$  must sum to 0, that is,

$$\sum_{j=1}^J c_j = 0 \quad (2.102)$$

Another place where we see linear combinations in statistics is in principal components analysis, or “PCA” for short. In fact, a principal component *is* a linear combination of the sort,

$$y_i = a_1 x_{i1} + a_2 x_{i2} + \cdots + a_p x_{ip}$$

As we will see later in the book, the coefficients  $a_1$  through  $a_p$  will be values of the eigenvector resulting from the eigen decomposition performed to obtain the components, and the  $x_{i1}$  through  $x_{ip}$  will be variables on which the PCA is being performed.

The *span* of a set of vectors is the set of vectors in a vector space consisting of all linear combinations of those vectors, while the *basis* of a set of vectors is the number of linearly independent vectors that span the space, which means out of all vectors in the space, the basis is the minimal number of vectors that can, essentially, generate the space. There are typically lots of vectors in a vector space, but the *basis* vectors are those for which all those other “less important” vectors can be generated. This is why basis vectors are so important. The basis vectors are essentially the “boss” of the vector space.

Linear combinations are also featured in the committee machine of *boosting*, where one weights classifiers by scalars to generate a stronger learner from a series of weak learners. We feature a discussion of boosting later in the book.

### 2.39 Determinants

Determinants play a central role in linear algebra and statistics. Determinants are only defined for *square* matrices. While determinants can be calculated for increasingly large matrices, they are most easily demonstrated for small matrices. For example, the determinant of a  $2 \times 2$  matrix  $\mathbf{A}$  can be easily computed as follows:

$$\begin{aligned} |\mathbf{A}| &= \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \\ &= a_{11}a_{22} - a_{12}a_{21} \end{aligned} \quad (2.103)$$

When would such a computation be useful in statistics? We note immediately that through the subtraction of the product  $a_{12}a_{21}$ , the determinant becomes useful for matrices such as the covariance matrix in which we wish to consider variances in the product  $a_{11}a_{22}$  but then subtract out the product of covariances in  $a_{12}a_{21}$ . In the case where covariances are equal to 0, it follows that the determinant is computed simply as

$$a_{11}a_{22} - a_{12}a_{21} = a_{11}a_{22} - 0 = a_{11}a_{22}$$

As we will see shortly, this is one reason why the determinant is an ideal measure for constructing an overall “generalized” version of variance in a covariance matrix.

For  $3 \times 3$  matrices and beyond, the computation of the determinant becomes increasingly unwieldy and is, not surprisingly, best left to a computer (even very higher-order determinant calculations can be time-consuming for computers). Still, there is intuition that can be gleaned from computing a determinant for a  $3 \times 3$  matrix (we leave it to the reader to extract the algorithm):

$$\begin{aligned} |\mathbf{A}| &= \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \\ &= a_{11} \det \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} (-1)^2 + a_{12} \det \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} + (-1)^3 + a_{13} \det \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} (-1)^4 \end{aligned} \quad (2.104)$$

Geometrically, we can also easily visualize the determinant in this case as the subtraction of shapes (areas). For example, consider the matrix  $\mathbf{A}$  defined as

$$\mathbf{A} = \begin{bmatrix} 3 & 2 \\ 1 & 3 \end{bmatrix}$$

The determinant of  $\mathbf{A}$  is equal to  $(3)(3) - (2)(1) = 7$ . As described in Young (2023) and shown in Figure 2.12, this subtraction can be visualized as the subtraction of corresponding areas (i.e., the larger area  $(3)(3)$  minus the smaller  $(2)(1)$ ).



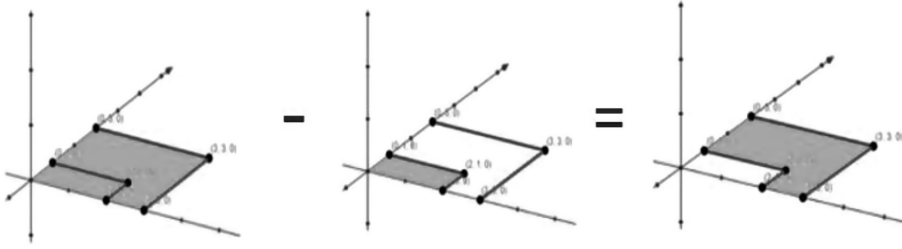


Figure 2.12 Determinant as a subtraction of areas. For this matrix, the smaller area (2) is being subtracted from the larger (9).

Figure 2.12 is for the  $2 \times 2$  case and hence the measure of *area* is suitable for geometrical representation. In matrices of  $3 \times 3$  or greater, we shall require a measure of *volume*, and hence, for the general case, we will typically speak of the *volume* of a matrix instead of that of area. From this geometrical interpretation, we see then that *the determinant is the factor by which the volume of a matrix increases or decreases*. A determinant equal to 0 corresponds to a *parallelogram* (in the  $2 \times 2$  case) having zero area or *parallelepiped* (in the  $3 \times 3$  or higher case) having zero volume (Young, 2023). As an example, consider the  $3 \times 3$  matrix in Figure 2.13 with corresponding drawn parallelepiped.

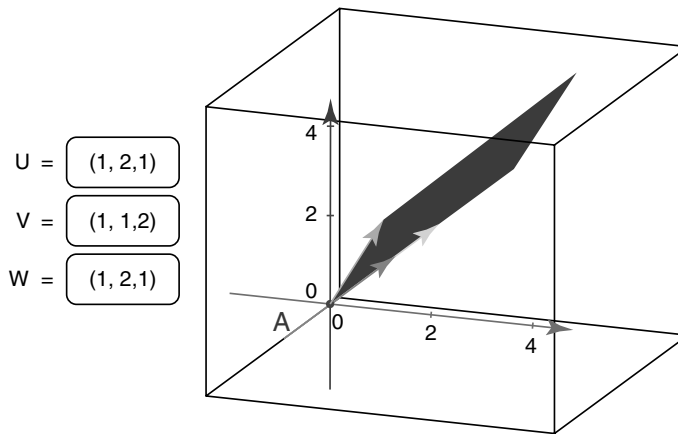


Figure 2.13 Parallelepiped with determinant (and hence, volume) equal to 0.

We see that the parallelepiped has collapsed upon itself and contains no volume since the determinant of the corresponding matrix is equal to 0. These notions of area and volume are not of terrible interest in this abstract sense but will be very useful as we consider statistical measures that feature determinants later in the book. Certainly, in virtually every multivariate method, one will find determinants somewhere.

A few of the many properties of these measures have been established, which include the following:  $|\mathbf{A}| = |\mathbf{A}^T|$ ,  $|c\mathbf{A}| = c^n |\mathbf{A}|$ ,  $|\mathbf{AB}| = |\mathbf{A}| |\mathbf{B}|$ . A determinant equal to 0 implies the matrix is not invertible, which means the matrix is *singular*. This has consequences for statistical procedures (such as regression) that typically require an inverse. The matrix is called *nonsingular* if it is invertible and hence  $|\mathbf{A}| \neq 0$ . Determinants also have numerous relationships with other objects in linear algebra. For instance, the determinant of a *diagonalizable matrix* is equal to the product of its eigenvalues, where diagonalizing a matrix, in this context, means transforming the matrix such

that the values along the main diagonal are the eigenvalues of the matrix and all other values are equal to 0. More generally, a diagonal matrix is a matrix such that all entries off the main diagonal are equal to 0. Hence, a matrix with eigenvalues down its main diagonal and all other values in the matrix equal to 0 is a diagonal matrix.

Matrix decompositions include the *LR decomposition*, *Cholesky decomposition*, and *QR decomposition*, which are also used as efficient methods for computing determinants. For details, see Strang (1993).

Determinants in R can be easily computed. For example, the following computes the determinant of an identity matrix:

```
> A <- diag(x = 5)
> det(A)
[1] 1
```

In Python, we can compute the determinant of a matrix using `numpy()`:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 4 - 6 = -2$$

```
np.linalg.det(a)
Out [336]: -2.0000000000000004
```

## 2.40 Means and Variances of Matrices

Recall that the goal of statistical inference is largely to estimate parameters. In a univariate context, we often wish to estimate the population mean,  $\mu$ . A good estimator of  $\mu$  is that of the sample mean,  $\bar{y}$ , since recall that  $\bar{y}$  is an unbiased estimator of  $\mu$ . That is, the expectation,  $E$ , of  $\bar{y}$  is that of  $\mu$ ,  $E(\bar{y}) = \mu$ . Likewise, we saw that  $E(s^2) = \sigma^2$ .

Vectors and matrices are simply larger objects than these comparative univariate counterparts, and hence it makes sense that we should want to estimate means and variances on these objects as well. Not surprisingly, the expectation of a mean vector,  $\bar{\mathbf{X}}$ , is that of a population vector,  $\mu$ , and we write

$$E(\bar{\mathbf{X}}) = \mu \tag{2.105}$$

However, contrary to simply estimating a variance,  $s^2$  as we did in the univariate case, in the multivariate case we not only have variances, but also *covariances*. The variances and covariances you will recall are packaged up in the covariance matrix,  $\Sigma$ . As usual, we typically do not have the population covariance matrix at our disposal and need to estimate it. An unbiased estimator of  $\Sigma$  can be obtained as a function of the sample covariance matrix,  $\mathbf{S}_n$ :

$$E\left(\frac{n}{n-1}\mathbf{S}_n\right) = \Sigma \tag{2.106}$$

It can easily be shown that the bias of the estimator,  $\mathbf{S}_n$ , is equal to  $-(1/n)\Sigma$  (Johnson and Wichern, 2007, p. 121).

## 2.41 Determinant as a Generalized Variance

As a result of the covariance matrix incorporating both variances and covariances, we can establish a generalized version of the variance that parallels the univariate case. A measure that is often used to denote a generalized variance in a multivariate setting is the determinant of the covariance matrix,  $|\mathbf{S}|$ . For a visualization of the generalized variance, see Johnson and Wichern (2007, p. 125).

It should be emphasized that  $|\mathbf{S}|$  only provides a measure of variance and like  $s^2$  in the univariate setting does not by itself provide any information as to the *pattern* of the data or direction. That is, without visualization, a report of  $|\mathbf{S}|$  simply gives us a sense of the volume of the space; *it does not by itself tell us which directions are most prominent or emphasized in the data*. This is not surprising since as mentioned the reporting of a univariate variance  $s^2$  does not either by itself give us information on the *particular* distances between data objects or points; it only provides an *average* of those (squared) distances. A generalized variance of  $|\mathbf{S}| = 0$  implies the entire space has collapsed onto itself with a corresponding volume equal to 0. A volume greater than zero is indicated by  $|\mathbf{S}| > 0$ .

Instead of the determinant of the covariance matrix, we can also conceive of a *total* variance as the sum of univariate variances along the main diagonal of the covariance matrix,

$$s_{11}^2 + s_{22}^2 + \cdots + s_{pp}^2$$

where  $s_{11}^2$  through  $s_{pp}^2$  are variances in the corresponding covariance matrix. More concisely, this version of the variance is simply the *trace* of the covariance matrix. Note that this interpretation does not include covariances and hence is not a true measure of “multivariate” variance in the same spirit as that of the generalized variance,  $|\mathbf{S}|$ .

## 2.42 Matrix Inverse

The inverse of a matrix plays a fundamental role in matrix algebra and parallels the notion of an inverse in ordinary scalar algebra. To understand what an inverse is, consider a simple example from scalar algebra. The inverse of the number 6 is that number which, when multiplied by 6, will equal 1. That number, of course, is  $\frac{1}{6}$ , since  $\frac{1}{6}$  multiplied by 6 yields 1.

The concept is analogous in matrix algebra. The inverse of matrix  $\mathbf{A}$  is that matrix  $\mathbf{A}^{-1}$  such that  $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$ . For two matrices  $\mathbf{B}$  and  $\mathbf{A}$ , if  $\mathbf{B}\mathbf{A} = \mathbf{I}$ , then  $\mathbf{B}$  is called the *left inverse* of  $\mathbf{A}$ . For another matrix  $\mathbf{C}$ , if  $\mathbf{A}\mathbf{C} = \mathbf{I}$ , then  $\mathbf{C}$  is the *right inverse* of  $\mathbf{A}$ . An invertible matrix has both a right and a left inverse (which are identical) and hence in this book, whenever we speak of the inverse of a matrix, the distinction between left vs. right inverse is immaterial. In this book, the inverse of a matrix is simply *the one and only inverse*.

Computing inverses in numerical linear algebra is “big business” and some mathematicians have devoted much of their lives to developing new and better algorithms for approximating inverses. Thankfully, for simple matrices, computing inverses is relatively easy and straightforward. As an example, consider the generic matrix  $\mathbf{A}$ :

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

The inverse of  $\mathbf{A}$  is equal to

$$\mathbf{A}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \quad (2.107)$$

An inverse for a matrix exists only if the determinant of that matrix is unequal to 0. If the determinant is equal to 0, then the inverse does not exist. Hence, for the simple square  $2 \times 2$  matrix  $\mathbf{A}$  above, we require

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \\ &= a_{11}a_{22} - a_{12}a_{21} \neq 0 \end{aligned}$$

The properties of inverse matrices are many. A few of the more important ones are the following:

$(\mathbf{A}^{-1})^{-1} = \mathbf{A}$ , the inverse of the inverse of a matrix is the original matrix. This parallels what we would expect in simple scalar algebra, since the inverse of  $1/a$  is equal to  $a$  and so the inverse of the inverse of  $1/a$  is  $1/a$ .

$(\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$ , the inverse of the transpose is equal to the transpose of the inverse.

$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ , the inverse of a product is equal to the product of inverses in reverse order.

$|\mathbf{A}^{-1}| = 1/|\mathbf{A}|$ , the determinant of the inverse is equal to the reciprocal of the determinant.

In R, we can get the inverse by the function `solve()`. For example, if  $\mathbf{A}$  is a square matrix, the inverse is computed as `solve(A)`. In Python, we can compute the inverse using `numpy().linalg.inv()`:

```
a = np.array([[1, 2], [3, 4]])
np.linalg.inv(a)
Out [337]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

If  $\mathbf{A}^{-1}$  does not exist, such as with rectangular matrices, a pseudo-inverse may be used known as the *Moore-Penrose inverse*. For details, see Rencher and Christensen (2012).

### 2.42.1 Nonexistence of an Inverse and Singularity

To emphasize the point already made, the most important property (or “consequence”) of inverses, for our purposes anyway, is that if the inverse of a matrix does not exist, then it implies that that matrix has a determinant equal to 0 and is thus deemed *singular*. Hence, it is generally a “big deal” in statistics when a matrix is not invertible. An additional consequence is that a non-invertible matrix also implies a *dependency* within the matrix; that is, not all rows and columns are linearly independent. We discuss linear independence and dependence shortly. In brief, for now, matrices that are singular and have linearly dependent columns or rows, or even nearly so, are statistically generally regarded as *ill-conditioned* and not “well-behaved.” Non-invertible matrices can provide a lot of problems when it comes to statistical analyses.

### 2.43 Quadratic Forms

A quadratic form is nothing more than a polynomial for which *all* terms (not just *some*) are of the second degree. An example of a quadratic form is  $y = x^2 + z^2$ . The following polynomial  $y = x_1x_2x_3$  is not a quadratic form since it involves a cubic form (Hubbard & Hubbard, 2002). In vectors and matrices, as we will soon elaborate on, an important quadratic form is that of  $\mathbf{x}'\mathbf{A}\mathbf{x}$ , for which if  $\mathbf{x}'\mathbf{A}\mathbf{x} > 0$  for all vectors  $\mathbf{x} \neq \mathbf{0}$ , it indicates that  $\mathbf{A}$  (symmetric) is positive definite. We will come across quadratic forms in one way or another throughout this book mostly implicitly and “behind the scenes.”

### 2.44 Positive Definite Matrices

Referring to the matrix representation of quadratic forms above,  $\mathbf{x}'\mathbf{A}\mathbf{x}$ , it is easy to see that for  $\mathbf{x} = \mathbf{0}$ ,  $\mathbf{x}'\mathbf{A}\mathbf{x} = 0$ . Now, consider the case where  $\mathbf{x}'\mathbf{A}\mathbf{x} \geq 0$ , where  $\mathbf{A}$  is a symmetric nonzero matrix (since zero matrices are, by definition, also symmetric). As already noted when discussing quadratic forms, if the only way for equality to hold for  $\mathbf{x}'\mathbf{A}\mathbf{x} \geq 0$  is for  $\mathbf{x}$  to be the zero vector, then  $\mathbf{A}$  is called *positive definite*, which implies that the quadratic form  $\mathbf{x}'\mathbf{A}\mathbf{x}$  will be positive, that is,  $\mathbf{x}'\mathbf{A}\mathbf{x} > 0$ .

The above result has important and pragmatic implications. Most pragmatically for statistics and machine learning in an applied sense, positive definite symmetric matrices are those for which *all eigenvalues are strictly positive*. *Semi-positive definite* matrices, on the other hand, have some (at least one) eigenvalues equal to 0 and all others positive, while indefinite matrices have at least one eigenvalue that is also negative. A further implication of positive definite matrices is that such matrices have all row and column vectors linearly independent, which implies that the matrix is of full rank and hence nonsingular, and this equates to the determinant of the matrix being nonzero.

### 2.45 Inner Products

One of the most significant foundational aspects of linear algebra is that of *inner products*, also generally known as *dot products*. Though the two terms are often used interchangeably, as we do as well in this book, they are not exactly the same. The dot product can be considered a *special case* of the more general Euclidean inner product, where dot products are usually restricted to finite Euclidean spaces of  $\mathbb{R}^2$  or  $\mathbb{R}^3$  but where *Euclidean inner products* are generalizable to higher dimensions, including to infinite-dimensional vector spaces (Anton & Rorres, 2000, p. 163). Inner products are at the heart of many computations in linear algebra and hence are also heavily used in statistics and machine learning. For example, covariance and correlation are based entirely on the notion of a dot product. The *kernel trick* featured in support vector machines and other algorithms also features the inner product, where, as we will see, computation of the kernel in the original dimensionality of the data points generalizes to an inner product space in higher dimensions.

An inner product is defined as

$$\mathbf{x}'\mathbf{y} = \mathbf{y}'\mathbf{x} = x_1y_1 + x_2y_2 + \cdots + x_my_m \quad (2.108)$$

where  $\mathbf{x}'\mathbf{y} = \mathbf{y}'\mathbf{x}$  indicates commutativity. The inner product is also distributive where for three vectors,  $\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = \mathbf{x}\mathbf{y} + \mathbf{x}\mathbf{z}$ . A vector space together with an inner product is called an *inner product space* (Roman, 2008).

Recall that the *length* of a vector  $\mathbf{x}$  can be expressed as the square root of its inner product,

$$L_x = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \sqrt{\mathbf{x}'\mathbf{x}}$$

For two random vectors,  $\mathbf{x}$ ,  $\mathbf{y}$ , the *Cauchy-Schwarz inequality* holds that

$$|\langle \mathbf{x}'\mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\| \quad (2.109)$$

which in words says that the absolute value of the inner product,  $\mathbf{xy}$ , is equal to or smaller than the product of individual lengths (Bilodeau & Brenner, 1999). Equality exists in the case that  $\mathbf{x}$  and  $\mathbf{y}$  are linearly dependent (i.e., scalar multiples since there are only two vectors). A proof of the inequality can be found in Garcia and Horn (2017, p. 93).

In Python, we can easily compute the inner product:

```
a = np.array([1, 2, 3])
b = np.array([0, 1, 0])
np.inner(a, b)
Out [342]: 2
```

A *Hilbert space* is an abstract vector space with an inner product that is complete under the metric that is induced by the inner product (Roman, 2008, p. 327). Formality matters here, but we can make this definition much more practical. Pragmatically, as it regards statistics and machine learning, a Hilbert space is essentially a kind of feature space (Mohri et al., 2018). The concept of inner products here is key because it forms the basis of the aforementioned kernel function or *kernel trick*. The idea of the kernel trick is to compute a given kernel using raw data in the original space (called the input or “data space”) so as to tell us something about the inner product of vectors in the higher “feature” space. That is, *the kernel function in the data space essentially reproduces kernel Hilbert spaces*. In this way, the elements of the kernel function “live” in the more abstract higher-dimensional Hilbert space, which allows us to compute kernels in low dimensions to tell us something about inner products in higher dimensions, that is, in the feature space. We extend this discussion of the kernel trick shortly.

## 2.46 Linear Independence

The idea of linear independence is of fundamental importance and has wide implications in statistics and machine learning. Consider two vectors,  $\mathbf{x}$  and  $\mathbf{y}$ , weighted by scalars  $c_1$  and  $c_2$ ,

$$c_1\mathbf{x} + c_2\mathbf{y} = \mathbf{0}$$

If both  $c_1$  and  $c_2$  are both equal to 0, then the equality is easily shown to be true. This is the *trivial* case, and hence not very interesting. If, on the other hand, we can find scalars  $c_1$  and  $c_2$  both not equal to 0 such that the equality still holds, then the vectors  $\mathbf{x}$  and  $\mathbf{y}$  are said to be *linearly dependent*. It essentially means, *for the case of only two vectors*, that we can write one vector as a *scalar multiple* of the other. In the case of more than two vectors, linear dependence implies we can write one vector as a linear combination of others in the set. If we cannot find such a set of scalars (other than zero), then the set of vectors is *linearly independent*.

As mentioned, the above example features only two vectors. We can generalize this principle of linear independence and dependence to a much broader set of vectors, indeed,  $k$  vectors,

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_k\mathbf{x}_k = \mathbf{0}$$

If there exist scalars  $c_1$  through  $c_k$  not all zero such that the above equality holds, then the vectors  $\mathbf{x}_1$  through  $\mathbf{x}_k$  are said to be *linearly dependent*. Otherwise, the set of vectors is linearly independent. Linear independence basically implies that we cannot write one vector as a linear combination of any others in the set, for which in the special case of two vectors in a set, the linear combination necessarily implies a scalar multiple. It is important to note that linear dependence or independence is relative to the *set of vectors* only. For example, two vectors are not themselves *intrinsically* linearly dependent or independent without considering the set in which they are featured. That is, the vector  $(0, 2, 4)$  is not by itself linearly independent or dependent; its status must be concluded *relative to the set in which it is referenced*. If that vector is the only (nonzero) vector in the set, then it is still regarded as linearly independent, but it is only so *relative to the set*, which in this case, consists of no other vectors.

## 2.47 Rank of a Matrix

The maximum number of linearly independent rows or columns of a matrix is the matrix's *rank*. A fundamental, if not surprising result of linear algebra is that *row rank* will always equal *column rank*, which also implies that the matrices  $\mathbf{A}$  and  $\mathbf{A}^{-1}$  (if it exists) have the same rank. Throughout our study of statistics, we will typically be interested in matrices of *full rank*, which implies that the total number of columns or rows is inclusive in the rank of the matrix.

As an example, if we have a  $3 \times 3$  matrix and say it is of “full rank,” then it means the number of linearly independent rows (and columns) is equal to 3. If we say a matrix is less than full rank, or “rank deficient,” then it suggests the rank of the matrix is less than the number of columns or rows. Rank-deficient matrices have one or more columns (or rows) that are linear combinations of other columns (or rows). For instance, the following matrix is less than full rank:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 5 \\ 1 & 2 & 6 \end{bmatrix}$$

It is less than full rank because the second column is simply a scalar multiple of the first. That is, the second column is exactly a factor of twice that of the first column. The rank of the matrix is 2 because there are only two linearly independent columns (and rows). Rank is also equal to the dimensionality of the largest submatrix  $\mathbf{A}$  (i.e., deleting certain rows and columns) that has a nonzero determinant (Izenman, 2008). In an applied sense, the rank of a matrix gives us a sense of how much information is “duplicated” in the matrix. A matrix of full rank tells us that each row (or column) is providing “unique” information. A matrix less than full rank tells us that there is, essentially, “overlap” in the matrix in terms of information. Though this is a nonrigorous way of conceiving rank, it is nonetheless helpful in grasping the concept in an applied sense.

As an example of computing the rank of a matrix in Python, we can use `numpy()`, where in the following the rank of the matrix is equal to 1:

```
np.ones(4)
Out[345]: array([1., 1., 1., 1.])
from numpy.linalg import matrix_rank
matrix_rank(np.ones(4))
Out[347]: 1
```

## 2.48 Orthogonal Matrices

Orthogonal matrices are those square matrices for which when we multiply by its transpose, we obtain the identity matrix. As an example,  $\mathbf{X}\mathbf{X}^T = \mathbf{I} = \mathbf{X}^T\mathbf{X}$  implies that  $\mathbf{X}$  is an orthogonal matrix. If  $\mathbf{X}^2 = \mathbf{X}$ , the matrix is regarded as *idempotent* (Izenman, 2008). For vectors, we can more simply define orthogonality as when the inner product  $\mathbf{x}\mathbf{y} = 0$ . Orthogonality also implies perpendicularity, which is expressed by  $\mathbf{x} \perp \mathbf{y}$ . *Orthonormal* vectors are orthogonal and have length equal to 1; hence if  $\mathbf{x}$  is orthonormal to some vector, then it stands that  $\|\mathbf{x}\| = 1$ .

## 2.49 Kernels, the Kernel Trick, and Dual Representations

Kernels play a significant role in machine learning and we see them in techniques such as *neural networks* and *support vector machines* (SVMs). Indeed, the “kernel trick” is something that quickly becomes common territory as one studies machine learning. In general, *kernel functions* (as distinct from the kernels featured earlier in relation to matrices) are used to define *similarity* between data objects. Kernels of inner products are often taken to determine the similarity of input data so that a linear classifier can be used to solve what is essentially a nonlinear problem. In support vector machines, the kernel trick was first introduced by Boser et al. (1992).

Even though much of the emphasis of statistics is on linearity, nonlinear problems pervade scientific practice. Attempting to model such nonlinearity via polynomial transformations, for instance, or other *basis function transformations*, as we will see later, can be complex, timely, and computationally infeasible. By taking kernels of input data, however, what was originally linearly *inseparable* data now becomes *separable* without having to actually mathematically go to those higher dimensions. That is, the kernel trick will allow us to stay in 2-D, for instance, when the data are only separable, for example, in 3-D. Essentially, the *kernel trick* or *kernel substitution* (Bishop, 2006, p. 292) allows one to take an infinite-dimensional problem and reduce it to a finite dimensional problem (Hastie et al., 2009).

The kernel function  $k$  is defined by

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}) = \phi(\mathbf{x})' \phi(\mathbf{x}') \quad (2.110)$$

where the inner product  $\mathbf{x}, \mathbf{x}'$  is a measure of *similarity* between two vectors, and hence the kernel function is considered a measure of such “likeness” of data. That is, the kernel function essentially measures how similar observations are to one another. As shown in (2.110), the kernel function here is also symmetric such that  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ . Computations of  $k(\mathbf{x}, \mathbf{x}')$  take place in the original data space, but equate to the inner product  $\phi(\mathbf{x})' \phi(\mathbf{x}')$  in the feature space.

Of the only key requirements of kernels is that they be *positive definite* as it can be shown that positive definite kernels induce an inner product in a Hilbert space (Mohri, 2018). The reader may recall that positive definiteness ensures that the eigenvalues of  $\mathbf{K}$ , the *kernel matrix* in this case, are (strictly) positive. What is more, *for a positive definite kernel, such will always produce (or “associate with”) a kernel matrix that is positive semidefinite* (Mohri et al., 2018, p. 108). A variety of kernels exist, such as *polynomial kernels*, *Gaussian kernels*, *sigmoid kernels*, and others. Of importance to remember with regard to kernels is that they need not be actually computed as inner products in the original data space  $\mathbf{x}, \mathbf{x}'$  for them to imply an inner product  $\phi(\mathbf{x})' \phi(\mathbf{x}')$  in the feature space. *Many other kernels are possible that do not computationally and directly calculate the inner product in data space. What is important is that the kernel expresses a degree of similarity in the data space, such that it induces an inner product space in the feature space.*

To better illustrate the kernel trick, consider the data in Figure 2.14 depicting two classes.



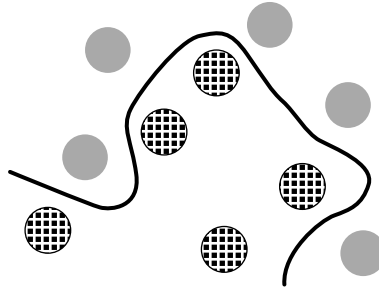


Figure 2.14 Data on two classes that are clearly not linearly separable.

Of course, the data in Figure 2.14 is not linearly separable; that is, we need a swishy curve to separate the two groups. By computing the *kernel* of the data however, *in its original dimensionality*, we essentially transform this nonlinearity into a linearly separable problem in the feature space. But why should this work? Why does the kernel trick do what it does?

The reason why the kernel trick works is related to *Mercer's theorem* (1909), which essentially yields the following very important result:

*A kernel can be used to approximate a function when it is known that the function exists even if knowledge of what that function is, is incomplete or not known.*

That is, the kernel essentially contains a wealth of information even about higher dimensionalities for which we may be currently unaware. The fact that we can work on the problem in its original configuration or via a kernel is known as *dual representation* (Bishop, 2006). In words, the so-called “kernel trick” essentially communicates the idea: *You're working on a complicated problem in higher dimensions, you're not possibly even aware of what those dimensions are, or even how many there might be, but if you compute the kernel in your existing lower-dimensional space you can learn a lot about those higher dimensions without ever having to go there **explicitly***. By way of fictional analogy, the kernel might allow you to stay and compute in this 3-D spatial universe while potentially learning something about a different one (e.g., a universe possibly in five dimensions).

As stated earlier, the utility of kernels is based on the fact that many data sets that we work with are not linearly separable. *Applying a kernel (hopefully) makes the data linearly separable and also allows for a solution that is convex* (Goodfellow et al., 2016). Convex solutions (as we will discuss later in more detail) are desirable because they have *global* minima and hence the algorithm will not get “stuck” in *local* minima. That is, an important implication of Mercer's condition is that it establishes the convexity of the optimization problem for support vector machines and similar algorithms (Mohri et al., 2018). Hence, a global minimum is assured in optimization.

To consolidate what the kernel trick does, it is helpful to remember the following:

*Nonlinearly Separable Data in Original Data Space + Kernel Trick = Linearly Separable Data in the Feature Space*

Aggarwal (2015) summarizes the kernel trick concisely in the following way in relation to support vector machines; however, the following quote is applicable to a much broader set of statistical and machine learning methods well beyond SVMs:

The kernel trick leverages the important observation that the SVM formulation can be fully solved in terms of dot products (or similarities) between pairs of data points. One does not need to know the feature values ... all computations are performed in the original space and the actual transformation does not need to be known as long as the kernel similarity function is known.

### 2.49.1 *When Are Kernel Methods Useful?*

Based on the above discussion, it stands to reason that kernels are most useful when dealing with data that are not linearly separable in the data space. If we did not have the kernel trick at our disposal we would need to map the data to higher dimensions of nonlinearity, that is, *enlarge the feature space to account for the complexity of the data*. This can be time-consuming, complicated, and sometimes not even doable. By using a kernel transformation, we do not need to map the data to higher dimensions at all but instead can simply use the kernel trick, which essentially means computing kernels in the *original dimensionality* of the problem instead of trying to enlarge the feature space ourselves. Kernel functions are, in general, and as already emphasized, less concerned about the actual data than they are about how *similar* data points are to each other (Alpaydin, 2020).

Yet another way of conceptualizing kernels is that they can also be seen as a way to express or “encode” prior knowledge. Consider that if  $x_i$  and  $x_j$  are similar before applying a function to them, then it will generally imply that functions of those observations,  $f(x_i)$  and  $f(x_j)$  are also similar. Indeed, in many procedures, the specifying of the *kernel matrix* is essentially somewhat akin to specifying a kind of “Bayesian” *prior knowledge*.

## 2.50 Systems of Equations

Many problems in statistical modeling at their core reduce to solving systems of linear equations. What is a “system” in this respect? A system is simply a *set of equations* for which we are seeking a solution for the variables that will solve the set of equations *simultaneously*. The idea of a *set* here is, as the reader is already likely familiar, a collection of objects, the objects in this case being *equations*. The solution set for one set of linear equations consisting of two equations will likely be different from the solution set for those same two equations but with a third equation added to the set. Once again, *context matters*.

A simple example will suffice to demonstrate the principle of solving a set of linear equations. Consider the following two linear equations:

$$4x_1 + 4x_2 = 5$$

$$2x_1 - 4x_2 = 1$$

When we plot these two equations, we obtain the linear system depicted in Figure 2.15.

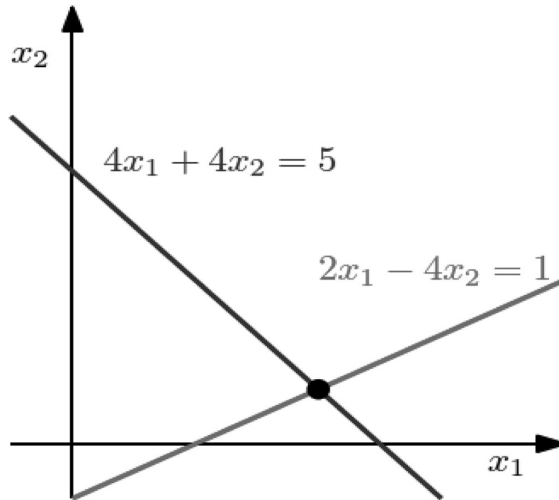


Figure 2.15 The solution set to the linear system is where the lines intersect.

The *solution set* is where the *intersection* occurs between the two lines. A system of linear equations such as this can be solved using the technique of *Gaussian elimination*, where the matrix is reduced to what is known as *reduced row-echelon* form which allows one to easily see the solution. We do not review Gaussian elimination here due to space limitations but also because it is not terribly enlightening from an applied statistics point of view.

We can solve systems of linear equations in R quite easily. For example, consider the following two equations set up in R, where first we designate the two coefficients for  $x_1$  and  $x_2$  on the left-hand side of the equation:

```
> left_matrix <- matrix(c(5, 2, 4, 6), nrow=2)
> left_matrix

      [,1] [,2]
[1,]    5    4
[2,]    2    6
```

Next, we define the  $y$  values on the right-hand side:

```
> right_matrix <- matrix(c(35, 36), nrow=2)
> right_matrix

      [,1]
[1,]   35
[2,]   36
```

We next solve the equation:

```
> solve(left_matrix, right_matrix)
```

```
      [,1]
[1,]     3
[2,]     5
```

We can see that the solutions to the linear system are 3 and 5. That is, we have just solved the equations

$$5x_1 + 4x_2 = 35$$

$$2x_1 + 6x_2 = 36$$

for which when we substitute solutions  $x_1 = 3$  and  $x_2 = 5$  we obtain,

$$5(3) + 4(5) = 35$$

$$2(3) + 6(5) = 36$$

In Python, suppose we want to solve the following system of equations:

$$24a + 4b = 35$$

$$8a + 4b = 94$$

```
a = np.array([[24, 4], [8, 4]])
b = np.array([35, 94])
print(np.linalg.solve(a,b))
[-3.6875 30.875 ]
```

The solutions to the above equations are thus  $-3.6875$  and  $30.875$ .

There are a variety of methods useful for solving systems of linear equations, including the *elimination method*, *substitution method*, and *graphical method*. Regression analysis is an example of solving a system of equations when an exact solution is not achievable and hence an *approximation* is required, typically that of ordinary least-squares. The approximation is required as a result of there being too many equations to provide a unique solution. That is, the set of equations is *overdetermined*, meaning that there are too many equations because of the scatter of the data and as such many lines *could* be fit. However, we only wish to fit a single line, and that will mean approximating weights for  $\beta_0$ , the intercept, and  $\beta_1$ , the slope.

Many times when considering systems of equations, some equations will be a consequence of others. That is, one equation may reveal the same *information* as one or more other equations. For example, if two equations are scalar multiples of one another, then one equation, as previously discussed, essentially duplicates the “information” provided by the first. Such can be visually depicted by two lines taking up the same space (i.e., laying on top of each other). As an easy example, consider the following two equations:

$$x_1 + x_2 = y$$

$$2x_1 + 2x_2 = 2y$$

Clearly, the second equation is simply a scalar multiple of the first and as such provides no new “information” to the system. As previously discussed in terms of vectors, the equations are said to be *linearly dependent*.

Note that we have only surveyed linear equations above. What made them linear was that each term in the equation was raised to no power higher than one. For instance,

$$x_1 + x_2 = y$$

only featured powers to  $x_1$ ,  $x_2$  and  $y$  equal to one. In contrast, the following is a *nonlinear* equation because at least one of the terms is raised to an exponent greater than one:

$$x_1^2 + x_2 = y$$

Many times in applied problems, systems of equations are *nonlinear*. *Newton's method* is a classic algorithm used to solve nonlinear equations. For details, see Hubbard and Hubbard (2002).

## 2.51 Distance

Virtually all statistical techniques are organizations and re-organizations of the distance metric in some capacity or another. Even the Pearson correlation coefficient can be conceived as a measure of distance between points (we challenge the reader to think about this).

The most common metric for distance is that of *Euclidean distance* given by

$$d(O, P) = \sqrt{x_1^2 + x_2^2 + \cdots + x_p^2} \quad (2.111)$$

where  $d$  is the distance between origin  $O$  and  $P$ , where  $x_1^2$  through  $x_p^2$  are coordinates of observations. A strength of Euclidean distances is that they are *rotation-invariant*, which means that the straight-line distance between two points does not change with the orientation of the axis system (Aggarwal, 2015).

While Euclidean distance is what we typically think of when we consider distance, there are a host of definitions for distance, including *Minkowski*, *city-block (Manhattan)*, and many others. Distance metrics are especially useful in the technique of cluster analysis that seeks to generate *groups* (in some sense) of cases based on their similarity or distance from case to case. In the context of this and other methods, we will survey a number of distance measures throughout the book. Simple distances in Python can be computed using `numpy.linalg.norm(a-b)`.

## 2.52 Projections and Basis

The projection likely already most familiar to the reader will undoubtedly be that of linear regression, where we take the vector  $y$ , generate a linear function, and project onto the span of  $x$  resulting in a new line called the *regression line* containing predicted values,  $\hat{y}$ . The fit of the regression line to the swarm of points is essentially as a result of projecting  $y$  onto the space “inhabited” by  $x$ . The residual is  $y_i - \hat{y}$ .

When we speak of an *orthogonal projection*, this implies that for a subspace  $S$ ,  $y_i - \hat{y}$  is orthogonal to  $S$ . That is, the residual is perpendicular (orthogonal) to the space  $S$  “occupied” by the predictors (typically called the *column space*). Said differently, parameter estimates for the regression (which then generate  $\hat{y}$ ) are chosen so that the residual vector  $y_i - \hat{y}$  is orthogonal to the space “generated” by the predictors in the model (Hastie et al., 2009).

The orthogonality, coupled with the Pythagorean theorem, further implies that we can decompose the squared norms of these objects in the following manner,

$$\|y_i\|^2 = \|\hat{y}\|^2 + \|y_i - \hat{y}\|^2 \quad (2.112)$$

or more generally for arbitrary observed and fitted vectors where  $\hat{\mathbf{v}}$  is the more general fitted value of the vector and  $\hat{y}$  above only a special case:

$$\|\mathbf{v}\|^2 = \|\hat{\mathbf{v}}\|^2 + \|\mathbf{v} - \hat{\mathbf{v}}\|^2 \quad (2.113)$$

The above identities are important, as they essentially describe what occurs in regression and ANOVA; that is, the total sums of squares can be broken down into that due to treatment and that due to error (Marden, 2015, p. 89). The “treatment variation” is represented by  $\|\hat{\mathbf{v}}\|^2$  while the error is given by  $\|\mathbf{v} - \hat{\mathbf{v}}\|^2$ .

In *principal components analysis*, the goal will be to obtain a subspace onto which we also project observed data onto that subspace. The projected data are *principal component scores* and the new coordinates from which that data are projected are the new *basis vectors*. The original basis, or *canonical basis* as it is sometimes called, are the original coordinates at a right angle. For example, in  $\mathbb{R}^3$ , the standard basis vectors are the following:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The axes of the standard basis or any other basis need not be at right angles (Hubbard & Hubbard, 2002). That is, there is no *absolute* requirement that the above vectors  $\mathbf{e}_1$ ,  $\mathbf{e}_2$ ,  $\mathbf{e}_3$  be at right angles, *ontologically*. Historically, and for most problems, the decision to have bases at right angles is very much due to *convenience*. Descartes’ invention of the Cartesian coordinate system in the 17th century allowed algebra to be expressed geometrically where each axis was set perpendicular or orthogonal to one another. Pierre de Fermat (1607–1665) discovered the same coordinate system although his work was published only after his death, and hence Descartes is typically associated with the traditional coordinate system we know today. A mathematician by the name of John Wallis (1616–1703) was one of the first to conceive of the possibility of negative coordinates (Tanton, 2005). Not starting off with pairwise orthogonality of basis vectors would complicate problems tremendously and hence like many other things in mathematics, it is convenient to start with the assumption or constraint of orthogonality of canonical vectors in many cases. It is a convenient and natural starting point to many problems, and indeed, to much of mathematics in general.

The new basis vectors in the case of principal components are *eigenvectors* and possess favorable properties from which the projection is made. Hence, PCA is often regarded at a technical level as a *change of basis* methodology from which we project new scores. It is also generally regarded as a *dimension-reduction technique*, but this dimension reduction “happens” through a change of basis.

### 2.53 The Meaning of Linearity

The word *linear* is perhaps the most common word used in all statistics and machine learning. But what does linear mean, exactly? A linear transformation  $T$ , as already discussed, is a mapping from  $\mathbb{R}^n$  to  $\mathbb{R}^m$  such that the following holds for scalars  $a$  and vectors  $\mathbf{x}$  and  $\mathbf{y}$ :

$$T(\mathbf{x} + \mathbf{y}) = T(\mathbf{x}) + T(\mathbf{y}) \quad (2.114)$$

and

$$T(a\mathbf{x}) = aT(\mathbf{x}) \quad (2.115)$$

That is, linear transformations are said to *preserve* vector addition and scalar multiplication. Linear transformations, when viewed from a geometrical point of view, are an example of an *affine transformation*. Affine transformations and associated geometry also preserve such things as ratios of distances between points. For example, transforming degrees Celsius into degrees Fahrenheit is an example of an affine transformation because the relative change between say, 10 degrees and 20 degrees Celsius is preserved when we transform to Fahrenheit:

$$^{\circ}\text{C} = ((^{\circ}\text{F}) - 32)(5/9)$$

If one draws the corresponding plot, one will see that the relationship between degrees Celsius and Fahrenheit is indeed linear and the ratios of distances are preserved.

Linear transformations in R and Python are very simple to execute. For example, in Python, suppose we want to determine the aforementioned temperature in Celsius corresponding to a temperature of 32.0 Fahrenheit:

```
F = 32.0
(F-32)*(5/9)
Out [16]: 0.0
```

or, for a temperature of  $F = 0$ , we have

```
F = 0
(F-32)*(5/9)
Out [18]: -17.77777777777778
```

## 2.54 Basis and Dimension

We have already discussed informally the nature of a basis. We now put the concept on a formal footing. *A basis is a set of linearly independent vectors in a vector space that spans the space.* Conceptually, a basis essentially provides a “frame of reference” for vectors in a subspace. The dimension of a basis is the number of linearly independent vectors in the given set making up the basis. More formally, a basis is a *maximum linearly independent set* such that if we added one more vector, the set would cease to be linearly independent. That is, if there exists a basis with  $n$  vectors in a vector space, then there cannot be a basis with more than  $n$  vectors in that vector space (since by definition, a basis is the *maximum* linearly independent set). This last result is a consequence of the *Steinitz exchange lemma* (Stillwell, 2016). Equivalently, a basis is a minimal spanning set such that if we dropped a vector, it would no longer span the given subspace (Hubbard & Hubbard, 2002).

### 2.54.1 Orthogonal Basis

Recall that the canonical basis vectors of the coordinate system generate an *orthogonal basis*, and that since each of the vector's lengths is equal to 1, it is simultaneously an *orthonormal basis*. In many procedures, the goal is to generate an orthogonal basis from one that may not already be orthogonal. This is a very powerful idea that actually accounts for how many statistical methods operate “under the hood” so to speak.

The *Gram-Schmidt process* is an algorithm used in linear algebra for finding an orthogonal basis. The details of the procedure are not enlightening from an applied point of view. For mathematical details on the approach, see virtually any good introductory linear algebra text (e.g., Anton & Rorres, 2000). The Gram-Schmidt process implies that *if you have a finite inner product space, an orthonormal basis can be constructed in that space* (Garcia & Horn, 2017). Such a result will serve us well when we survey methods such as principal components analysis, where, as already mentioned, the goal will be to establish a new set of basis vectors with special properties and from which to project the observed data. Principal components analysis is essentially a (linear) transformation of the data onto a new basis. That is, it generates new coordinates from which to visualize and interpret the original data.

The very concept or general idea of a basis can also be made much simpler. Moving away from matrices and eigenvectors, and as a helpful imperfect analogy, consider the “basis” on which numbers are computed. For example, expressing the number 0.5 is relative to one basis (i.e., “frame of reference”), while expressing it as  $\frac{1}{2}$  is relative to another (Hubbard & Hubbard, 2002).

## 2.55 Review Exercises

1. Distinguish between a *vector* and a *vector space*. How is a vector different from a matrix?
2. Show that  $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$ .
3. Distinguish between the *Euclidean norm* and the *inner product*. How are they different? Similar?
4. Explain the essence of *eigen analysis* and why it forms the basis of dimension-reduction techniques such as *principal components analysis*.
5. Precisely define what is meant by a *linear transformation*.
6. In words, describe the nature of a *linear combination*. If you have studied statistical methods before, explain how linear combinations are at the heart of many statistical techniques such as the analysis of variance, regression, and others.
7. Show that a determinant equal to 0, geometrically, corresponds to an object with zero volume.
8. Show that  $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$ .
9. Show that the length of a vector is equal to the square root of its inner product.
10. Discuss the nature of the *kernel trick* and at least, intuitively, why it is so valued in seeking to work in higher dimensions.
11. Distinguish between a *basis* and an *orthogonal basis* and why the concept of basis is relevant to statistics, specifically dimension reduction.

## 2.56 Calculus and Optimization

Calculus is usually regarded as one of the most, if not the most, crowning intellectual achievements of the human race. Though presumably discovered independently by Newton and Leibnitz, as



Boyer (1959) has shown, the rise of calculus-like thinking had been going on for centuries prior to these two giants of mathematics putting the final touches on the concepts (mostly the *limit* concept) and developing notation. Optimization theory, a central part of statistics and machine learning, uses calculus and other mathematics to determine values that maximize or minimize functions and has a long history as well.

In this final section of our essential statistics and mathematics review, we survey the most important and salient elements of calculus and optimization theory and preview how these elements are used in statistics and machine learning.

## 2.57 Functions, Approximation, and Continuity

Much of statistics and machine learning can be regarded as an exercise in *approximating functions* or other mathematical objects. But what is an approximation? An approximation can be extremely simple; for instance,  $\pi = 22/7$  is an approximation to  $\pi$  just as differentiating and integrating in calculus, as we will see, are implementations of an approximation process. When speaking of approximations, however, we are only getting started. It doesn't take long to realize that virtually everything, or at least most things in statistics and machine learning feature approximations of some kind.

For example, when we compute an arithmetic mean on some data, such can be regarded as an approximation to the central tendency, in some sense, of the distribution. When we use that mean as an estimator of the population mean, we are again approximating, in this case, some parameter. The least-squares line or any other function fit to data is an approximation of the true underlying functional form. In real analysis, limits and convergence are once more examples of where approximation surfaces. For example,  $\lim_{x \rightarrow 5} f(x) = x + 3$  denotes a limit of 8, but that number only occurs *in the limit*. What the expression  $\lim_{x \rightarrow 5} f(x)$  is actually communicating is the idea of an *approximation*, that as  $x$  approaches 5, the value of the function goes to 8. We are converging on the value of 8 based on this limiting process.

Calculus is all about finding tangents to curves, the slope of which are called *derivatives*, and *integrals*, which are areas under curves found by integration. Recall that a function,  $f(x)$  is the set obtained by inputting  $x$  into the rule or argument. For a rule to be a function, for every value of  $x$  there must be only one value for  $f(x)$ , or  $y$ . The domain of a function is all inputs that are possible for the given function, while the *codomain* of the function consists of all theoretically produceable outputs (i.e., not only those potentially observed). For example, for the simple linear function  $f(x) = x$ , the domain and codomain can be defined as the set of real numbers, denoted  $\mathbb{R}$ .

Examples of functions from the physical sciences include such classic functions as Newton's law,  $F = ma$ , where  $a$  is the acceleration of a mass  $m$  subjected to a net external force  $F$ , and of course Einstein's theory of relativity,  $E = mc^2$ , where under a constant speed of light  $c^2$ , energy  $E$  is proportional to the mass  $m$  of the object.

### 2.57.1 Definition of Continuity

A function is *continuous* at a given point  $x = a$  if both the function is defined at that point and the limit of  $f(x)$  exists at that point, and both are equal. That is,

$$\lim_{x \rightarrow a} f(x) = f(a) \quad (2.116)$$

in which we recall again that the *limit* of a function is the number *approached* by that function, and not necessarily defined *at that function*. For example, in the sequence 0.9, 0.99, 0.999, the limit of the sequence is 1 regardless of whether the sequence ever attains 1. A sequence that has a limit is referred to as *convergent*. Limits can be computed quite easily in Python. For examples, see Saha (2015).

Elementary as we have portrayed continuity to be above, it is in fact a very deep concept and took years for mathematicians to develop and rigorize. In modern mathematics, continuity is usually conceptualized using epsilon-delta definitions (see Denis, 2021a for a brief and simple overview of how these definitions work). Of course, continuity is not first and foremost a mathematical idea. It is first and foremost a *concept*. The *idea* of continuity in the consciousness of human beings has, just as is true of many essential mathematical concepts, likely existed since the dawn of humanity. It is not difficult to imagine early civilizations planning travel routes based on the continuity of one route instead of taking an alternative route that would be *discontinuous* and have “gaps” or “jumps” in the route (perhaps a river that needed to be crossed). The fact that mathematicians and philosophers have defined these concepts *rigorously* using precise notation should not be taken to imply that such concepts were “born” in mathematics. If anything, like most things, they were likely born in early philosophical “nonspecialist” thinking.

## 2.58 The Derivative

The reader will recall from calculus that to differentiate a function,  $f(x)$ , we first compute the change in  $y$  ( $\delta y$ ) relative to the change in  $x$  ( $\delta x$ ):

$$\frac{\delta y}{\delta x} = \frac{f(x + \delta x) - f(x)}{\delta x} \quad (2.117)$$

The above is called the *difference quotient*, and represents the average rate of change of a function. To get the *derivative*, we compute the change in the function relative to the change in  $x$  as in (2.117) but with one very important addition; we take the change in  $y$  relative to the change in  $x$  *to the limit*, where now  $df$  corresponds to change in  $y$  (or change in the *function* more generally) and  $dx$  corresponds to change in  $x$ , represented now by  $h$  in the ratio (i.e.,  $h = \delta x$ ):

$$\frac{df}{dx} = f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \quad (2.118)$$

That is, the derivative  $f'(x)$  is the limit of the difference quotient as the change in  $x$ , represented by  $h$ , goes to zero. Be sure to note that the derivative is a *slope*, but it is only the slope in the limit as  $h \rightarrow 0$ . In other words, it's the *instantaneous rate of change*. Without this qualifier of “*limit as  $h \rightarrow 0$* ,” the derivative has no meaning and is not defined. Differential calculus (and integral, for that matter) is all about *limits*. The derivative is easily visualized in the graph in Figure 2.16 for two dimensions, where we see the tangent to the curve as well as an approximating secant line (to be discussed shortly).

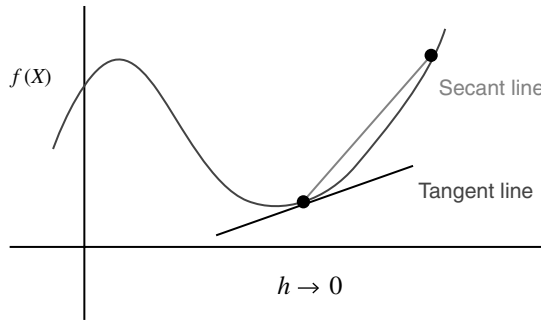


**Figure 2.16** Tangent to the curve coupled with the limit concept is the basis for the derivative. The derivative has multiple interpretations, one of which is the limit of the difference quotient as change in  $x$  goes to zero ( $h \rightarrow 0$ ). A second interpretation of the derivative is as the limit of secant lines as  $h \rightarrow 0$ . Both interpretations yield the tangent to the curve at the particular point of interest, where the derivative is the slope of the tangent at that particular point.

Note that we could have drawn a number of slopes around the given point and all would be slightly different from the slope in Figure 2.16. How is the *tangent* defined at that particular very small point? This is again where the concept of a limit comes in. For  $h > 0$ , we can fit many different secant lines (i.e., lines intersecting the curve closer and closer to the eventual tangent line). However, as  $h \rightarrow 0$ , we *zero in* on the tangent line, the slope of which we will call the *derivative*. That is, *the derivative is the slope defined by the limit of the secant lines as  $h \rightarrow 0$* . In this respect, the derivative is not simply the slope of the line *touching* the point at the given location; it is the *limiting (secant) line that touches the point*, even though informally we will often simply say “touching the point” or “tangent to the point” or “slope at the point” when describing the derivative.

It stands that if the derivative of a function is equal to 0 at each point along some interval of  $x$ , then this implies that the function is *constant* on that interval. That is, if  $f'(x) = 0$  at each point, then this implies  $f(x) = c$  at each point, where  $c$  is a constant. If a function does have a limit at a given point, that limit is *unique*. In higher dimensions, much more difficult to visualize, instead of a *tangent to the curve*, we will have a *plane tangent to the surface*. The vocabulary has changed but the concept is the same. Higher dimensions can be intimidating, but are less so if one has a good grounding in understanding concepts in lower dimensions.

Hubbard and Hubbard (2002, p. 126) summarize the goal of differential calculus beautifully, providing great insight into its purpose:

The object of differential calculus is to study nonlinear mappings by replacing them with linear transformations: we replace nonlinear equations with linear equations, curved surfaces by their tangent planes, and so on. Of course, this linearization is useful only if you understand linear objects reasonably well. Also, this replacement is only more or less justified. Locally, near the point of tangency, a curved surface may be very similar to its tangent plane, but further away it isn't. The hardest part of differential calculus is determining when replacing a nonlinear object by a linear one is justified ... nonlinear is much too broad a class to consider. Dividing mappings into linear and nonlinear is like dividing people into left-handed cello players and everyone else.

We can also distinguish derivatives into left and right. The left-derivative is the limit, not surprisingly, from the left,

$$f'_-(x) = \lim_{h \rightarrow 0^-} \frac{f(x+h) - f(x)}{h} \quad (2.119)$$

where the right-derivative is given by

$$f'_+(x) = \lim_{h \rightarrow 0^+} \frac{f(x+h) - f(x)}{h} \quad (2.220)$$

It can be shown that the derivative of a function at a point only exists if both the left and right derivatives exist and are equal to one another.

In Python, we can find derivatives of functions as follows:

```
from sympy import Symbol, Derivative
t = Symbol('t')
St = 5*t**2 + 2*t + 8
Derivative(St, t)
Out [68]:
Derivative(5*t**2 + 2*t + 8, t)

d = Derivative(St, t)
d.doit()

Out [70]:
10*t + 2
```

That is, the derivative of the above function,  $f(x) = 5t^2 + 2t + 8$  is, as expected using the power rule,  $f'(x) = 10t + 2$ , where the power rule is given by  $\frac{df}{dx}(x^n) = nx^{n-1}$ .

### 2.58.1 *Local Behavior and Approximation*

Much insight can be gleaned from gaining perspective on the derivative and its concept and can be used to indirectly foreshadow many of the methods in this book. First, note the obvious point that the derivative is not *equal* to the curve on which it is being computed. We can perceive the derivative as an *approximation* to the curve at that particular point. The function is nonlinear, but at that local level we can treat it as though it were linear. That is, *locally, or “zoomed in,” a nonlinear function behaves a lot like a linear function*. This idea of *locality* will return time and time again in this book. Some methods focus on more *global* structure, while others focus on more local behavior. As a preview of this distinction, polynomial regression is a global function computed on a curve, while regression splines, as just one example, model the data at a more local level.

Most of differential calculus is about establishing and using rules of differentiation to make it easier than using the above “long form.” Rules such as the *product rule*, *quotient rule*, *sum rule*, and *chain rule* are typically used and derived. Much of introductory calculus is in practicing using shorthand rules such as these to compute derivatives in a variety of contexts. We assume the reader is at least somewhat familiar with those rules or has at least at one point in their past come across them. It is not essential to be able to perform them at the drop of a hat, since we can rest assured that computers will “know” the correct rule (if the shorthand is used at all, computationally) to implement and compute the derivative that we need. What *is* essential is knowing what a derivative *is* and that there are numerous rules, all formally provable, to obtaining derivatives.

Though we dispense from any in-depth discussion of rules of differentiation, we will mention one rule, however, that is especially important and relevant for our purposes since it is used in neural networks, in the *backpropagation algorithm* more specifically, and that is the *chain rule of differentiation*. For a function  $y = m(x) = f[g(x)]$ , the chain rule is defined,

$$m'(x) = f'[g(x)]g'(x) \quad (2.221)$$

The chain rule deals with *composite functions*; we are taking a *function of a function*, that is,  $f'[g(x)]$  is the function  $f()$  of the function  $g()$ . The function  $g$  on the inside is referred to as the *core* or *inner function* and is applied second to the outer function. The above is for two functions, but we can generalize to a greater number of functions. It can also be shown that the composition of two continuous functions is itself continuous.

As we will see later in the book when we study neural networks, understanding backpropagation is as simple, in a “big picture” sense, as understanding the chain rule. Indeed, most continuous optimization methods employ differential calculus. The reader should know that not all functions are differentiable even if the given family of functions one is working with is continuous. For example, the absolute value function is continuous everywhere yet not differentiable at the “cusp” of the function (i.e., the bottom of the function with the sharp edge).

### 2.58.2 Composite Functions and Basis Expansions

In discussing composite functions, in addition to previewing the idea of a neural network, is an opportune time to highlight and forecast another theme that will reappear throughout this book, and that is the idea of *basis expansions*, of which, as we will see, neural networks can be conceived as a special case of such an expansion. Consider the following:

$$f(x) = \sum_{i=1}^M \beta_m h_m(x)$$

In the above,  $h_m$  is called a *basis function*. Colloquially, it is providing a “basis” for  $x$  before we weight  $h_m(x)$  by estimated weights,  $\beta_m$ . How is this idea linked to that of composite functions? Well, consider that  $h_m$  is taking a function of  $x$  and  $\beta_m$  is then applied to this inner (or “core”) function. This process can be thought of, in a very general sense, as taking a *function of a function*, which by definition is the very idea of a composite function.

Note that when  $h_m = 1$ ,

$$\begin{aligned} f(x) &= \sum_{i=1}^M \beta_m h_m(x) \\ &= \sum_{i=1}^M \beta_m (1)(x) \\ &= \sum_{i=1}^M \beta_m x \end{aligned}$$

we obtain the classic regression model-style weighting. The key point for now is to recognize that the basis function is applied to the observations in  $x$  before the relevant weighting by  $\beta_m$  takes place. In theory, there is practically no limit to how we can designate  $h_m$  to transform  $x$  before relevant weighting by  $\beta_m$ . As we will see throughout the book, basis functions are one of the themes

that unify a great deal of statistical and machine learning models and we will revisit these ideas often as we survey novel methods. *Understanding basis functions is a powerful way to understand the “forest” of many statistical and machine learning models and methods.*

## 2.59 The Partial Derivative

While single-valued derivatives featured earlier explicate the concepts of differentiation, in practice, we are usually more interested in computing *partial derivatives* of functions. A partial derivative is the derivative of a function while holding other variables *constant*. The collection of partial derivatives for a multivariable function is typically referred to as the *gradient*, though the word “gradient” is also commonly used to denote the slope between two variables. One has to decipher the meaning based on the context of the situation to know what is being referred to.

The partial derivative of a function  $f$  with respect to  $x_1$  holding all other variables,  $x_2, x_3, \dots, x_n$  constant is given by

$$\frac{\partial f}{\partial x_1} = \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(\mathbf{x})}{h} \quad (2.222)$$

where  $\partial x_1$  denotes the variable about which we are taking the partial derivative. All other variables are held constant. As was the case for the derivative for single variables, the partial derivative only becomes such as  $h \rightarrow 0$ , that is, in the limit.

Since there will necessarily be a “collection” of these derivatives, it is convenient to group them in a row vector,

$$\nabla_x f = \text{grad} f = \frac{df}{d\mathbf{x}} = \left[ \frac{\partial f(\mathbf{x})}{\partial x_1} \quad \frac{\partial f(\mathbf{x})}{\partial x_2} \quad \dots \quad \frac{\partial f(\mathbf{x})}{\partial x_n} \right] \quad (2.223)$$

where such a vector is typically referred to as the *gradient* of the function and the *Jacobian* the vector of first-order partial derivatives of the function (Deisenroth et al., 2020).

## 2.60 Optimization and Gradients

As we will learn throughout this book, many statistical and machine learning methods, complex as they may at first glance appear, center around the general problem of *optimization*, which loosely translates to finding “best” weights for a given function, from simple to more complex, within a structural framework or *model*. To arrive at these weights, a designated function, which will depend on the particular method, will typically need to be *maximized* or *minimized*. The classic and most familiar example is that of least-squares where we wish to minimize the sum of squared errors. The general principle is that we are minimizing an *objective function*, the objective function in this case being the sum of squared deviations or errors from the mean. Of course, sum of squared errors is by no means the only objective function we may wish to minimize, and hence optimization is a very general idea applicable, at least in theory, to a whole host of functions. That is, the loss  $L_2(y, y') = |y - y'|^2$  is a special case of the more general  $L_p$  loss defined by

$$L_p(y, y') = |y - y'|^p \quad (2.224)$$

where  $p$  is a number equal to or greater than 1. Objective functions are often referred to as *cost*, *loss*, or *error* functions when they are minimized (Goodfellow et al., 2016).

A technique called *gradient descent* and derivations and extensions of it (soon to be discussed) is used, in part, because closed-form analytical solutions do not always exist or are not preferable or practical to those provided by gradient descent. The origins of gradient descent are usually ascribed to Cauchy (1847). Recall that a *closed-form* or *analytical solution* is an algebraic-type solution based on the solution of equations. Often, these types of “clean” solutions do not exist and we are instead required to use iterative numerical approaches to approximate solutions. Thanks to the computer, iterating can be done in a matter of seconds whereas in the past, even if the relevant algorithm had been invented, it was prohibitive to implement due to a lack of computing power.

For example, early maximum likelihood iterative algorithms were developed as early as 1912 (Fisher, 1922) but as was true of many computations back then, were difficult to implement due to the computational complexity of carrying out such procedures. This is precisely why computation is so important to modern algorithms and why the *computational capacity* of various algorithms is a field in and of itself and a primary research area of computer science. Gradient descent itself is actually a fairly old algorithm, but it has become increasingly popular relatively recently because of the advances in computing capacity. It would not be unreasonable to conjecture that many of the computational challenges experienced today will seem trivial even a decade or two from now. Computer scientists and engineers earn their accolades!

### 2.60.1 What Does “Optimal” Mean?

Before considering further how gradient descent works, it is well worth taking a moment to discuss what “optimal” means, since the goal of the typical algorithm will be to search for such an optimal solution. *Optimal implies that the weights (or more generally, solutions) are arrived at in such a way that they usually satisfy one or more conditions or constraints.* Again, a prime example of this that the reader will undoubtedly recognize is that of ordinary least-squares in classical regression. Recall the simple linear regression model:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Recall that the goal is to estimate weights  $\beta_0$  and  $\beta_1$ . What values for these parameters are considered “best” or “optimal?” We know from least-squares theory that they will be chosen such that they minimize the sum of squared errors around the regression line. That is, the “optimal” values are those that satisfy the least-squares conditions.

More generally and formally, we can say that many statistical methods aim to estimate parameters that minimize a *loss function*:

$$\hat{\theta}_{mle} = \arg \min_{\theta} \ell(\theta) \quad (2.225)$$

where  $\ell(\theta)$  is the loss function for the given problem and  $\hat{\theta}$  the maximum likelihood estimate of the parameter vector. In our example with least-squares, the loss function was the sum of squared errors. Maximum likelihood, as we have seen, is a popular method of parameter estimation that selects the parameters that essentially give the data the highest probability, that is,

$$\hat{\theta}_{mle} = \arg \max_{\theta} p(D | \theta) \quad (2.226)$$

The take-home message here is that “optimal” usually involves the idea of *constraints* or *conditions*, or more generally, some kind of *restrictions* before computing values, even if those conditions are implicit in the algorithm itself. In maximum likelihood, only those parameters  $\theta$  are selected that maximize the likelihood of  $D$ ; the parameter values are not chosen *unconditionally*. They are chosen in line with how maximum likelihood defines the “optimal” parameter.

As an analogy, consider the most optimal way to get home from your job. It is likely the distance which features the shortest *distance* and *time*, but in winter, going home must be subject to the constraint that the route is safe and free of ice. *Constraints and conditions are all around us, rule our activities, human relationships, and likewise appear in the minimization or maximization of functions in mathematics, statistics, and machine learning.*

### 2.60.2 Minima and Maxima via Calculus

As the reader may recall from calculus, finding *maxima* or *minima* of functions usually involves setting derivatives equal to 0 to determine at which point(s) on the function the slope is equal to 0. For instance, consider the function in Figure 2.17.

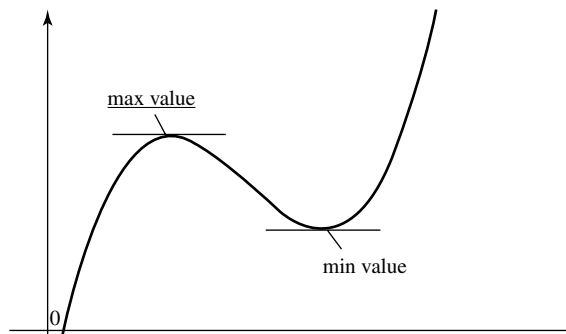


Figure 2.17 Function having a minimum and maximum value represented by derivatives equaling 0.

How can we know that the maximum value of the function occurs where it does? How about the minimum? We can tell this because at those locations, *the derivative is equal to 0*, and a derivative equal to 0 implies that the slope at that given point is equal to 0, since the derivative is the slope (or more precisely, the limit of secant lines approximating the function at the given point). This holds true for minima as well. That is, in general, for a differentiable function on an open interval  $(a, b)$ , if  $c$  is a maximum or minimum of the function, then this implies that  $f'(c) = 0$ . This result is implied by *Rolle's theorem* in analysis which essentially holds that if a function  $f$  is continuous on an interval  $[a, b]$  and  $f$  is differentiable on that interval  $(a, b)$  and that  $f(a) = f(b)$ , then there exists at least one point  $c$  between  $(a, b)$  such that the derivative is equal to 0, that is,  $f'(c) = 0$ . Rolle's theorem is a special case of the *mean value theorem*. For details, see Bartle and Sherbert (2011). Hence, when we set  $f'(x) = 0$ , we can determine the locations where the slope is equal to 0. A *critical point* of an objective function corresponds to parameter estimates that cause the derivative of the objective function to vanish at that point.

Figure 2.18 is a visual representation of the concepts of *global maximum* and *global minimum*, along with the concept of a *local minimum*. A global minimum or maximum applies to the entire function, whereas a local minimum or maximum is such only in a particular location of the curve; that is, if the minimum or maximum is local, then it is entirely possible for there to exist other minima or maxima elsewhere on the curve.



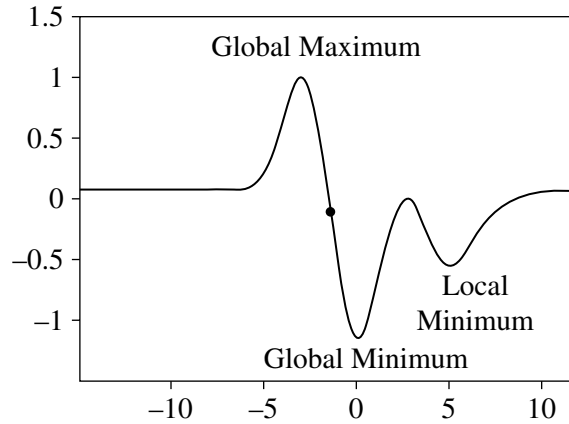


Figure 2.18 Global and local minima and maxima.

### 2.60.3 Convex vs. Nonconvex Functions and Sets

Establishing minima and maxima depends on the convexity of the function. Convex functions are preferred over nonconvex functions since they have *global* minima, while nonconvex functions often have several *local* minima, which can detract from finding a global minimum. Optimization problems that feature local minima can easily get caught in locality, or more colloquially, get “stuck” in local valleys.

Figure 2.19 gives a geometric interpretation of convexity. A convex function is one in which we can draw a line between line segments (as shown in the image) and the value of the function is guaranteed to be at or less than the line. For a nonconvex function, we cannot draw such a line without going through other segments of the function. Notice that for convex functions, there will be a global minimum. Another way of saying this is that for convex functions, any local minimum will also be a global minimum (Goodfellow, 2020). For nonconvex functions, there may be many local minima.

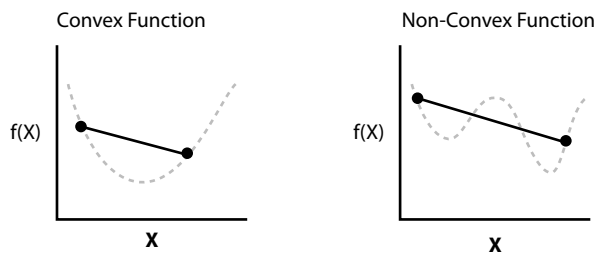


Figure 2.19 Convex vs. nonconvex functions. A convex function has a global minimum, whereas a nonconvex function may have many local minima.

When we discuss manifold learning later in the book, generalized concepts of line segments or paths that are the shortest distances between two points and can help indicate convexity in a manifold will be called *geodesics*. Geodesics, as we will see, however, are typically not unique. A

function defined on a manifold is convex if its *Hessian* is positive semidefinite, where the Hessian describes the local curvature of a function of many variables in terms of their second-order partial derivatives. Hence, when the Hessian is positive definite, this implies convexity of the surface we are working on, which will allow algorithms such as gradient descent to more easily perform optimization as a result of there being an assured global minimum.

## 2.61 Gradient Descent

Gradient descent is a popular algorithm used for minimizing an objective function for seeking global or local minima in many statistical and machine learning applications. It is the most common method of optimization used, for instance, in neural networks, but also many other machine learning and statistical approaches (Ruder, 2016). It is an *iterative* method where the goal is to minimize some arbitrary cost function which will differ by statistical or machine learning method employed. Iterations that perform slight changes to parameter estimates are stopped once the algorithm converges to a minimum value of a cost function, after which time no further reduction in the cost function is desirable or achievable. That's the goal: to perform slight iterative "updates" to parameter estimates until a local minimum is achieved. Though there are many varieties of gradient descent, there are generally three most popular: *batch gradient descent*, *stochastic gradient descent*, and *mini-batch gradient descent*.

### 2.61.1 How Does Gradient Descent Find Minima?

In finding minima of a cost or error function  $E(\theta_0)$  subject to updates in the model's parameters contained in the vector  $\theta_0$ , successive steps are taken that are proportional to the negative of the gradient of the function for the given current values of the model parameters. Note carefully what this means: we start off at some, often arbitrary parameter values,  $\theta_0$ , then based on the gradient (i.e., crudely "*slope*," but is actually the vector of partial derivatives) of the function, move toward a minimum in steps. We are seeking a minimum value and move in the direction of the *negative gradient*, that is,

$$\theta_1 = \theta_0 - \eta \cdot \nabla_{\theta} E(\theta_0) \quad (2.227)$$

where  $\eta$  is the step-size or *learning rate* and  $\nabla_{\theta} E(\theta_0)$  is the gradient vector which contains the entire set of partial derivatives of the error or cost function  $E(\theta_0)$ . Hence, for any value  $\eta \geq 0$ ,  $E(\theta_1) \leq E(\theta_0)$ . That is, for any value of  $\eta$  greater than 0, the function at  $\theta_1$  (i.e., the updated parameter vector) represents a *decrease* from the original cost function evaluated at  $\theta_0$ . In doing so, we are in the process of *converging* to a minimum value. The set of parameters  $\theta_0$  is optimized according to the step-size  $\eta$ , such that the new "guess" of the parameters,  $\theta_{0+1} = \theta_1$  is arrived at by this method of minimization.

Visually, gradient descent can be depicted as in Figure 2.20, where we see we are eventually converging on a local or global minimum toward the center and thus minimizing the cost or error function by iteratively updating parameter estimates with each iteration.

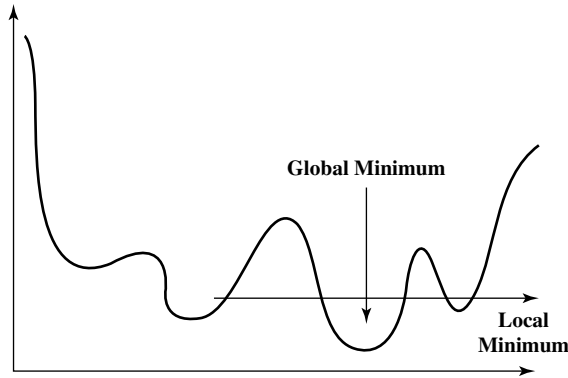


Figure 2.20 Concept behind gradient descent; successive iterations eventually lead to minimizing the error or cost function.

What we have described thus far is generally known as “vanilla” batch gradient descent (Bishop, 2006; Ruder, 2017). Gradient descent can be relatively slow close to the minimum value. Its asymptotic rate of convergence can also be inferior to other methods (Deisenroth et al., 2020). If the learning rate  $\eta$  is too fast, the true local minima could be “skipped” as the algorithm tries instead to maximize the shortest time it searches for such minima. On the opposite extreme, if the learning rate  $\eta$  is too slow, gradient descent may never converge as the algorithm tries “too hard” to find the local minima. Hence, choosing an appropriate step-size or learning rate  $\eta$  is important. There is a kind of “trade-off” in gradient descent between the accuracy of the parameter update vs. the time it takes to make such an update (Ruder, 2016). One useful strategy is to have a changing learning rate that slows down as the minimum is neared, commonly known as *annealing* (Ruder, 2016). Of the varieties of gradient descent, batch gradient descent is guaranteed to converge to a *global minimum* so long as the error surface is *convex*; if it is not convex, then it is guaranteed to converge to a *local minimum*.

*Stochastic gradient descent* performs a parameter update for *each* training data point,  $x_i$ , not a “batch” such as in batch gradient descent,

$$\theta_1 = \theta_0 - \eta \cdot \nabla_{\theta} E(\theta; x_i) \quad (2.228)$$

where  $x_i$  is the data point at a given iteration. Stochastic gradient descent improves on the redundancy weakness of batch gradient descent and is therefore much faster, and since it is only required that the gradient be an unbiased estimator of the true gradient, it is not absolutely required to use the full training data. Because of the frequent updates, it can cause rather high variance as the algorithm attempts to arrive at a minimum. Stochastic gradient descent can overshoot the minimum for too large of a learning rate; hence reducing the learning rate is often advised. The algorithm requires less computing resources as well compared to gradient descent for the reason that it does not use all the data at once, which for large-scale problems could have major benefits over gradient descent. Indeed, stochastic gradient descent has been found to be very effective for large-scale machine learning (Bottou et al., 2018) and is essentially the “workhorse” of most modern machine learning applications. Finally, stochastic gradient descent when applied to nonconvex functions has no guarantee of convergence and can be very sensitive to initial parameter values (Goodfellow et al., 2016).

*Mini-batch gradient descent* is often seen as a compromise between batch and stochastic methods such that the algorithm performs an update for each “mini-batch” of  $n$  training examples,

$$\theta_1 = \theta_0 - \eta \cdot \nabla_{\theta} E(\theta; x^{(i:i+n)}) \quad (2.229)$$

where now  $i+n$  is the new “mini-batch” added in the updating of  $\theta_0$  to  $\theta_1$ . Mini-batch, however, does not always guarantee good convergence and selecting a good learning rate can be challenging.

*Gradient descent with momentum* is yet another derivative of gradient descent that can help accelerate stochastic gradient descent. This and other modifications and adaptations to gradient descent are summarized in Ruder (2016).

## 2.62 Integral Calculus

Whereas differentiation is about finding the slope of tangents to curves, integral calculus is about finding *areas* under curves. Historically, finding areas of such things as rectangles has been known since the Greeks, but finding areas under more general shapes and curves would take a long time to figure out. Early attempts at computing such areas included that of Eudoxus of Cnidus (400–350 B.C.E.), who attempted to compute areas under curves by successively small rectangles, as illustrated in Figure 2.21.

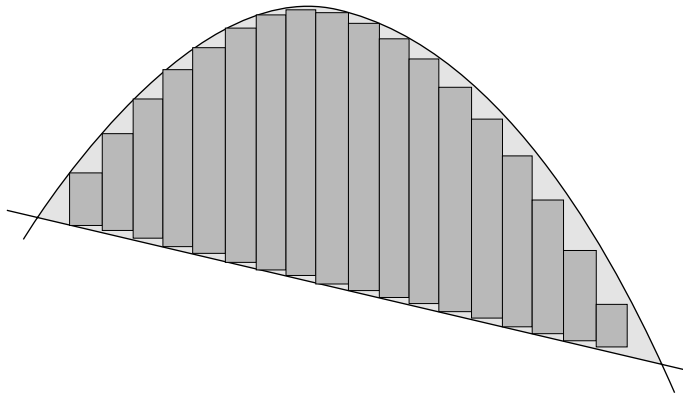


*Figure 2.21* The method of exhaustion of Eudoxus of Cnidus (400–350 B.C.E.) where successively small rectangles are used to approximate area under a curve.

This approach was known as the *method of exhaustion*. Centuries of thinking about the problem culminated with Newton and Leibniz’ (and many others) contribution of the calculus and the integral, where now we could find areas under curves using a more sophisticated method of summation, called the integral (Figure 2.22).

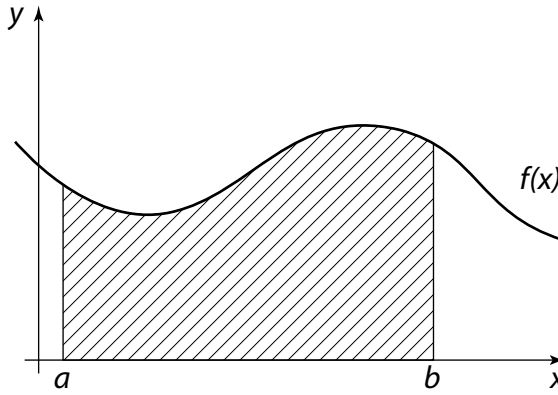


Figure 2.22 The calculation of area under the curve using integral calculus.

But how do we go about finding the areas under such “complex” curves? For small changes in  $x$ ,  $\Delta x$ , the reader may recall that we can sum these areas, called *Riemann sums*:

$$\sum_{i=1}^n f(x_i) \Delta x \quad (2.230)$$

As  $n \rightarrow \infty$ , the definite integral from  $a$  to  $b$  is obtained in the limit:

$$\int_a^b f(x) dx = \lim_{n \rightarrow \infty} \sum_{i=1}^n f(x_i) \Delta x \quad (2.231)$$

The integral of everyday calculus is known as the *Riemann integral* due to Bernhard Riemann (1826–1866). The Lebesgue integral followed which was applicable to a wider class of functions than the Riemann integral.

Integration is also generally known as *antidifferentiation* for the reason that integration is the reverse process of differentiation (i.e., integration “undoes” the differentiation). That is, if one is given a derivative and wants to know the original function of that differentiated function, one is seeking the antiderivative, which is arrived at through integration. The *fundamental theorem of calculus* encapsulates this idea that one technique is the reverse of the other. As was true for differentiation, not all functions are integrable. In brief, for a function to be integrable, it must be possible to make smaller and smaller pieces of area or volume, and increase the number of those smaller and smaller pieces. Most importantly, it can be shown that every polynomial is integrable over any set of finite volume (since polynomials are *continuous* functions), and hence for many to most functions dealt with in statistics and machine learning, integrability is not a concern. A deeper discussion into integral calculus would bring us into *measure theory*, which formally accounts for conditions of integrability. Much of modern statistics is based in one way or another on foundations in measure theory.

**2.62.1 Double and Triple Integrals**

Of course, in the multivariable domain, the aforementioned definitions will not suffice, and we need to extend these concepts to multiple dimensions. The double integral is given by

$$\iint f(x, y) dx dy$$

where, instead of integrating with respect to one variable, we are integrating with respect to two ( $x$  and  $y$ ). The *bivariate density*, for example, featuring the relationship between two variables requires the double integral, as we will see later in the book.

The next step up from a double integral is a triple integral, useful for integrating now in a three-dimensional space, once more corresponding to the volume of the space but in higher dimensions than the dimensionality previously discussed,

$$\iiint f(x, y, z) dx dy dz$$

Calculus of several variables, whether with regard to differentiation or integration, makes up the subject of *multivariable calculus*. A good reference for single and multivariable calculus is that of Salas et al. (1998).

**2.63 Review Exercises**

1. Describe the nature of a *derivative* in differential calculus. Why is the *limit* concept necessary to understanding the derivative?
2. In Python, compute the derivative of the function  $f(x) = 2x^2 + 10x$ .
3. Discuss the nature of a *composite function* in calculus.
4. What is a *basis function*? Write out a regression equation featuring the use of a basis function.
5. How is a *partial derivative* different from a single-valued derivative?
6. Conduct an online search and distinguish between a *Taylor series* and a *Taylor polynomial*. What function do these serve?
7. Describe how calculus can be used to learn of the *minima* and *maxima* of functions.
8. What is the role of the *learning rate* in seeking an updated value for parameter  $\theta_0$  (where  $\theta_1$  is the updated parameter) in gradient descent?
9. Contrast *integral* vs. *differential* calculus. The *fundamental theorem of calculus* relates the two procedures. Brainstorm how the two might be related.
10. Why is the *limit* concept central to integral calculus? Could we still compute areas under curves without the concept of a limit? How so? Explain.

## 3 R and Python Software

The rise of computers and computing power has revolutionized statistics and data analysis since at least the early 1980s, if not before. What used to take statisticians and scientists *days* to compute now takes, quite literally, *milliseconds*. For instance, computing a factor analysis in the 1930s took several *hours*, not seconds. Today, one can press a few button clicks (e.g., using Statistical Package for the Social Sciences (SPSS), for instance) and obtain factor-analytic output in less than a second. While this obviously has advantages, it also comes with it the potential peril of possibly overlooking important details prior to conducting data analysis, such as the integrity of data, good theorizing, etc., at least as it pertains to scientific fields. Computing can be done so quickly now that it can be all too easy to pay insufficient attention to *what is being computed and what is being measured*.

In business and healthcare analytics, for instance, the power of software has made it possible to analyze humongous databases and see patterns in data that were not possible before advanced computation and visualization. A business analyst can analyze terabytes of data relatively quickly to see emerging trends and advise management where to allocate marketing resources to maximize profits. As emphasized at the outset of this book, however, none of this awesome ability precludes the importance of ensuring the proper measurement and quality of data. Analyzing and concluding patterns from data only makes good sense if the data itself is of good quality and is tapping into the sought-after constructs. Poor measurement, or otherwise sketchy data can only mislead. Do people approve of your product? A rating scale of 1 through 5 suggests most folks may choose 4 or 5. However, that limited data does not tell you whether they approve of other products a lot more than yours. For instance, perhaps your product is at the lowest preference point of competing products, but you would never know it by simply asking a single question of how much people approve of your product. Hence, knowing what you are measuring and putting it into proper context is just as, if not more, important than before “big data” arrived on the scene. Big data equals big conclusions, which leads to big decisions, which may be faulty if not carefully considered because of lightning-fast computing speed and facility.

### 3.1. The Dominance of R and Python

While there are many software packages (e.g., SAS, Matlab, etc.) useful for data analysis, data science, statistics, and machine learning, two of the more popular programs are R and Python. R is probably the more popular in *statistics-based* coursework and applications, whereas Python is more frequently used in computer science departments as a general computing language. Both have their strengths and weaknesses. Other programs that have been popular in analyzing data include SAS and SPSS. Both are excellent for what they do but are limited since they do not have as much flexibility as R and Python (however, SAS is generally far more capable than SPSS). In R

and Python you can be more creative and program new things as needed than you can in programs such as SPSS and SAS. What is great about SPSS (and Excel), however, is the spreadsheet interface which makes it exceedingly convenient for quick or common data tasks. One should not fall prey to the “cool factor” that one uses complex software to do simple data analytic tasks that can be accomplished even faster with simpler software! The benefit, however, of using complex software such as R or Python even for simple tasks is that when jobs do get more complex (and they always do), you already will have your foot in the door. The universe of software is vast. For instance, packages such as *Theano* (Bergstra et al., 2010) in Python, *Caffe* (Jia et al., 2014) and *Torch* (Collobert et al., 2010) are other libraries and software useful for deep learning neural networks.

In this chapter, we provide a very (*very*) brief “sketchy” and trivial overview of R and Python software, some of their capabilities and options, along with a few simple demonstrations of what they can do. We move rather quickly in our introduction as we assume the reader has had at least *some* exposure to computing and can “pick things up” rather quickly in this regard. The capabilities of these software packages are immense and we could easily fill up a few volumes with examples of what software can do, but such would detract from the primary focus of the book, which is about multivariate statistics and machine learning, not *primarily* about computing. While computing and programming can of course get very deep and require immense talent, training, and experience, we do not see computing as something “to learn” in this book. That is, we do not see memorizing functions in R or Python as time well spent. We encourage the reader to focus more on *understanding* what functions accomplish.

All of the functions we discuss in this book and offered by R and Python can be looked up in references and otherwise searched in online manuals as needed. Much of our code is adapted in one way or another from existing packages in R or from Python’s site. Often, when working on code in a research setting, even sample code must be adjusted and customized to fit the situation at hand, and hence the most the student of these methods can hope for is some familiarity and practice with what these packages can do to prepare themselves for situations where they will have to work with code, not simply retrieve something from memory. Then, when the time comes to implement a given model or compute something, working through code, tweaking, and hacking will not be something “new” to them. A good course in computing teaches students the *capacity* to compute so that they gain experience, not necessarily *what* to compute in every conceivable instance of what *might* need to be computed over one’s career. Experience and practice in problem-solving are what are most important to prepare for novel situations. Need to do a *recursive neural net* in Python? Finding the code is not hard. The code however will not tell you what a neural net *is*; only a good book on theory will do that. Always be mindful of the following distinction:

*As a computer scientist, you wish to truly understand the underworkings of the code. As a scientist, you need to understand what code works, and generally, why it works and possibly how to adapt it, but you do not necessarily require understanding such code at a deep “language” level.*

Of course, the more one understands about the specific language, the better a programmer he or she will be and the greater power and flexibility one will have with regard to applications. But we do not venture into that territory in this book. At the conclusion of the chapter we provide a couple recommendations and resources for those who wish to pursue their computing knowledge of either R or Python, or both, and perhaps specialize in one software or the other (or both).

### 3.2 The R Project

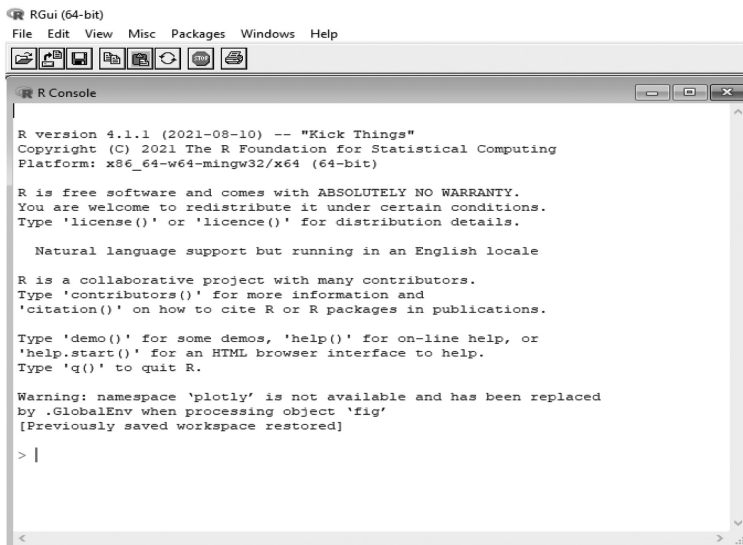
R is based on the S language that was developed in the 1970s by Bell Laboratories. R was released under a GNU public license in the 1990s, and since then has been developed extensively through its



many libraries and packages. Not only is R relatively easy to use (but has a relatively steep learning curve), it is the *packages* and *libraries* that makes it so useful. Literally, the best statistics and computing minds on the planet are daily adding and revising packages for R that can be downloaded and implemented by the user. These packages can conduct very specific functions on data, and in many cases, functions that preprogrammed packages (such as SPSS) are incapable of performing. In this way then, R is “live” on the internet and is being constantly updated. While at first this benefit may not seem that important, as you increasingly work with data, you will likely come to appreciate it. Many, maybe even most analyses and problems that you come across will call for solutions that deviate from clean “textbook” scripts or problems and in no time you will be searching for a package that may be uniquely suited for the data you have to analyze. This is the norm even if you already have extensive training in statistics or computing. *Each data problem and data set is different and may require a unique solution.* As a consultant in statistics, I have found that problems can get complicated very quickly and that routine text and code usually have to be edited and adapted in some way or another to best address the specific problem at hand, so much so that each problem worked on might eventually seem like an *exception* to the safe and predictable textbook-type problems students are used to. Those who teach “cookbook” statistics may not be aware that many problems need relatively customized answers, for which the cookbook approach does no service to students in understanding and appreciating the nature and complexities of statistical modeling.

### 3.2.1 Installing R

The latest version of R can be installed by going to the R Project website: [www.r-project.org](http://www.r-project.org). When you open R, you will arrive at the following screen:



The prompt can be adjusted and named whatever you like. For instance, we can name it “R” via the `options` command, or virtually anything else we would like, such as “Hello”:

```
> options(prompt="R> ")
R>
```

```
R> options(prompt="Hello> ")
Hello>
```

Many times when programming, we wish to include comments to help keep track of what we are doing. Comments can be indicated by “#”. For example, consider the following, where we square the number 5 and include a comment to describe what we are doing:

```
> 5^2 # here, we are raising 5 to the second power
[1] 25
```

Note that the appended comment is not reproduced; only the computation is. One can obtain help on a topic in R by coding `help()`. For instance, suppose we wanted help on computing means:

```
> help(mean)
```

The above will bring up an online page (not shown) with R functions available on the arithmetic mean. One can also simply code `?mean` to get help as well. Coding `??mean` will bring up more options as well. One can also get an example of an object via the `example()` function. For instance, suppose we wanted an example of computing the arithmetic mean:

```
> example(mean)
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

Quitting R can be done by `q()`.

### 3.2.2 *Working with Data*

Of course, we wish to work with data! Here, we survey a bit on how to examine a data set in R. Below are the first few observations of the infamous *iris* data set that we will use throughout the book. The *iris* data appeared in a paper by Fisher (1936) titled *The Use of Multiple Measurements in Taxonomic Problems* and contained three plant species (*setosa*, *virginica*, and *versicolor*) and four features measured on each sample, which were *sepal length* and *width* and *petal length* and *width*. The function `head()` will output the top few cases of the data set:

```
> data(iris)
> head(iris)
```

|   | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 2 | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |
| 3 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |
| 4 | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| 5 | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 6 | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |

To see the last few observations in the data, we code `tail()`:

```
> tail(iris)
```

|     | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species   |
|-----|--------------|-------------|--------------|-------------|-----------|
| 145 | 6.7          | 3.3         | 5.7          | 2.5         | virginica |
| 146 | 6.7          | 3.0         | 5.2          | 2.3         | virginica |
| 147 | 6.3          | 2.5         | 5.0          | 1.9         | virginica |
| 148 | 6.5          | 3.0         | 5.2          | 2.0         | virginica |
| 149 | 6.2          | 3.4         | 5.4          | 2.3         | virginica |
| 150 | 5.9          | 3.0         | 5.1          | 1.8         | virginica |

The `names()` function will reveal the variable names in the data set:

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

The `summary()` function generates some descriptive statistics on the iris data, including quartiles, means, medians, minima, and maxima:

```
> summary(iris)
```

| Sepal.  | Length | Sepal.  | Width  | Petal.  | Length | Petal.  | Width  |
|---------|--------|---------|--------|---------|--------|---------|--------|
| Min.    | :4.300 | Min.    | :2.000 | Min.    | :1.000 | Min.    | :0.100 |
| 1st Qu. | :5.100 | 1st Qu. | :2.800 | 1st Qu. | :1.600 | 1st Qu. | :0.300 |
| Median  | :5.800 | Median  | :3.000 | Median  | :4.350 | Median  | :1.300 |
| Mean    | :5.843 | Mean    | :3.057 | Mean    | :3.758 | Mean    | :1.199 |
| 3rd Qu. | :6.400 | 3rd Qu. | :3.300 | 3rd Qu. | :5.100 | 3rd Qu. | :1.800 |
| Max.    | :7.900 | Max.    | :4.400 | Max.    | :6.900 | Max.    | :2.500 |

|            | Species |
|------------|---------|
| setosa     | :50     |
| versicolor | :50     |
| virginica  | :50     |

To obtain data for sepal width specifically, we can request the second variable (`[2]`) in the data set:

```
> iris[2]
```

|   | Sepal.Width |
|---|-------------|
| 1 | 3.5         |
| 2 | 3.0         |
| 3 | 3.2         |
| 4 | 3.1         |
| 5 | 3.6         |

If we would like the data horizontal instead of vertical, we code double brackets:

```
> iris[[2]]
[1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4
```

Data can be easily sorted in R using the `sort()` function, either ascending or descending. For example, the following sorts `x` from smallest to largest value:

```
> x <- c(0, 4, 2, 5)
> sort(x)
[1] 0 2 4 5
```

### 3.2.2.1 Building a Data Frame

Most data sets we deal with in this book are already part of R or are contained in a specific package and hence easily loaded into R. However, we can easily also build a data frame directly in R using the `data.frame()` function. In the following, we build a data frame on three individuals, recording their age and sex:

```
> mydata <- data.frame(subject=c("Bob", "John", "Louis"),
+ age=c(20, 25, 30),
+ sex=factor(c("M", "M", "M")))
> mydata
```

|   | subject | age | sex |
|---|---------|-----|-----|
| 1 | Bob     | 20  | M   |
| 2 | John    | 25  | M   |
| 3 | Louis   | 30  | M   |

### 3.2.3 Installing Packages in R

As mentioned, packages are one of the highlights of R software as they allow one to implement specific functions prepared by experts in the area and are continuously being improved daily. Packages in R can be installed via the `install.packages()` function. For example, in the following, we install the package “car”:

```
> install.packages("car")
> library(car)
```

The `library(car)` function loads the car package into the workspace so that it is ready and available to use. Coding `library()` will reveal which packages are installed in the current workspace (try it). Packages are updated regularly by authors. Re-installing the given package will ensure you have the most recent, updated version. Getting help for a particular package can be obtained via the `help(package = )` function; for example, for the car package, one would code `help(package = car)`.

### 3.2.4 Writing Functions in R

A function in R, as opposed to mathematics, is essentially and most generally a program for carrying out something. Functions can range from simple to very complex. As an example of writing a simple function, consider the following where we wish to compute speed as a function of distance over time, where time is equal to 2 hours and distance is 180 miles:

```
>speed <- function(d){
+ d/2
+ }
>print(speed(180))
[1] 90
```

Speed is thus equal to 90 miles per hour. Of course, writing functions in R or other software can become very complex and will usually require a lot of hacking as well as trial and error.

### 3.2.5 Mathematics and Statistics Using R

The chief benefit of computers is not in inventing new mathematics. That pretty much still has to be done by the creative intellect. Computers, however, can help mathematicians know if they are on the right track by verifying proofs and such. More than anything, computers are useful in performing mathematics once you know what has to be computed and what it means once computed. Computers speed up the task and are less error-prone compared to their human counterparts.

For instance, knowing that you need to compute the derivative of a function and what it means in a given context is more important than computing a derivative manually. The computing of the derivative is best left to software such as R. *Humans make mistakes*. Computers are typically much more accurate and precise. As anxiety-provoking as it can be trusting the computer to fly your airplane, the computer is usually more trustworthy and accurate than a human pilot!

#### 3.2.5.1 Addition, Subtraction, Multiplication, and Division

Adding and subtracting numbers in R is straightforward. For example,

```
> 5 + 10
[1] 15
> 10-5
[1] 5
```

We can multiply and divide numbers quite easily as well:

```
> 10/5
[1] 2
> 5*10
[1] 50
```

#### 3.2.5.2 Logarithms and Exponentials

Recall that *logarithms* and *exponentials* are inverses of one another. An inverse function is one such that  $x=f^{-1}(y)$  if and only if  $f(x) = y$ .

A logarithm is essentially an *exponent*. The logarithm is the exponent to which the base needs to be raised in order to get the particular number we are taking the logarithm of. For example, consider the number 4. The log of 4 is

```
> log(4)
[1] 1.386294
```

The above reads that 1.38 is the exponent to which the base needs to be raised in order to get 4. But what is the base here? When we take a log in R without specifying the base, R automatically associates it with the *natural log*, that is, to base  $e$ . Recall that  $e$  is a constant equal to approximately 2.718, and hence when we compute the exponential of 1.386294, we obtain a number that when rounded up is equal to 4:

```
> exp(1.386294)
[1] 3.999999
```

We can easily change the base of the logarithm by the following, where we compute  $\log(4)$  but to base 10, which ends up being equal to 0.60206:

```
> log(x = 4, base=10)
[1] 0.60206
```

### 3.2.5.3 *Vectors and Matrices*

To create a vector in R, we “concatenate” elements of the vector via `c()` as follows,

```
> a <- c(0, 2, 7, 8, 10)
> a
[1] 0 2 7 8 10
```

To create a matrix in R, we designate the number of rows (`nrow`) as well as columns (`ncol`):

```
> A <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
> A
```

```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Many times in data management and statistical analyses, we wish to extract a specific given row or column from a matrix. This is a very useful task. We can do this by *indexing* the particular position in the given column or row:

```
> A[, 1]
[1] 1 2
> A[1, ]
[1] 1 3
```

Note above that `A[, 1]` extracted the first column, while `A[1, ]` extracted the first row.

#### 3.2.5.4 Means

The following computes the arithmetic mean on a small data set:

```
> data <- c(1, 5, 8, 9, 10)
> mean(data)
[1] 6.6
```

The variance and standard deviation of the above data are computed as:

```
> sd(data)
[1] 3.646917
> var(data)
[1] 13.3
```

Note that both the variance and standard deviation use  $n - 1$  in the denominator rather than  $n$  (Davies, 2016, p. 279). That is, R assumes we are computing these statistics to serve as *estimators* of their corresponding population counterparts. Recall that these statistics with  $n - 1$  in the denominators are sometimes referred to as the *corrected* versions of the variance and standard deviation, and losing a degree of freedom makes them *unbiased estimators*.

#### 3.2.5.5 Covariance and Correlation

For the following two vectors, we compute both the covariance and correlation:

```
> x <- c(0, 3, 7, 2, 8, 9)
> y <- c(3, 6, 8, 3, 8, 10)
> cov(x, y)
[1] 10.06667
> cor(x, y)
[1] 0.9576546
```

We see that the covariance is equal to 10.06 while the correlation is equal to 0.95. Of course, we could have also computed the correlation the long way as follows where we divide the covariance by the product of standard deviations:

```
> r <- cov(x, y) / (sd(x) * sd(y))
> r
[1] 0.9576546
```

3.2.5.6 *Sampling with Replacement in R*

We can easily sample with replacement in R. Suppose we wish to sample 100 observations with replacement; we compute:

```
> x = c(0:100)
> x = sample(y, 100, replace = T)
> x
```

|      |     |    |    |    |    |     |    |    |    |    |     |    |    |    |    |    |    |
|------|-----|----|----|----|----|-----|----|----|----|----|-----|----|----|----|----|----|----|
| [1]  | 14  | 2  | 2  | 68 | 98 | 48  | 26 | 93 | 45 | 98 | 16  | 29 | 47 | 38 | 56 | 3  | 26 |
| [18] | 42  | 37 | 23 | 24 | 53 | 88  | 67 | 0  | 56 | 86 | 60  | 17 | 91 | 3  | 47 | 6  | 44 |
| [35] | 98  | 24 | 27 | 53 | 46 | 100 | 80 | 25 | 84 | 18 | 100 | 47 | 68 | 15 | 97 | 17 | 55 |
| [52] | 100 | 5  | 82 | 57 | 41 | 93  | 32 | 6  | 93 | 99 | 32  | 55 | 69 | 85 | 76 | 58 | 7  |
| [69] | 48  | 14 | 75 | 21 | 90 | 81  | 22 | 41 | 59 | 89 | 59  | 16 | 30 | 14 | 87 | 46 | 33 |
| [86] | 82  | 14 | 72 | 14 | 29 | 18  | 44 | 77 | 77 | 53 | 94  | 46 | 95 | 0  | 12 |    |    |

The first row in the code `x = c(0:100)` gives the range of values from which we want to sample. If we do not want to replace, then we designate `replace = F` instead of `replace = T`.

3.2.5.7 *Visualization and Plots*

One of the benefits of using R (or Python) over many other packages (e.g., SPSS) is the awesome and virtually unlimited capacity for visualization. As the ease with which data is collected along with the sheer volume and size of data in the age of “Big Data,” visualizing data becomes all the more important as to discern patterns and trends.

We can produce a basic plot in R as follows on data in vectors `a` and `b` (left plot, Figure 3.1):

```
> a <- c(0, 4, 8, 9, 10)
> b <- c(3, 7, 8, 9, 11)
> plot(a, b)
```

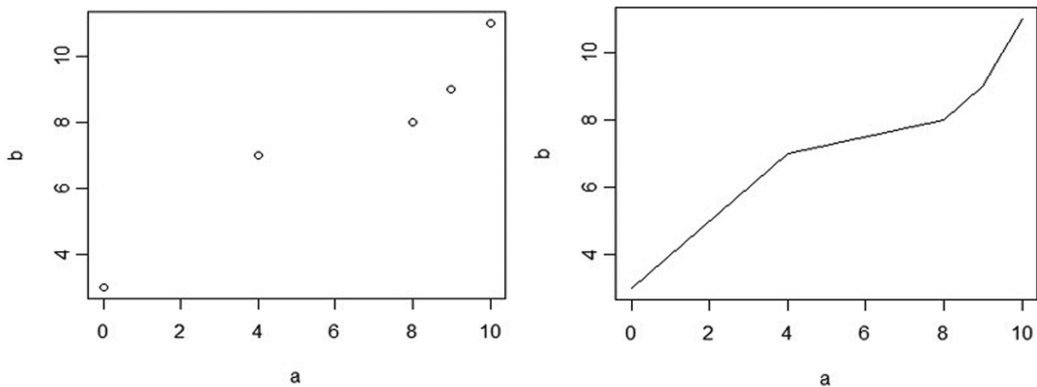


Figure 3.1 Basic plot of data (left). Joining the data with lines (right).



By specifying `type=` in the plot, we can do things such as join lines as in the above (right):

```
> plot(a, b, type="l")
```

### 3.2.5.7.1 BOXPLOTS

Boxplots give us a sense of a distribution by providing the median (middle bar) and upper and lower quartiles of the distribution. Creating boxplots in R is very easy. Consider the boxplot in Figure 3.2 computed on a given vector of data:

```
> x <- c(0, 3, 8, 1, 7, 4, 1, 10)
> boxplot(x)
```

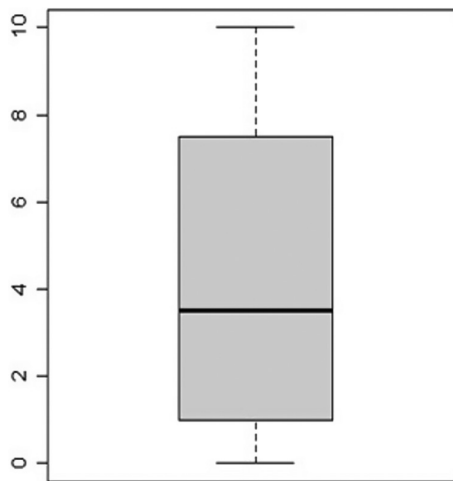


Figure 3.2 Boxplot on data set 0, 3, 8, 1, 7, 4, 1, 10.

For the data in Figure 3.2, we can see that the distribution is skewed toward higher values since there is more area in the plot toward the upper values. That is, the distribution is slightly positively skewed.

### 3.2.6 Further Readings and Resources in R

There are many excellent books and sources on R software, from beginner to more advanced, including a wealth of information online. Though the two sources below we will mention are excellent, neither of them should be used *exclusively* in learning statistics or machine learning. They are most useful in conjunction with more theoretical texts. This is a reminder of the theme emphasized earlier in the book that *computing* statistics is not the same as *learning* statistics. The actual statistics are in logic and mathematics, and while computing may help in *demonstrating* these concepts, it should never be regarded as equivalent to them. The “computing climate” of today too often replaces the “statistical climate” of yesterday. Long story short, *thinking must always precede computing, which implies mathematics and statistics (and reasoning about such) should precede computing.*

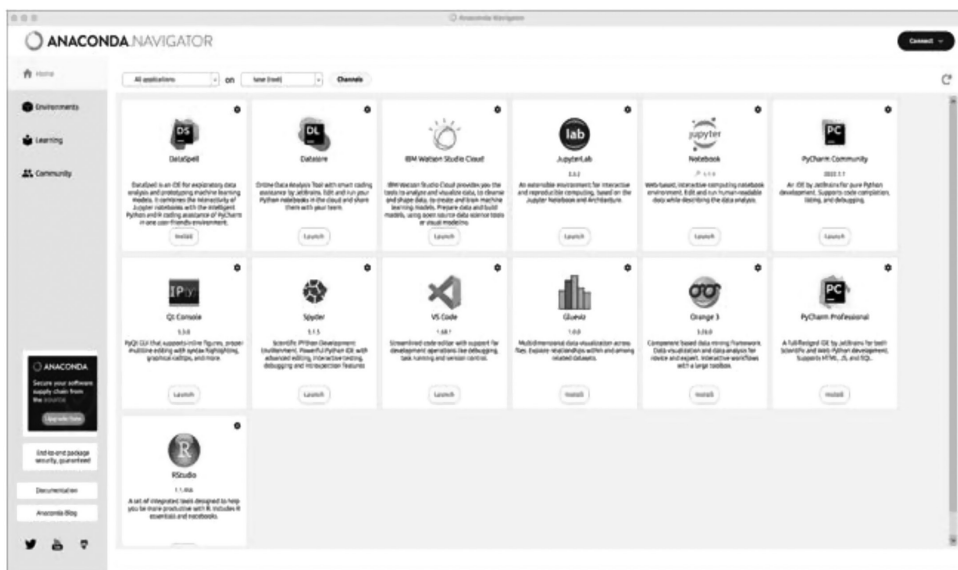
*The Art of R Programming* (Matloff, 2011) is one of the better introductory books on R that goes beyond simple functions and introduces the reader to some of the background of the R language. Hence, as the title suggests, the book consists of an introduction to actual *programming* rather than only *using* R. Among topics discussed are *data frames*, *programming structures*, *object-oriented programming*, as well as *debugging* and performance enhancement as it relates to the speed and memory of programs and computers respectively. *The Book of R* (Davies, 2016) is a fabulous compendium and overview of functions in R as they relate to statistics, including modeling in the analysis of variance as well as simple linear and multiple regression.

### 3.3 Python

In contrast to R, which is a programming language specifically designed for statistical applications, Python is a more *general-purpose* computing language. As such, it is commonly used for machine learning while R is perhaps more often used in statistical communities. Like R, there are many packages and libraries available in Python. The package *scikit-learn* is by far the most common package for machine learning applications. Some of the more popular packages that are used in statistics and machine learning problems include *NumPy*, *SciPy*, *Matplotlib*, *pandas*, and others. In most applications of machine learning these and other libraries are often imported before importing the package that actually performs the machine learning algorithm needed.

#### 3.3.1 Installing Python

Python is most easily accessed via Anaconda ([www.anaconda.com](http://www.anaconda.com)):



From there, one can load a variety of notebooks; *Jupyter* and *Spyder* are common options. In this book, we chose to use *Spyder* for its convenience, interface, and facility for data analysis applications. *Spyder* can also be downloaded directly without the use of Anaconda from [www.spyder-ide.org](http://www.spyder-ide.org).

### 3.3.2 Elements of Python

As was the case for R, Python operates on *objects*, where an object can be of two types: *scalar* vs. *non-scalar*. Scalar objects are indivisible and can be thought of as the “atoms” of the language (Gutttag, 2013). Scalar objects include `int` for integers, `float` for real numbers, `bool`, which are Boolean “true” vs. “false” values, and `None`, which has only a single value. As a simple example (Gutttag, 2013, p. 9),  $4 + 2$  is an expression and denotes the object 6 of type `int`, whereas the expression  $3.0 + 7.0$  denotes the object 10.0 of type `float`.

We can ask Python to confirm the type of object via `type()`:

```
type(3)
Out[195]: int
type(3.0)
Out[196]: float
```

A *variable* in Python simply denotes a name; it does not have the same meaning as in mathematics. For example, the following is a variable:

```
> pi = 3.14159
```

That is, we have identified 3.14159 with “pi.” Of course, `pi` is in actuality a *constant*, but the point is that in the above we have simply identified 3.14159 with a variable name (`pi`).

Just as we did in R, comments in Python can be indicated by “#” and are not read by Python when executing code. For example,

```
# the following is an example of addition
2 + 3
Out[203]: 5
```

### 3.3.3 Working with Data

As we did in R, we can easily import the *iris* data (and of course other data sets as well) into Python and print only a few cases:

```
from sklearn import datasets
iris = datasets.load_iris()
iris
Out[9]:
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
```

As another example, in the following we select cases 0 through 10 of a data set called `iq_data`:

```
iq_data[0:10]
Out[12]:
verbal quant analytic group
0    56.00\t56.00\t59.00\t.00
1    59.00\t42.00\t54.00\t.00
```

```

2  62.00\t43.00\t52.00\t.00
3  74.00\t35.00\t46.00\t.00
4  63.00\t39.00\t49.00\t.00
5  68.00\t50.00\t36.00\t.00
6  54.00\t54.00\t29.00\t.00
7  56.00\t52.00\t57.00\t.00
8  51.00\t46.00\t65.00\t.00
9  49.00\t39.00\t61.00\t.00

```

If you are completely unfamiliar with importing data into Python and would like a more thorough introduction with more examples, see Denis (2021b) or instruction on Python’s *scikit-learn* site.

### 3.3.4 Python Functions for Data Analysis

The *scikit-learn* library is the “go-to” package for machine learning in Python. It is open-source and in constant development. It also has a devoted user guide on the Python website and can be installed via *Anaconda*. Python also offers many preprocessing functions, which are functions applied to data before actually engaging in analysis. Oftentimes in statistical and machine learning applications we need to transform data in some way before implementing the algorithm or method. In statistical parlance, this is usually and simply referred to as *transforming* data while in machine learning jargon it is often called *preprocessing* data. There are multiple options for preprocessing data in Python.

One useful and popular function is that of the `MinMaxScaler()` function. This function transforms features by scaling each feature to a given range (e.g., between zero and one). Another popular class of functions is that of the `pipeline()` function. The pipeline function reduces the code necessary for carrying something out and chains together multiple steps in the machine learning workflow so that instead of fitting multiple functions you can simply fit the pipeline function. It is also a way to organize all of your code into one function, especially if you regularly run the same coding steps on different data sets. `StandardScaler` in *scikit-learn* transforms data to z-scores having a mean of 0 and standard deviation of 1. Other scaling options include `RobustScaler()` which removes the median and scales the data according to the interquartile range as a default. `Normalizer()` scales each variable or vector to have Euclidean length equal to one. There are of course other scaling options and we refer the reader to Python documentation or other resources to explore them.

Using *sklearn*, we can split data into train and test sets using the `train_test_split` function. Features (i.e., variables) are usually denoted by *X*, while labels with *y*. We will revisit shortly how to perform data splits in the software a bit later in the current chapter. We will also see examples of these and other functions throughout the book as we survey specific applications of Python applied to various statistical and machine learning methods.

#### 3.3.4.1 Mathematics Using Python

As with R, we can get quite a bit of essential mathematics done in Python. For example, basic numerical operations are very easy:

```

2 + 5
Out [8]: 7

```

Python can also perform complex analysis with ease via `cmath`. Recall that the *conjugate* of a complex number has the same real part to the number but with an opposite sign on the imaginary part. For example, consider the following complex number:

```
z = 2 + 3j
```

The complex conjugate is computed:

```
z.conjugate()
Out [12]: (2-3j)
```

That is, the complex conjugate of  $2 + 3j$  is equal to  $2 - 3j$ . To obtain the absolute value of a number, we can compute:

```
abs(-5)
Out [13]: 5
```

We can also work directly with lists (or vectors in this context) in Python:

```
l = [1, 2, 3]
l
Out [21]: [1, 2, 3]
for item in l:
    print (item)
1
2
3
```

The `enumerate()` function will inspect the list and print the ordered index number of the item as well as the item:

```
for index, item in enumerate(l):
    print(index, item)
0 1
1 2
2 3
```

To see the above more clearly, let's change the contents of the list:

```
l = [14, 12, 100]
for index, item in enumerate(l):
    print(index, item)
0 14
1 12
2 100
```

Countless other programming functions exist in Python, though due to space considerations not surveyed here. The reader interested in learning more can assess the recommended sources at the end of the chapter.

Mathematics can also be performed using the `math` module. As an example, in the following we divide  $\pi$  by 2 after importing `math`:

```
import math
math.pi/2
Out [59]: 1.5707963267948966
```

### 3.3.4.2 *Splitting Data into Train and Test Sets*

There are a variety of ways of splitting a data set into *train* vs. *test* sets. A common way of getting this done is to use *sklearn*'s `train_test_split ()` function:

```
import numpy as np
from sklearn.model_selection import train_test_split
```

To demonstrate the function, we first produce some data:

```
X, y = np.arange(10).reshape((5, 2)), range(5)
X
Out [143]:
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
X_train, X_test, y_train, y_test = train_test_split(
... X, y, test_size=0.33, random_state=42)
```

In the above, `test_size=0.33` indicates the proportion of the test data set. When we execute the above code, we get the following for `X_train`:

```
X_train
Out [151]:
array([[4, 5],
       [0, 1],
       [6, 7]])
```

and the following for `X_test`:

```
X_test
Out [152]:
array([[2, 3],
       [8, 9]])
```

The reader will note the above is 0.40 of the data and not 0.33. This is simply due to rounding, as selecting only a single case would comprise a proportion of only 0.20 of the data. We can easily adjust the size of the test set by adjusting `test_size`:

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.50, random_state=42)
X_test
Out[154]:
array([[2, 3],
[8, 9],
[4, 5]])
```

Again, the proportion is rounded up. We requested `test_size=0.50`, but 3/5 is 0.60. We can also choose to shuffle the training and test sets through the `shuffle=True` or `shuffle=False` option. In the first instance below, we do not shuffle, where in the second we do. Note that the values of the split are different depending on this decision:

```
train_test_split(y, shuffle=False)
Out[155]: [[0, 1, 2], [3, 4]]
train_test_split(y, shuffle=True)
Out[156]: [[4, 1, 2], [3, 0]]
```

### 3.3.5 Further Readings and Resources in Python

There are several great books on learning Python. One of the more popular books as it pertains to machine learning is that of Müller and Guido (2017) as well as Géron (2017). Much of Python coding can be dug up using online sources such as Python's scikit-learn site: <https://scikit-learn.org>.

## 3.4 Chapter Review Exercises

1. Access the iris data in R, compute `summary(iris)`, and comment on the output.
2. In R, request the first column of the iris data.
3. Generate a small data set in R of two columns, giving a name to each column.
4. Install the *car* package in R and access its online documentation via [cran.r-project.org/web/packages/car/car.pdf](https://cran.r-project.org/web/packages/car/car.pdf)
5. In Python, import the data set *iris* and describe what the first row of numbers represents.
6. Generate a list of ten numbers in Python and print the list.
7. Import the *math* module in Python and compute the square root of 2. Why is the answer unsatisfactory?



# Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>



## **PART II**

# **Models and Methods**



# Taylor & Francis

Taylor & Francis Group

<http://taylorandfrancis.com>

## 4 Univariate and Multivariate Analysis of Variance Models

Analysis of variance (ANOVA) models have historically been the bedrock of applied statistical approaches in the experimental sciences such as biology, psychology, and anywhere else where randomization and random assignment are used. Beginning with R.A. Fisher in 1925, ANOVA models were introduced as a convenient way to “package” the general linear model first developed in the late 1800s primarily by Yule and Pearson in a regression context, but with the aim specifically for experimenters who actively manipulated an independent variable to observe its effects on a dependent variable. Though regression originated primarily in the 1890s and ANOVA in the 1920s, there is of course a wide overlap between both models, where ANOVA can in most cases be considered a “special case” of the wider regression model, and regression can be considered a special case of the even wider *general linear model*. To go even further, the general linear model can be considered a special case of the even wider *generalized linear model*. And when several response variables are modeled simultaneously yet the independent variable(s) remain categorical, such gives rise to the *multivariate analysis of variance (MANOVA)*, surveyed later in the chapter. Linear discriminant analysis, or “LDA” for short, can be considered the “reverse” of MANOVA as we will see, where instead of evaluating group differences on some linear combination of variables, we seek instead to know the discriminant function(s) responsible for such differentiation.

Our review of the classic ANOVA model is relatively brief, both because we assume the reader is already familiar with ANOVA at least to some extent, but also because of space considerations in a book where there are so many other things to write about. Our brief treatment does not, however, imply that ANOVA is not *theoretically deep*, or somehow not as “important” as it once was, historically. Indeed, pedagogically, understanding “basic ANOVA” (as non-quantitative folks like to call it) at a deep level pays dividends to understanding statistical methods in their entirety. So-called “advanced” methods such as multilevel modeling are nothing more than, at their core, generalizations and extensions of the “basic” ANOVA model coupled with regression principles. Thus, the ANOVA model may be considered relatively simple in the bigger scheme of things, but understanding its concepts in some depth is definitely not, especially for the relative newcomer.

The complexity of ANOVA when studied at a deeper theoretical level is evidenced by sources such as that by Scheffé (1959), who thoroughly unpacks the model and its underlying mathematics. For an introduction to how ANOVA is constructed from the ground up in terms of simple deviations from arithmetic means leading up to the construction of the classic ANOVA table, see Denis (2021a).

### 4.1 The Classic ANOVA Model

In classic ANOVA, we seek to evaluate null hypotheses of the kind

$$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 \quad (4.1)$$

against alternative hypotheses that somewhere among the means is at least one pairwise difference; for example,

$$H_1 : \mu_1 \neq \mu_2, \mu_2 = \mu_3, \mu_3 = \mu_4$$

Of course, we usually do not typically have knowledge of the *population means* since in most cases populations are extremely large. If we do have the populations at our disposal, then evaluating the null hypothesis is a simple matter such that it is enough to simply look at the population means and see if they are equal or not and no statistical inference is required. This point should not be overlooked; if you find yourself conducting a significance test on *population means*, then something is amiss in your understanding of statistical inference! To reiterate, if you have the population parameters, the “game” is done; you can simply observe what the summary statistics on the population have to say and conclude population difference on this descriptive basis alone.

In the more realistic case where we only have sample data, the job is to evaluate the probability of observing sample differences of the kind we have observed under the assumption that the null hypothesis is tenable, or “true.” As a simple example, suppose we obtained the following four sample means:

$$\bar{y}_1 = 10, \bar{y}_2 = 15, \bar{y}_3 = 20, \bar{y}_4 = 21$$

The question we wish to ask is, *What is the probability of observing mean differences of this kind under the assumption of the null hypothesis?* More succinctly, we wish to address the probability of the data given the null hypothesis, that of  $p(D / H_0)$ . If that probability is small (for example, less than 0.05 as the typical significance level), it suggests differences of the kind we have observed are unlikely to have occurred under the null hypothesis and we begin to doubt the tenability of the null. We may begin to doubt it so much that we reject the null and infer the *statistical alternative hypothesis* (more on this shortly) that somewhere among the means there is reason to believe there is at least one pairwise difference between means.

The outcome of the ANOVA procedure culminates in determining the  $p$ -value for evaluating  $p(D / H_0)$ . As the reader may recall, the  $p$ -value is determined by partitioning sources of variation *between* and *within* in ANOVA, producing variances from these sources, the so-called *mean squares* of ANOVA. Take note that mean squares simply imply an average (i.e., mean) of sorts, where sums of squares are divided by degrees of freedom, hence the name “mean squares.” From this, we formulate a ratio ( $F$ -ratio) of the variance *between* relative to the variance *within* groups:

$$F = \frac{\sigma^2_{\text{Between}}}{\sigma^2_{\text{Within}}} \quad (4.2)$$

The expectation for  $F$  is approximately equal to 1.0. More precisely, the exact expectation  $E$  is equal to

$$E(F) = \frac{df_{\text{error}}}{df_{\text{error}} - 2}$$

where  $df$  error are the degrees of freedom for the error term in the ANOVA summary table, equal to  $N - J$  where  $N$  is the total number of data points and  $J$  is the number of groups or levels on the independent variable.

If  $\sigma^2_{Between} > \sigma^2_{Within}$  more so that we would have expected by random error or chance alone, then this counts as evidence against the null hypothesis and we reject it in favor of the statistical alternative. Each ANOVA has accompanied with it a summary table that shows the partition of the various sums of squares and the production of mean squares. Table 4.1 is a classic summary table illustrating the classic ANOVA partition.

Table 4.1 Partition of Sums of Squares in Classic One-Way Fixed-Effects ANOVA Table

| Source  | Sum of Squares                                     | Degrees<br>of<br>Freedom | Mean Squares                                                                                                                        | F-Ratio                                                                                                     | p-Value                                                                                   |
|---------|----------------------------------------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Between | $\sum_j n_j (\bar{y}_j - \bar{y})^2$               | $J-1$                    | $\frac{\sum_j n_j (\bar{y}_j - \bar{y})^2}{J-1}$                                                                                    | $\frac{\sum_j n_j (\bar{y}_j - \bar{y})^2 / J-1}{\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y}_j)^2 / (N-J)}$ | Evaluate<br>obtained<br>ratio as $F$ -stat<br>on $J-1$ and<br>$N-J$ degrees<br>of freedom |
| Within  | $\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y}_j)^2$ | $N-J$                    | $\frac{\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y}_j)^2}{N-J}$                                                                      | —                                                                                                           | —                                                                                         |
| Total   | $\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y})^2$   | $N-1$                    | $s^2 = \frac{\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y})^2}{N-1}$<br>$(N-1)(s^2) = \sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y})^2$ | —                                                                                                           | —                                                                                         |

In the summary table, we note the partition of sources of variation into *between* and *within*. The sums of squares for between, equal to

$$\sum_j n_j (\bar{y}_j - \bar{y})^2$$

is the sum of squared deviations of group means,  $\bar{y}_j$ , minus the overall or “grand mean” of the data,  $\bar{y}$ , where  $n_j$  before the squared term is simply the number of observations per group, in this case assumed to be equal. Deviations from the mean are squared since any sum of deviations from the mean will necessarily equal zero, that is,  $\sum_j n_j (\bar{y}_j - \bar{y})$  would always equal zero and formulating the aforementioned  $F$ -ratio would be impossible. By squaring deviations, we allow for  $n_j (\bar{y}_j - \bar{y})^2$  to increase in magnitude under the condition that group means  $\bar{y}_j$  differ from the overall mean,  $\bar{y}$ . Hence, the between sums of squares in the partition is of most interest to the scientist because, all else equal, a larger value than not hints that there is a difference between being in one treatment group versus being in another.

The second sums of squares (SS) term, that of SS within

$$\sum_{j=1}^J \sum_i^n (y_{ij} - \bar{y}_j)^2$$

assesses the degree of deviation that exists within each group for observations  $y_{ij}$  from corresponding group means,  $\bar{y}_j$ . Since these observations are occurring within the same group, they cannot be due to treatment effects and instead we chalk them up to random error and perhaps sources *not* included in the model. Once more we sum squared deviations to avoid the sum always equaling zero if we had simply summed deviations without squaring. A large *within* term is typically not a favorable result because it suggests that quite a bit of variation in one's data is due to factors unaccounted for by the treatment variable. For example, if SS within are equal to SS between, then this would suggest that variation in the data is just as well “explained” by random chance or noise than it is by group differences. The favorable result for the experimenter then is that SS between is substantially larger than SS within which would suggest that of the total variation in the data under study, indicated by SS total,  $\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y})^2$ , a large proportion of it can be “explained” by between-group differences (i.e., SS between). This ratio of SS between to SS total is of interest then, and we will shortly coin it as an *effect size* measure called *Eta-squared*.

#### 4.1.1 Mean Squares

The analysis of variance gets its name because it analyzes *variances*, and not simply sums of squares. The sums of squares we have surveyed are not variances. Recall that, as was the case for the simple sample variance, to generate a variance we need to divide by degrees of freedom. Dividing each source of variation, that of *between* and *within*, by corresponding degrees of freedom will produce the aforementioned *mean squares*, which is simply another name for variances. The degrees of freedom for between are equal to  $J - 1$ ; that is, one less than there are number of groups on the independent variable. The degrees of freedom for within are equal to  $N - J$ , the total number of observations minus the number of groups.

Dividing sums of squares by degrees of freedom generates the *mean squares* for between (*MS between*) and within (*MS within*), of which we can then make a comparison of how large MS between is relative to MS within. This comparison via a ratio is the aforementioned *F*-ratio. Recall that we said the expectation of *F* under the null hypothesis is equal to approximately 1. If there are treatment effects, then we would expect MS between to be larger than MS within, and hence  $F > 1$ . However, small deviations from 1.0 can be easily attributed to chance or random fluctuation, and hence we will conduct a statistical significance test on *F* to determine the probability of observing an *F* as we have obtained under the hypothetical (and somewhat unrealistic) situation that the null hypothesis was “true.” If the probability of the *F* we have observed is less than some conventional level such as 0.05, then we reject the null hypothesis and infer the statistical alternative (not necessarily the *substantive alternative*; we will discuss the difference later) that somewhere among the population means there is at least one pairwise difference.

### 4.1.2 Expected Mean Squares of ANOVA

In our overview of how ANOVA works, though we walked through the logic of the summary table, we did not in any way defend or justify that the  $F$ -ratio is actually estimating quantities of interest to us. In other words, we did not justify that computing the  $F$ -ratio as we have done actually makes good sense, even if it did seem intuitive arithmetically to compute it that way. MS between and MS within are simply computations made on *sample* data, but how do we know what they are actually estimating in the population?

Recall that when we computed a simple sample variance on our data, we justified its computation by taking its expectation, and found that with  $n - 1$  in the denominator it produced an unbiased estimate of the population variance,  $\sigma^2$ . We need to ask an analogous question of the variances we have generated from ANOVA. That is, *what is the expectation of MS between and MS within?* Only then will we know that we are in fact estimating quantities that justify the ensuing  $F$ -ratio.

For full details on taking expectations, see Denis (2021a). Due to space limitations, we here only summarize the end result. In a *fixed-effects model*, the expectation of MS between turns out to be

$$E(\text{MS Between}) = \frac{\sum_j (n_j \alpha_j^2)}{J - 1} + \sigma_e^2 \quad (4.3)$$

and for within,  $E(\text{MS Within}) = \sigma_e^2$ . If there are no treatment effects, then  $\alpha_j^2 = 0$  and hence

$$\begin{aligned} E(\text{MS Between}) &= \frac{\sum_j (n_j (0)^2)}{J - 1} + \sigma_e^2 \\ &= \sigma_e^2 \end{aligned}$$

and so an  $F$ -ratio computed on these variances would produce a ratio approximately equal to 1.0, since  $\sigma_e^2 / \sigma_e^2 = 1$ . The extent to which  $\alpha_j^2 > 0$  is the extent to which treatment effects exist and thus we would expect  $\sum_j (n_j \alpha_j^2)$  to be greater than 0, resulting in  $E(\text{MS Between})$  being greater than simply  $\sigma_e^2$ . Hence, we would expect under these conditions an  $F$  statistic greater than 1.0 such that if large enough when evaluated as  $F$  on  $J - 1$  and  $N - J$  degrees of freedom, we would deem *statistically significant* and reject the null hypothesis.

We mentioned that this was for a *fixed-effects model*, for which the reader may already be familiar. In a fixed-effects model, we assume that in theoretical repetitions of the experiment, the same levels of the independent variable would be used. For instance, a fixed-effects ANOVA with drug levels 0, 1, and 2 mg implies that these *specific* levels are of interest to the researcher and would be used again if we performed the experiment repeatedly. This is in contrast to a *random-effects model* in which each experiment would feature a random sampling of levels from a theoretical probability distribution of potential levels that could have been selected based on some probability model (usually random sampling). Understanding fixed-effects models benefits greatly from understanding random-effects models, which we survey shortly.

To reiterate, the purpose of the above is simply to assert that the computation of the  $F$ -ratio for the fixed-effects analysis of variance by taking MS between to MS within is justified by taking expected values. In practice, the user of ANOVA is not computing expected mean squares each time she conducts the procedure. Rather, she is simply computing  $F$  (or interpreting output from software) and making a decision on  $H_0$ . The purpose of the above expected mean squares

is simply to provide theoretical evidence that the  $F$ -ratio is theoretically justified by expected mean squares theory. It may seem at first trivial to require such justification in the classic one-way ANOVA. However, more complicated models in random-effects and mixed-models theory reveal that MS within (or “MS error”) is not always the most suitable denominator for evaluating null hypotheses for various terms in the ANOVA. We will survey these models a bit later in the chapter.

#### 4.1.3 Effect Sizes for ANOVA

Rejecting a null hypothesis in ANOVA only suggests that the observed statistic is relatively implausible or unlikely under the null hypothesis and as such there is evidence to suggest population means are unequal. It does not, by itself, suggest that differences observed are in any sense *sizable*, *meaningful*, or otherwise *scientifically important*. Indeed, inspection of the ANOVA summary table reveals that with sufficiently large sample size,  $N$ , statistical significance and thus rejection of the null hypothesis is assured even for small deviations of group means from the grand mean. As an example of this, consider what happens to SS within when divided by a sizable  $N - J$  term, due to  $N$  being extremely large. The numerator  $\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y}_j)^2$  is divided by an increasingly large number,  $N - J$ , and the ratio becomes smaller and smaller and in the limit as  $N$  grows without bound, MS within eventually goes to 0:

$$\lim_{N \rightarrow \infty} \text{MS}_{\text{Within}} = 0$$

A numerator divided by an increasingly larger denominator generates a ratio that gets smaller and smaller. As  $\text{MS}_{\text{Within}}$  gets smaller,  $\text{MS}_{\text{Between}}$  begins to look large in comparison, and hence a statistically significant  $F$ -ratio is possible even if the actual distance between means is extremely small. Hence, “ $p < 0.05$ ” can occur even under conditions where mean differences are negligible and not substantively meaningful or important. If one naively associates the statement “ $p < 0.05$ ” with the experiment necessarily revealing an “important” effect in any way, such a conclusion could prove tremendously misleading to a reader who may be otherwise ignorant of what a  $p$ -value is a function of. *Not understanding what a  $p$ -value tells you vs. what it does not tell you can result in serious miscommunication of results and this occurs daily in published research results across many professional and even “high impact factor” journals. Conflating statistical significance with effect size is still very common across the sciences despite almost 100 years of attempts to correct these misconceptions.*

As a result of the  $p$ -value not indicating the strength of effect in an experiment or study, a measure that should always be reported alongside the  $p$ -value is that of *effect size*. A good effect size does not allow its magnitude to be influenced by simple increases or decreases in sample size. For ANOVA, a classic effect size is obtained by taking a ratio of SS between to SS total:

$$\eta^2 = \frac{\text{SS Between}}{\text{SS Total}} = \frac{\sum_{j=1}^J n_j (\bar{y}_j - \bar{y})^2}{\sum_{j=1}^J \sum_{i=1}^n (y_{ij} - \bar{y})^2} \quad (4.4)$$

The extent to which SS between is large relative to SS total is the extent to which total variation is “accounted for” by treatment effects. It is easy to show that  $\eta^2$  has a maximum of 1.0 and a minimum of 0. A value of 1.0 would indicate that 100% of variation in the data is accounted for by



treatment effects, whereas a value of 0 would indicate that 0% of variation in the data is accounted for by treatment effects and would also imply that SS within/SS total is equal to 1.0 since SS between + SS within must equate to SS total.

*Eta-squared* is a popular statistic reported as a first step to effect size; however, it is biased upward, meaning that it typically *overestimates* the true effect size in the population. A less biased effect size is that of *omega-squared*,  $\omega^2$ , for which the following is the corresponding estimator:

$$\hat{\omega}^2 = \frac{\text{SS between} - (J-1)\text{MS within}}{\text{SS total} + \text{MS within}} \quad (4.5)$$

We note that  $\omega^2$  features degrees of freedom for between  $(J-1)$  in its formulation, making it more “inferential” than Eta-squared which only includes sums of squares. The extent to which the numerator is large relative to the denominator is the extent to which  $\omega^2$  will be larger than not. Unlike for  $\eta^2$ ,  $\omega^2$  can be negative (Okada, 2017), where a value of 1 indicates 100% of the variance in the response is accounted for by mean differences on the independent variable. An alternative but equivalent computation for  $\omega^2$  is

$$\omega^2 = \frac{(J-1)(F-1)}{(J-1)(F-1) + nJ} \quad (4.6)$$

where  $F$  is that obtained from the ANOVA and  $n$  is sample size per group, assuming in this case equal  $n$  per level of the independent variable. This alternative formula for  $\omega^2$  can be of use in cases where a researcher or journal article may not present the full summary table for the ANOVA but instead only make mention of  $F$ . More details on  $\omega^2$  can be found in Kirk (1995). Ideally, researchers should report *at least* both  $\eta^2$  and  $\omega^2$  in summarizing ANOVA findings. Generalized versions of both of these statistics that incorporate design features into their computation can also be computed. See Olejnik and Algina (2003) for details.

#### 4.1.4 Contrasts and Post-Hoc Tests for ANOVA

Typically, researchers are not so much interested in simply a statistically significant  $F$ -ratio from the ANOVA as much as they are in specific group *contrasts* between levels of the independent variable. In a two-group ANOVA, which is simply a generalization of univariate  $t$ , there is only one contrast to make, one group vs. the other, and the ANOVA or  $t$ -test has already made it. However, in cases where there are three groups or more, researchers often wish to specify specific pairwise group comparisons. For example, in a three-group problem, the researcher may wish to compare group 1 to a combination of groups 2 and 3. Perhaps group 1 is the control group and groups 2 and 3 are experimental groups. To do this, a *population comparison*  $c_i$  can be computed,

$$C_i = c_1\mu_1 + c_2\mu_2 + \cdots + c_J\mu_J = \sum_{j=1}^J c_j\mu_j \quad (4.7)$$

where  $c_1$  through  $c_J$  are weights, usually integers, and  $\mu_1$  through  $\mu_J$  are population means. A population comparison then is a *sum of weighted population means*. If we choose weights  $c_1$  through  $c_J$  such that these weights sum to 0, the population comparison becomes a population

*linear contrast*. Should the sum  $\sum_{j=1}^J c_j \mu_j$  deviate from 0 at all under this situation, it is clearly not due to the linear sum of weights  $c_1$  through  $c_J$  but instead must be a function of an inequality of population means  $\mu_1$  through  $\mu_J$ . This is the precise reason for choosing weights  $c_1$  through  $c_J$  so that they sum to 0, so that a deviation from 0 for  $C_i$  is reflective of a pairwise group comparison of interest to the researcher (in the form of the defined contrast).

As an example of a comparison featuring three groups (populations), we have the following:

$$C_i = c_1 \mu_1 + c_2 \mu_2 + c_3 \mu_3$$

Suppose the researcher hypothesizes that  $\mu_1$  is different from  $\mu_2$  and  $\mu_3$  as a set. This hypothesis could be reflected in the following population contrast:

$$\begin{aligned} C_i &= c_1 \mu_1 + c_2 \mu_2 + c_3 \mu_3 \\ &= (2)\mu_1 + (-1)\mu_2 + (-1)\mu_3 \end{aligned}$$

By choosing weights “2” and “−1” and “−1” that sum to 0, we see that the above successfully compares  $\mu_1$  to  $\mu_2$  and  $\mu_3$  as a set. Any deviation of  $C_i$  from 0 can only be due to a difference between  $\mu_1$  vs.  $\mu_2$  and  $\mu_3$ .

As usual, we rarely if ever know the values for  $\mu_1$ ,  $\mu_2$ , and  $\mu_3$ . Instead, we must estimate from them on data by computing corresponding sample means,  $y_1, y_2, y_3$ :

$$\begin{aligned} C_i &= c_1 \bar{y}_1 + c_2 \bar{y}_2 + c_3 \bar{y}_3 \\ &= (2)\bar{y}_1 + (-1)\bar{y}_2 + (-1)\bar{y}_3 \end{aligned}$$

which is now an *estimated linear contrast*, which can be stated more generally as

$$\begin{aligned} \hat{C}_i &= c_1 \bar{y}_1 + c_2 \bar{y}_2 + \cdots + c_J \bar{y}_J \\ &= \sum_{j=1}^J c_j \bar{y}_j \end{aligned} \tag{4.8}$$

with the variance of the contrast equal to

$$\sigma_{\hat{C}_i}^2 = \text{MS}_{\text{Error}} \sum_j \frac{c_j^2}{n_j} \tag{4.9}$$

and estimated standard error equal to the square root of the variance,

$$\sigma_{\hat{C}_i} = \sqrt{\sigma_{\hat{C}_i}^2} \tag{4.10}$$

With the estimated standard error in place, we can now evaluate a null hypothesis that  $C_i = 0$ :

$$H_0 : C_i = 0 \quad (4.11)$$

against an alternative hypothesis that the true contrast value is unequal to 0:

$$H_1 : C_i \neq 0 \quad (4.12)$$

The null for the contrast is typically evaluated as a  $t$  statistic on  $N-J$  degrees of freedom from the overall ANOVA,

$$t = \frac{\hat{C}_i - C_i}{\sigma_{\hat{C}_i}} \quad (4.13)$$

The extent to which the difference  $\hat{C}_i - C_i$  is large relative to the estimated standard error,  $\sigma_{\hat{C}_i}$  is the extent to which there is eventually grounds to reject the null and infer the statistical alternative. That is, we have evidence, at least for the given experiment, that the particular mean difference hypothesized by the researcher exists in the population from which the data were drawn.

*Post-hoc tests* attempt to control for the inflation of the type I error rate across comparisons or contrasts. For a review of *post-hoc* tests, see Ott and Longnecker (2016). One popular test is the Tukey HSD (“Honestly Significant Difference”). We perform it following an ANOVA on some fictitious data:

```
> y <- c(5, 8, 2, 9, 4, 7, 2, 8, 20, 21, 23, 25)
> x <- c(0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3)
> x.factor <- factor(x)
> model <- aov(y ~ x.factor)
> summary(model)
```

|           | Df | Sum Sq | Mean Sq | F value | Pr(>F)    |
|-----------|----|--------|---------|---------|-----------|
| x.factor  | 3  | 599.0  | 199.67  | 7.729   | 0.0095 ** |
| Residuals | 8  | 206.7  | 25.83   |         |           |

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> tukeyHSD <- TukeyHSD(model)

Tukey multiple comparisons of means
 95% family-wise confidence level

Fit: aov(formula = y ~ x.factor)

$x.factor
```

|     | diff      | lwr         | upr      | p adj     |
|-----|-----------|-------------|----------|-----------|
| 1-0 | 1.666667  | -11.6229870 | 14.95632 | 0.9766194 |
| 2-0 | 5.000000  | -8.2896536  | 18.28965 | 0.6409284 |
| 3-0 | 18.000000 | 4.7103464   | 31.28965 | 0.0106625 |
| 2-1 | 3.333333  | -9.9563203  | 16.62299 | 0.8512024 |
| 3-1 | 16.333333 | 3.0436797   | 29.62299 | 0.0181554 |
| 3-2 | 13.000000 | -0.2896536  | 26.28965 | 0.0551647 |

From the output, we can see that the difference between groups 3 and 0 is statistically significant ( $p = 0.0106625$ ) while the difference between group 3 and 1 is also statistically significant ( $p = 0.0181554$ ). The difference between group 3 and 2 is marginally significant ( $p = 0.0551647$ ).

We also demonstrate the Scheffé test, which is a more conservative test than the Tukey, meaning that it is more protective against type I errors (i.e., it is harder to reject the null with the Scheffé than with more *liberal* post-hoc tests):

```
> library(DescTools)
> ScheffeTest(model)

Posthoc multiple comparisons of means: Scheffe Test
 95% family-wise confidence level

$x.factor
```

|     | diff      | lwr.ci     | upr.ci   | pval     |
|-----|-----------|------------|----------|----------|
| 1-0 | 1.666667  | -12.827677 | 16.16101 | 0.9824   |
| 2-0 | 5.000000  | -9.494344  | 19.49434 | 0.7027   |
| 3-0 | 18.000000 | 3.505656   | 32.49434 | 0.0170 * |
| 2-1 | 3.333333  | -11.161010 | 17.82768 | 0.8832   |
| 3-1 | 16.333333 | 1.838990   | 30.82768 | 0.0282 * |
| 3-2 | 13.000000 | -1.494344  | 27.49434 | 0.0800 . |

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can see that differences 3-0 and 3-1 are still statistically significant, yet  $p$ -values are slightly larger than with the Tukey. The  $p$ -value for 3-2 is now elevated at 0.0800, whereas recall with the Tukey, it was equal to 0.055.

#### 4.1.5 ANOVA in Python

Our demonstrations of ANOVA that follow are very simple and we proceed quite briskly, as we assume the reader already has some experience in fitting ANOVA models and evaluating assumptions. See Denis (2021b) for more Python software examples.

There are numerous ways to conduct a simple one-way analysis of variance in Python. In the following, we use the *SciPy* library to evaluate the null hypothesis that mean temperature is the same across populations 1–4:

```
from scipy.stats import f_oneway
temperature1 = [89, 89, 88, 78, 79]
temperature2 = [93, 92, 94, 89, 88]
temperature3 = [89, 88, 89, 93, 90]
temperature4 = [81, 78, 81, 92, 82]
```

Notice there are four groups. The first group has data values 89, 89, 88, 78, and 79. The second group has data values 93, 92, 94, 89, 88, and so on for the other two groups. We conduct the ANOVA using `f_oneway()`:

```
f_oneway(temperature1, temperature2, temperature3, temperature4)
Out [553]: F_onewayResult(statistic=4.6250000000000002,
pvalue=0.016336459839780215)
```

We see the reported  $F$ -stat is 4.625 with an associated  $p$ -value of 0.0163, which exceeds  $p < 0.05$ , and hence we would typically reject the null hypothesis of equal population means assuming assumptions are satisfied for the analysis. That is, somewhere among the means is at least one statistically significant mean difference.

One can also use `statsmodels` to conduct ANOVA, where in the following we also model an interaction (interactions will be discussed later; however, we assume the reader is at least somewhat familiar with them):

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

For this example, we will load the *Moore* data and adapt an example from `statsmodels`:

```
moore = sm.datasets.get_rdataset("Moore", "carData",
cache=True)
data = moore.data
data
Out [11]:
```

|   | partner.status | conformity | fcategory | fscore |
|---|----------------|------------|-----------|--------|
| 0 | low            | 8          | low       | 37     |
| 1 | low            | 4          | high      | 57     |
| 2 | low            | 8          | high      | 65     |
| 3 | low            | 7          | low       | 20     |
| 4 | low            | 10         | low       | 36     |
| 5 | low            | 6          | low       | 18     |

We will rename `partner.status` to `partner_status`:

```
data = data.rename(columns={"partner.status":
"partner_status"})
data
Out [13]:
```

|   | partner_status | conformity | fcategory | fscore |
|---|----------------|------------|-----------|--------|
| 0 | low            | 8          | low       | 37     |
| 1 | low            | 4          | high      | 57     |
| 2 | low            | 8          | high      | 65     |
| 3 | low            | 7          | low       | 20     |
| 4 | low            | 10         | low       | 36     |
| 5 | low            | 6          | low       | 18     |

We now conduct the ANOVA:

```
moore_lm = ols('conformity ~ C(fcategory, Sum)*C(partner_status, Sum)',
data=data).fit()
```

We now request a printout of the ANOVA summary table:

```
table = sm.stats.anova_lm(moore_lm, typ=2)
print(table)
```

|                                          | sum_sq     | df   | F         | PR(>F)   |
|------------------------------------------|------------|------|-----------|----------|
| C(fcategory, Sum)                        | 11.614700  | 2.0  | 0.276958  | 0.759564 |
| C(partner_status, Sum)                   | 212.213778 | 1.0  | 10.120692 | 0.002874 |
| C(fcategory, Sum):C(partner_status, Sum) | 175.488928 | 2.0  | 4.184623  | 0.022572 |
| Residual                                 | 817.763961 | 39.0 | NaN       | NaN      |

We can see from the above that the interaction effect is statistically significant ( $p = 0.022572$ ) as well as one of the main effects ( $p = 0.002874$ ), while the other main effect is not statistically significant ( $p = 0.759564$ ). Given that the interaction is statistically significant, it should be interpreted *prior* to the main effect and if the main effect is interpreted at all, the mention of a statistically significant interaction should be included in the interpretation. The interaction provides a wider *context* to the main effect. The results suggest that mean differences in partner status are contingent upon level of category.

#### 4.1.6 ANOVA in R

As in Python, there are many ways of fitting ANOVA models in R. We feature an example from the *Plant Growth* data set featuring the weight of different groups of plants (a few cases are printed below):

```
> library(plotly)
> data <- PlantGrowth
> set.seed(123)
> head(PlantGrowth)
```

|   | Weight | group |
|---|--------|-------|
| 1 | 4.17   | ctrl  |
| 2 | 5.58   | ctrl  |
| 3 | 5.18   | ctrl  |
| 4 | 6.11   | ctrl  |
| 5 | 4.50   | ctrl  |
| 6 | 4.61   | ctrl  |

```
> levels(data$group)
[1] "ctrl" "trt1" "trt2"
> res.aov <- aov(weight ~ group, data = data)
> summary(res.aov)
```

|           | Df | Sum Sq | Mean Sq | F value | Pr(>F)   |
|-----------|----|--------|---------|---------|----------|
| group     | 2  | 3.766  | 1.8832  | 4.846   | 0.0159 * |
| Residuals | 27 | 10.492 | 0.3886  |         |          |

```
---
```

```
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see there is a group main effect ( $p = 0.0159$ ). We can now conduct multiple comparisons to learn where the group differences are:

```
> TukeyHSD(res.aov)
Tukey multiple comparisons of means
95% family-wise confidence level

Fit: aov(formula = weight ~ group, data = data)

$group
      diff      lwr      upr      p adj
trt1-ctrl -0.371 -1.0622161 0.3202161 0.3908711
trt2-ctrl  0.494 -0.1972161 1.1852161 0.1979960
trt2-trt1  0.865  0.1737839 1.5562161 0.0120064
```

We see that the difference between treatment 1 and 2 is statistically significant with a  $p$ -value of 0.012. The other pairwise differences are not statistically significant. Equality of variances can also be evaluated. Many tests are available, we choose Bartlett's test:

```
> bartlett.test(weight ~ group, data)

Bartlett test of homogeneity of variances

data: weight by group
Bartlett's K-squared = 2.8786, df = 2, p-value = 0.2371
```

We do not have evidence to reject the assumption of equal variances ( $p = 0.2371$ ). It is wise to conduct various equality of variance tests on the same data to see if they report similar findings. If the assumption of equality of variances is rejected, while transformations on data are an option, what is even more important is to get curious as to the *why* behind the violation. Why are the variances unequal? Get curious from a *scientific* perspective before you address the problem *statistically* (e.g., such as with a transformation). Slight violations usually do not require any kind of remediation as ANOVA is quite robust to slight violations of assumptions. However, if violations of assumptions such as variances become more pronounced, especially for unequal sample sizes, corrective measures should be taken, or at minimum, results should be reported with requisite caveats. A nonparametric test such as the Kruskal-Wallis may be a useful option.

We assume the reader is already somewhat familiar with ANOVA and have only skimmed the surface as it pertains to applications of ANOVA in software. One should always verify assumptions (e.g., normality), check for equality of variances, and be sure that observations are independent of one another to the best of one's ability. Excellent theoretical resources on ANOVA include Hays (1994) and Kirk (1995). Denis (2021a) covers ANOVA in much more detail and features more software demonstrations in R and SPSS as well. Denis (2021b) also features ANOVA using Python.

## 4.2 Factorial ANOVA and Higher-Order Models

We have surveyed the one-way classic ANOVA in which there is only a single independent variable. Factorial ANOVA features more than a single independent variable, usually with the goal of seeking out *interactions*, that of the effect of one variable being contingent upon one or more other variables.

For example, it may be true that income (response variable) is found to be a function of education, but that relationship may depend, statistically, on what level of age we consider. Hence, there is said to be a *statistical interaction* between education and age. Note carefully that this conclusion alone does not imply that age “affects” education in any way, or that age or education “affect” income. Based on the statistical interaction alone, we do not have evidence to make any kind of “impact” statement. To even *approximate* such a claim would require a strong *research design* or feature variables that have such a naturally “organic” connection to one another (e.g., increasingly colder temperatures “affect” the rate at which water freezes).

Visualizing interactions is the best way to get a feel for them. For instance, in Figure 4.1 there is an interaction between gender and drug dose:

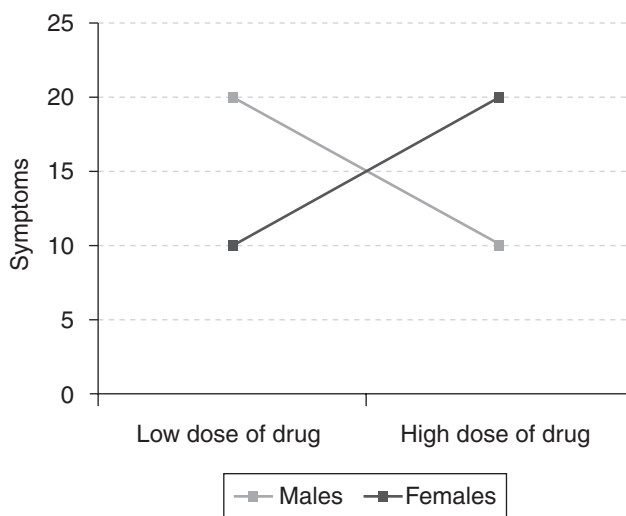


Figure 4.1 Interaction between gender and drug dose in the sample.

We can see from the plot that for females, scores increase from low dose to high dose. However, for males, the opposite effect holds. That is, scores begin high at low dose and then decrease to high dose. This interaction is rather pronounced; all that is technically required for an interaction is that lines be *nonparallel*. *A bit of thought reveals that interactions in sample data are almost always present. The real question, however, is whether we can infer such a sample interaction in the population from which the data were sampled.*

#### 4.2.1 Factorial ANOVA in Python

As a simple example of factorial ANOVA in Python, we adapt an example from Denis (2021b). First, we generate some data on achievement (`ac`), teacher (`teach`) and textbook (`text`). It is hypothesized that achievement scores are a function of teacher and textbook used in the course. The `pd.DataFrame()` function creates the data frame, where “pd” is “pandas”:



```

import pandas as pd
import statsmodels as sm
import statsmodels.api as sm
data = {'ac' : [70, 67, 65, 75, 76, 73, 69, 68, 70, 76, 77, 75, 85, 86,
85, 76, 75, 73, 95, 94, 89, 94, 93, 91],
'teach' : [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4,
4, 4, 4],
'text' : [1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1,
2, 2, 2]}
df_2 = pd.DataFrame(data)
df_2
Out[19]:

```

```

      ac  teach  text
0      70      1      1
1      67      1      1
2      65      1      1
3      75      1      2
4      76      1      2
5      73      1      2
6      69      2      1
7      68      2      1
8      70      2      1
9      76      2      2
10     77      2      2

```

Below, we fit the model of achievement as a function of teach and text, featuring type III sums of squares (`typ = 3`):

```

from statsmodels.formula.api import ols
model_1 = ols('ac ~ C(teach) + C(text)', data = df_2).fit()
table_1 = sm.stats.anova_lm(model_1, typ = 3)
print(table_1)

```

|           | sum_sq       | df   | F           | PR(>F)       |
|-----------|--------------|------|-------------|--------------|
| Intercept | 23885.408333 | 1.0  | 1220.637252 | 1.051343e-18 |
| C(teach)  | 1764.125000  | 3.0  | 30.051216   | 2.029775e-07 |
| C(text)   | 5.041667     | 1.0  | 0.257649    | 6.175816e-01 |
| Residual  | 371.791667   | 19.0 | NaN         | NaN          |

We can see from the summary table that both teach and text are statistically significant. We will now fit the interaction term, that between teach and text:

```

import statsmodels.api as sm
from statsmodels.formula.api import ols
model_2 = ols('ac ~ C(teach) + C(text) + C(teach)*C(text)', data = df_2).
fit()
table_2 = sm.stats.anova_lm(model_2, typ = 2)
print(table_2)

```

|                  | sum_sq      | df   | F          | PR(>F)       |
|------------------|-------------|------|------------|--------------|
| C(teach)         | 1764.125000 | 3.0  | 180.935897 | 1.488709e-12 |
| C(text)          | 5.041667    | 1.0  | 1.551282   | 2.308781e-01 |
| C(teach):C(text) | 319.791667  | 3.0  | 32.799145  | 4.574415e-07 |
| Residual         | 52.000000   | 16.0 | NaN        | NaN          |

We can see from the above that the interaction term is statistically significant ( $p = 4.574415e-07$ ). That is, teacher is statistically related to achievement, but that relation is contingent upon textbook used (i.e., based on the evidence for an interaction).

### 4.3 Random-Effects and Mixed Models

As we have said, the ANOVA we have surveyed thus far in this chapter has assumed that levels on the independent variable are *fixed*. To reiterate, in theoretical repetitions of an experiment, it was assumed we would be using the same levels each time. This is why it was called a *fixed-effects analysis of variance*.

When we do not assume levels are fixed but instead *random samples* of a population of possible levels, the factor is no longer fixed but instead *random*, which gives rise to the *random-effects analysis of variance model*. For example, if one were interested in the effects of dosage on headache symptoms, in a fixed-effects model, one could purposely and strategically select levels 1 vs. 2 vs. 3 mg of drug. The advantage of this approach is that once we obtain means, we can compare mean differences across the drug groups. The disadvantage of this approach is that we have no knowledge of differences between other possible levels of drug (e.g., 1.5 vs. 2.3, for instance). Is variance in headaches a function of these other levels as well? Under the fixed model comparing 1 vs. 2 vs. 3 mg, we simply do not know. The comparison of means does not allow us to infer such things. However, if instead we designate drug to be a *random effect*, and selected the levels 1, 2, 3 or others *randomly*, then if the factor is found to be statistically significant, we can conclude differences not only between doses 1, 2, and 3, but instead on *drug doses in general*, which includes levels not observed in the sample. This is the benefit of random effects; they allow us to conclude that levels not seen in the sample are also associated with *variation* in the response variable. *Random effects allow for a more general statement*.

When a model contains a blending of fixed and random factors, it is referred to as a *mixed model*. The classic general form of the linear mixed model is given as follows:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{U} + \boldsymbol{\varepsilon} \quad (4.14)$$

where  $\boldsymbol{\beta}$  are the fixed parameters and  $\mathbf{U}$  the random, and where  $\mathbf{U}$  and  $\boldsymbol{\varepsilon}$  are independent zero mean random vectors. Both the random effects in  $\mathbf{U}$  and  $\boldsymbol{\varepsilon}$  are assumed to be normally distributed. The assumption that  $\mathbf{U}$  is normally distributed may be unrealistic, however, and hence an alternative nonparametric random-effects model may be more appropriate. We survey that possibility a bit later in this section. For an excellent overview of both parametric and nonparametric random-effects and mixed models, see Nguéma et al. (2021).

In the model in (4.3), for fixed factors, mean comparisons are of interest. For the random factor, variance on the response variable is what is being modeled. Simple as this brief summary may sound, mixed models become theoretically complex very fast. In fact, books (and careers) are devoted to the subject. See Demidenko (2004) for a book-length treatment on the complexity of mixed models.

#### 4.3.1 The Meaning of a Fixed vs. Random Effect

A precise definition of fixed vs. random effects is critical. As we will discuss shortly, there is sometimes a misunderstanding in the literature that random effects are defined by simply having numerous statistics such as intercepts and slopes of which we would like to take an average. For example, consider Figure 4.2, where there are numerous intercepts which might represent starting points of trajectories for given individuals in a longitudinal study, as well as numerous slopes that emphasize a difference in trajectories over time. We see that there is quite a bit of variation in both intercepts and slopes in the sample.

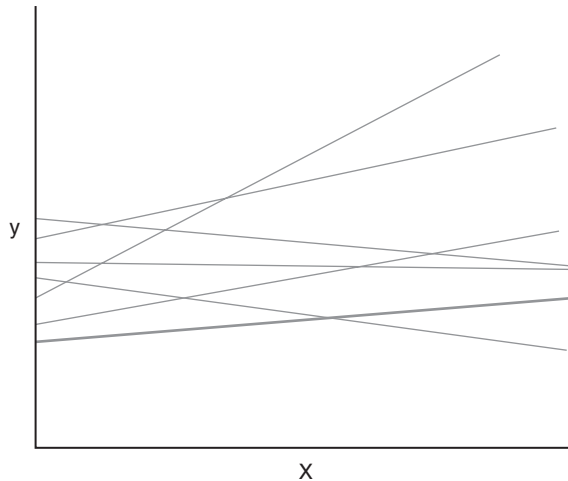


Figure 4.2 Plot with several intercepts and slopes, potentially representing different starting points and trajectories if the data were longitudinal.

What distinguishes an effect to be random or fixed in this case is *not the fact that we have many intercepts or slopes in our sample data*. Indeed, one could have a thousand different trajectories for different individuals and still identify the slope as a fixed effect. It may make more sense however to consider it random, but *that identification is not determined by the fact that we have numerous slopes. It is determined by the fact that we consider the slopes as randomly sampled from a wider population of slopes and thus having their own probability distribution*. This is the key component to considering an effect to be random, not the *volume* of variation in the sample. Whether to regard an effect as fixed or random is entirely a *modeling decision* and is embedded in the assumptions we make for the model. It is never “implicit” in the data, though that isn’t to say the data cannot “help” us make decisions on how to model the effect.

#### 4.3.2 Is the Fixed-Effects Model Actually Fixed? A Look at the Error Term

The reader may recall that the standard linear model assumes an error term that is normally distributed with finite variance. That is, in the standard linear model,  $\varepsilon_i \sim N(0, \sigma_\varepsilon^2)$  and  $\varepsilon_i, \varepsilon_{i'}$  are independent for  $i \neq i'$ . But what is the error term,  $\varepsilon_i$ , really? It is a random effect! Hence, when we are performing a fixed-effects ANOVA, we are, in reality, performing a *mixed model*. The random effect in the standard linear model is  $\varepsilon_i$ , the variance component is  $\sigma_\varepsilon^2$ , that is,  $V(\varepsilon_i)$ . As put so

well by Fox (2016, p. 701), “Unlike the standard linear model, linear mixed-effect models include more than one source of random variation – that is, more than one random effect.” And while this, logically, may be true, the custom is *not* to refer to fixed models as mixed models simply as a result of including an error term. By tradition, a mixed model is defined by having a blending of fixed and random factors beyond that of the error term. Our point is merely that if we look at these models in a bit of a deeper way, we come to realize that purely fixed-effects models actually do not exist unless we either fix the error term or leave it out entirely, which would be the same thing as setting it to zero.

### 4.3.3 Mixed Models in Python

We can fit a mixed model in Python using `statsmodels`. We adapt the following example from `statsmodels.org` where the growth curves of pigs are modeled. The response variable in the following is the *weight* of each pig, while the predictor is *time*. Hence, we wish to model weight as a function of time, but with the recognition that *data are nested (i.e., correlated) within each pig*. That is, measurements within each pig will be more similar than measurements between pigs. Thus, pig, which we will name “group” below, is a random effect:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
data = sm.datasets.get_rdataset("dietox", "geepack").data
data
Out [85]:
```

|     | Pig  | Evit    | Cu    | Litter | Start | Weight    | Feed       | Time |
|-----|------|---------|-------|--------|-------|-----------|------------|------|
| 0   | 4601 | Evit000 | Cu000 | 1      | 26.5  | 26.50000  | NaN        | 1    |
| 1   | 4601 | Evit000 | Cu000 | 1      | 26.5  | 27.59999  | 5.200005   | 2    |
| 2   | 4601 | Evit000 | Cu000 | 1      | 26.5  | 36.50000  | 17.600000  | 3    |
| 3   | 4601 | Evit000 | Cu000 | 1      | 26.5  | 40.29999  | 28.500000  | 4    |
| 4   | 4601 | Evit000 | Cu000 | 1      | 26.5  | 49.09998  | 45.200001  | 5    |
| ..  | ...  | ...     | ...   | ...    | ...   | ...       | ...        | ...  |
| 856 | 8442 | Evit000 | Cu175 | 24     | 25.7  | 73.19995  | 83.800003  | 8    |
| 857 | 8442 | Evit000 | Cu175 | 24     | 25.7  | 81.69995  | 99.800003  | 9    |
| 858 | 8442 | Evit000 | Cu175 | 24     | 25.7  | 90.29999  | 115.200001 | 10   |
| 859 | 8442 | Evit000 | Cu175 | 24     | 25.7  | 96.00000  | 133.200001 | 11   |
| 860 | 8442 | Evit000 | Cu175 | 24     | 25.7  | 103.50000 | 151.400002 | 12   |

```
[861 rows x 8 columns]
```

```
md = smf.mixedlm("Weight ~ Time", data, groups=data["Pig"])
```

Above we have specified a model of weight as a function of time, but nested within pig (groups = data["Pig"]). We now obtain a summary of the model:

```
mdf = md.fit()
print(mdf.summary())
Mixed Linear Model Regression Results
=====
Model:      MixedLM Dependent Variable: Weight
No. Observations: 861      Method:      REML
```

```

No. Groups:   72   Scale:   11.3669
Min. group size: 11   Log-Likelihood: -2404.7753
Max. group size: 12   Converged:   Yes
Mean group size: 12.0
-----
      Coef.   Std.Err.    z    P>|z| [0.025 0.975]
-----
Intercept   15.724    0.788   19.952   0.000  14.179  17.268
Time        6.943    0.033  207.939   0.000   6.877   7.008
Group Var   40.394    2.149
=====

```

We see from the output that time is a statistically significant predictor of the weight of pigs ( $p < 0.001$ ). We can also see that there is variance due to pig (i.e., group) equal to 40.394 (Group Var = 40.394). We also see from the output that the model was fit by REML (*restricted maximum likelihood*), which is usually the preferred method of estimation for a host of models.

#### 4.3.4 Mixed Models in R

We now run the same mixed model on the pig data using `lmer()` in *lme4* (Bates et al., 2015). The package *lme4* has extensive mixed modeling capacities and is highly recommended for all things mixed models. We also load the package *geepack* (Hojsgaard et al., 2006) which contains the pig data:

```

> library(lme4)
> library(geepack)
> print(summary(lmer('Weight ~ Time + (1|Pig)', data=dietox)))

```

Note that the designation `(1|Pig)` communicates the nesting or *hierarchical* structure of the data, such that weight is being modeled as a function of time, but nested within pig, where pig is a random effect, which implies in this case that it is a level 2 variable and that we wish to model the variance of its residuals. That is, our output should return a variance due to pig, since the model holds that pigs were randomly sampled from a larger population of *potential* pigs that could have appeared in the sample.

We now view output from the model:

```

Linear mixed model fit by REML ['lmerMod']
Formula: Weight ~ Time + (1 | Pig)
Data: dietox
REML criterion at convergence: 4810
Scaled residuals:
   Min    1Q  Median    3Q   Max
-4.712 -0.570 -0.094  0.488  4.773
Random effects:
Groups   Name      Variance Std.Dev.
Pig      (Intercept)  40.4    6.36
Residual 11.4      3.37
Number of obs: 861, groups: Pig, 72
Fixed effects:

```

```

      Estimate Std.   Error   t value
(Intercept)  15.7235   0.7881   19.9
Time        6.9425   0.0334   207.9
Correlation of Fixed Effects:
  (Intr)
Time -0.275

```

We note that the output is the same as we obtained in Python, only the group effect is given under its own random effects category (i.e., “Pig”), along with the residual variance (11.4) just below it.

The package *nlme* (Pinheiro et al., 2023) can also be used to fit mixed models, as well as the *VCA* package (Schuetzenmeister and Dufey, 2024). R’s capacity to fit increasingly complex mixed models is virtually unlimited using these and related packages in R.

## 4.4 Multilevel Modeling

The standard linear model typically estimated by least-squares regression is appropriate for independent observations, which implies that errors will also typically be *independent*. That is, it is assumed that  $\varepsilon_i, \varepsilon_{i'}$  are independent for  $i \neq i'$ . The standard linear model is usually not appropriate, or at minimum severely *compromised*, when this assumption is violated.

One type of dependency that naturally arises in applied settings is that of *clustering*, where  $\varepsilon_i, \varepsilon_{i'}$  are not independent for  $i \neq i'$ . A classic example of where this type of clustering or dependency surfaces is in classrooms within a school, where students may share the same teacher. We say *students are nested within classrooms*. It would be unreasonable to expect that errors within classrooms are *not* related. However, this is only one potential source of nesting in the hierarchy that we may consider. Classrooms exist in schools, and schools exist in states. That is, schools within the same state, *and their errors*, are likely not independent of one another. As we build the nesting hierarchy from scores to groups, we develop a *multilevel structure to observations*, thus the name *multilevel modeling*. Multilevel modeling allows one to deal with the non-independence of errors that manifest at each level of the hierarchy.

### 4.4.1 A Garbled Mess of Jargon

The jargon and confusion of words and terminology that surrounds random, mixed, and multilevel modeling is nothing short of extraordinary. Historically, statistics is a discipline that has always had a difficult time having consistency in how it defines its terms because many concepts arise independently in different applied disciplines. Mixed models may arise independently in ecology, for instance, while also being developed (or “rediscovered”) in sociology as well. Slightly different names may be given to the models, which makes learning (and teaching) them very difficult for the newcomer. It’s as if in addition to learning the rather difficult method one has to become keen on how the *vocabulary* differs by area of specialization. This not only applies to mixed models, it applies to many statistical techniques; jargon can easily ruin your day in attempting to learn them.

It is exceptionally important however when discussing these models to be very exact and precise regarding what we are referring to. Multilevel models are often casually referred to as *hierarchical linear models*, *random-effects models*, *mixed models*, and other identifiers, and it can become greatly confusing for the newcomer to these models to appreciate or understand what these models are all about beyond these generalities.

First, we need to make a distinction between *structure* vs. *model*. The term “multilevel” is a *structure*, it is not actually a “model,” though it is referred to as such in the literature (i.e., *multilevel modeling*), for reasons that will become clear shortly. It is best to reserve the word “multilevel” to describe structure, though in what follows we ourselves at times adopt the conventional phrase “multilevel modeling” as well and make no distinction. So long as one understands and appreciates the distinction, no harm is done. However, to the newcomer to multilevel models, a great deal of confusion can arise from not understanding what a *structure* is compared to a *model*. As we will see, multilevel structures can be analyzed in a host of different ways, only one of which is via random and mixed effects.

Second, *a multilevel structure need not necessarily imply a mixed model, nor does a multilevel structure need to incorporate any random effects*. They *usually* do include such random effects, but the definition of what is a multilevel (structure) does not absolutely require it. In reading the literature, however, the newcomer to these models is rarely made aware of this. Multilevel models are automatically assumed to consist of random effects. This implied association is not a necessary one. *As we will see, for a multilevel model to have one or more random effects (other than the naturally occurring error terms), this must be assumed or imposed by the researcher. No multilevel structure, by itself, necessarily implies or must otherwise contain random effects.*

As an example, consider the classic case once more of ten students per classroom in which there are five classrooms. Students are nested within classrooms, as students in one classroom will likely share features or common experiences over students in another classroom. They will be more *similar* than students in a competing classroom. Statistically, error terms of students within classroom will not be independent of one another. *We can assume their errors will be correlated*. In practice, if we desired, we could account for the multilevel *structure* by calculating separate regressions on each classroom. We could then compare the regressions by way of computing a product term that incorporates classroom. Notice that there is no random effect implied here and yet we are dealing with the multilevel structure just the same. *The multilevel structure is still intact without requiring any random effects*. It may not be the most “ideal” of solutions in modeling the multilevel structure, but the point is that addressing the multilevel structure does not *necessarily* imply, nor require, the modeling of random effects.

The purest or most accurate definition of a multilevel model often given or implied in the literature is that it *allows for residual analyses at each level of the hierarchy*. And while this is true, there is nothing in a multilevel *structure* that absolutely requires this level of analysis. The emphasis should be on “allows” in the previous phrase, not “necessitates.” Again, notice that we could quite easily analyze each regression or mean structure separately by classroom and assess residuals within each classroom. We do not *require* any random effects to do so. Including random effects, however, is often a very good idea, as we now discuss.

#### 4.4.2 Why Do Multilevel Models Often Include Random Effects?

If a multilevel *structure* by itself does not necessarily imply the inclusion of random effects, how do random effects therefore come to rise in a multilevel model? The answer is that the researcher has to *impose* them on the multilevel structure through an assumption of the modeling. Take note again that *we are imposing the random effect(s); they are not necessarily implied in the multilevel structure*. Again, and referring to our previous example, *observed variance in intercepts or slopes in exploratory analysis of data cannot serve as justification alone to model such as random effects*.

To emphasize this important point, consider an excellent description of multilevel modeling by one of the chief leaders in this area:

*Given that the intercept and slope coefficients vary randomly over group ...*

The above is completely correct, but what does it mean? The key point to recognize from the above is that it is *us* who decides whether such coefficients “vary randomly” or not. The fact that there is *observed variation* in them is not enough. We have to decide to model the intercept and slope as fixed or random based on the *assumptions* we are imposing on the model. If there is no observed variation, we probably would not model them as random effects, but in theory, we still could.

Consider another typical presentation of the multilevel model:

*Because multilevel models contain a mix of fixed effects and random effects, they are sometimes known as mixed-effects models.*

The problem with the above is that, as mentioned, not only do multilevel structures *not* necessarily require the inclusion of random effects, but that even if they did, they do not *need* to be modeled using a *traditional* mixed-effects modeling approach. A structural equation modeling approach, for instance, could be just as fruitful. Once more, *a multilevel structure does not necessarily imply a fixed, random, or mixed model.*

In summary then, the intercepts and slopes typically found in hierarchical (or “nested” designs) do not know where they “came from.” We have to make those decisions as part of the assumptions we are imposing on our analysis. In writings about multilevel models, authors too often imply that because there is observed variation in intercepts and slopes, it necessarily implies the corresponding effect should be modeled as a random effect. This is simply not the case. Observed variation in statistics, whatever they may be (e.g., intercepts, slopes, means, standard deviations, etc.), is insufficient reason for modeling such as random effects. The decision to model as a random effect has all to do with how the observed realizations are *conceptualized to have originated and how such statistics would likely behave in theoretical repetitions of the experiment*. These are decisions made by the investigator.

*A multilevel structure does not necessarily imply the presence of random effects. Random effects arise in such structures because the researcher decides to model them (intercepts and slopes) as such, but a multilevel structure need not necessarily imply the inclusion of these effects. Intercepts and slopes could, in theory, even when exhibiting much variation, be estimated as fixed effects.*

#### 4.4.3 A Priori vs. Post-Hoc “Nesting”

There is another important and key distinction to make with regard to multilevel ideas and nesting. In appreciating what a multilevel structure actually *is*, it is just as pedagogical to consider what a multilevel structure is *not*. Consider the classic one-way randomized analysis of variance design in which participants ( $n = 5$  per sample, e.g.,  $y_{11}$  is observation 1 in sample 1) are randomly assigned to treatments (Figure 4.3).



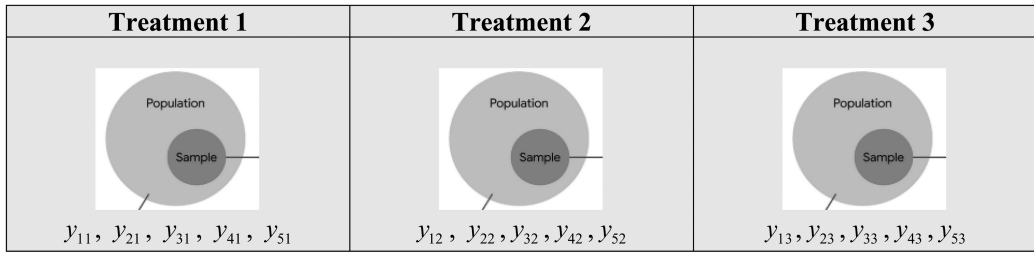


Figure 4.3 Classic one-way randomized analysis of variance layout. Each sample is drawn from the same population and observations are randomly assigned to different treatments.

Are participants nested within treatment groups? Are the errors within treatment groups expected to be related? *No*. Because of the random assignment to treatment groups, an individual's score *a priori* is not expected to be related to any other individual's score. Why is the qualifier "*a priori*" essential here? It is important because *after they undergo treatment*, we might indeed observe their errors to be related. For example, if being in the first treatment group involves ingesting a particular dosage of an energy drink, whereas other groups did not receive any energy drink, we would of course expect errors to be related within that first group *relative to other groups*. Those drinking energy drinks will become more "similar" due to the treatment they have received. They are now a "nest" due to receiving the treatment.

However, the above similarity is only apparent *after* the treatment has been administered, not *before*. Before receiving treatment, there was no reason to expect that the score of one participant would be related to the score (and error) of a second participant within that group. Participants in this case are not nested *a priori*, but once the treatment is applied, they are, in a sense, nested *post-hoc*. But this *post-hoc* nesting idea is immaterial to multilevel modeling. It would only play a role if we decided to build on the current study to perform a second one and used the treatment groups as starting points. In such a case, those drinking energy drinks would start out the new study having correlated errors. That is, those receiving the treatment would be more "alike" compared to others not receiving the treatment.

The key takeaway from this discussion is that for hierarchical, multilevel, or nesting structures to exist, errors should be *expected* to be related *prior* to receiving treatment, not *after*. Students in a classroom being taught by the same teacher are *expected* to have related errors well before they receive any treatment or intervention. This is the "likeness" that we wish to account for in multilevel structures. While the classic one-way ANOVA (for instance) can actually be considered a multilevel structure (the reader may want to think of how), the errors within each treatment group are expected to be *independent* even if they become correlated once the treatment is applied and measurement occurs. What is most important for modeling decisions such as this is the *expectation* of how errors will behave *prior* to treatment, not *after*.

#### 4.4.4 Blocking as an Example of Hierarchical/Multilevel Structure

When a researcher forces observations into blocks, the so-called *randomized block design*, this is an example and special case of a multilevel structure. *Observations are nested within blocks*. The nesting is *induced* or *created* by the researcher. For instance, if we put all first-grade school

children in one block and all second graders in another block, it stands that observations within each respective block will be more similar than observations across blocks. The observations are level-one units, while the blocks are level two. For this reason, blocking designs are often analyzed as mixed models where the blocking factor is regarded as random. For readers to whom blocking designs may be entirely new, Kirk (1995) provides a good and deep introduction.

#### 4.4.5 *Nonparametric Random-Effects Model*

Most multilevel models include at least one random effect that aids in dealing with the nesting structure and dependencies of observations. In the classic random-effects model, it is assumed that random effects are normally distributed. This assumption, however, may be untenable and we may wish to adopt instead, as we will be doing in many places in this book, a *nonparametric* approach that makes no (or at least *fewer*) assumptions about the distribution of the random effects. The *nonparametric random-effects model* can be considered a special case of the wider *latent class model*. The random-effects regression model is also commonly known as a *mixture model in regression* since it combines a mixture of regressions that helps deal with the potential heterogeneity of groups (Jiang and Guntuboyina, 2021).

As an easy example of a nonparametric random-effects model and to get an idea of how it differs from the classic parametric random-effects model, consider first the typical (parametric) random-intercept model that contains no predictors,

$$g[E(y_{ij})] = \alpha_j$$

where it is assumed that the intercept terms  $\alpha_j$  are normally distributed with a given mean and variance, that is,  $\alpha_j \sim N(\mu, \sigma^2)$ . Traditionally, we would like to estimate the mean  $\mu$  of these intercepts and more importantly their variance ( $\sigma^2$ ). The approach is *parametric* because we are imposing a *probability model* on the intercepts. In a nonparametric approach, we assume that the distribution of  $\alpha_j$  is indexed by an unspecified discrete mixing distribution of *latent classes*, where the intercept associated with a given class  $x$  is denoted by  $\alpha_x$  and the size of the given class is given by  $P(x)$ , where  $\alpha_x$  is the location and  $P(x)$  the “weight” of the given node or class.

Simplified, the nonparametric model is given by

$$g[E(y_{ij} | x)] = \alpha_x \tag{4.15}$$

which is simply the expectation of  $y_{ij}$  for a given class  $x$ , which will be subject to  $P(x)$ . A two-level model in which predictors are included is a natural extension of (4.15), where now  $\mathbf{z}_{ij}$  denotes the inclusion of predictors into the model:

$$g[E(y_{ij} | \mathbf{z}_{ij}, x)] = \alpha_x + \sum_{k=1}^K \beta_k \cdot z_{ijk} \tag{4.16}$$

Key to note is that what makes the model nonparametric is not the inclusion of predictors in  $\mathbf{z}_{ij}$ , but as was the case in the intercept-only model, the fact that  $y_{ij}$  is a function of a mixing distribution of classes,  $x$ . To reiterate,  $E(y_{ij})$  is a function of a *probability distribution* in the classic parametric model, whereas it is a function of a *mixing distribution* (classes) in the nonparametric counterpart. Random slopes, in addition to random intercepts, can also be included, giving rise to a full-scale *multilevel nonparametric random-effects model*.

Multilevel models, as was generally the case for mixed models, are best implemented using R (e.g., *lme4*) software instead of Python. *Latent Gold* is a software program that can estimate latent class models and the special case of nonparametric random-effects models. In R, the package *npmlreg* (Einbeck et al., 2018) can be used to fit these types of models as well.

## 4.5 Repeated-Measures and Longitudinal Models

Often in research studies, measurements are taken over a series of days, weeks, months, or years, or even *epochs*, as in the case of attempting to estimate whether today's rising temperatures are just normal long-term temperature variation versus human-made climate change. Models that involve a temporal component are often called *repeated-measures models* or *within-subjects designs*, or, more generally, *longitudinal models* or *time series*. The reader should be aware, however, that longitudinal models can comprise a whole host of models and complexities which can include mixed-effects components as well as elements of structural equation models. As noted earlier, jargon and terminology are often not consistent from discipline to discipline, and hence it behooves the reader to be cautious when coming across these and similar terms to learn exactly what the author or researcher means by them. Making assumptions as to what they mean will typically only lead to misunderstandings!

### 4.5.1 Classic Repeated-Measures Models

For lack of a better terminology, and to differentiate among repeated-measures approaches, we will refer to the repeated measures that typically assumes a *sphericity structure* for the covariance matrix to be that of *classic repeated measures*. The assumption of sphericity is essentially that of equal variances among differences between all possible pairs of within-subject conditions. An even more stringent assumption is that of *compound symmetry* that assumes all variances and covariances in the matrix of within-subject variables are equal. Both of these assumptions give rise to what we are calling the “classic” repeated-measures model to distinguish it from more sophisticated models such as random-effects, mixed-effects, and structural equation longitudinal models and growth curves. *Latent growth models* are a sub-type of the more general *structural equation model* in which the latent variables are typically such things as intercepts and slopes that “give rise” to time points.

Just as we did for classic ANOVA, we assume the reader has had at least some exposure to basic repeated measures under the assumption of a spherical covariance matrix. Hence, we move quickly and simply provide the expected mean squares for the classic model. There is a lot of theory and explanation that goes along with these expected mean squares and for readers unfamiliar, Hays (1994) and Denis (2021a) are good sources that explain how these expected mean squares arise, including how MS residual is a function of the intraclass coefficient  $\rho$  below.

The expected mean squares for the classic additive “sphericity” repeated-measures model are thus:

$$E(\text{MS}_{\text{subjects}}) = \sigma_e^2 + K\sigma_{\text{subjects}}^2 \quad (4.17)$$

$$E(\text{MS}_{\text{treatment}}) = \sigma_e^2 + \frac{\sum_j \alpha_j^2}{K-1} \quad (4.18)$$

$$E(\text{MS}_{\text{residual}}) = \sigma_e^2 = \sigma_T^2(1 - \rho) \quad (4.19)$$

where  $J$  is the number of rows for subjects,  $K$  is the number of columns for treatment,  $\rho$  is the intraclass coefficient, and  $\sigma_T^2$  and  $\sigma_e^2$  are the total and error variances, respectively.

There are numerous ways of fitting repeated-measures models in R. The function `aov()` in R has easy capabilities for fitting such models (see Denis, 2021a for details). Many packages are also devoted to fitting repeated-measures or longitudinal models, for example the *mmrm* package (Sabanés et al., 2024). Repeated-measures models can also be fit in a variety of ways in Python. Probably the easiest way is via `statsmodels` using `AnovaRM`. The following is sample code (output not shown) for a basic one-way repeated measures where the dependent variable is `y`, the subject variable is `person`, and the repeated measure is `time`:

```
from statsmodels.stats.anova import AnovaRM
print(AnovaRM(data=df, depvar='y', subject='person', within=['time']).
fit())
```

#### 4.6 Multivariate Analysis of Variance (MANOVA)

Univariate ANOVA allows for more than a single independent variable or factor but only allows for a single response or dependent variable. The name *univariate* derives from this fact, that of a single response. When there exists more than a single response variable, we enter into *multivariate* territory. Multivariate analysis of variance, or “MANOVA” for short, is defined by having more than a single continuous response variable.

The multivariate hypothesis about mean vectors can be stated as follows,

$$H_0 : \begin{pmatrix} \mu_{11} \\ \mu_{21} \\ \mu_{31} \\ \vdots \\ \mu_{j1} \end{pmatrix} = \begin{pmatrix} \mu_{12} \\ \mu_{22} \\ \mu_{32} \\ \vdots \\ \mu_{j2} \end{pmatrix} = \begin{pmatrix} \mu_{13} \\ \mu_{23} \\ \mu_{33} \\ \vdots \\ \mu_{j3} \end{pmatrix} \dots = \begin{pmatrix} \mu_{1p} \\ \mu_{2p} \\ \mu_{3p} \\ \vdots \\ \mu_{jp} \end{pmatrix} \quad (4.20)$$

which is evaluated against an alternative hypothesis that somewhere among the set of mean vectors is a difference, for example of one possibility, that of a mean vector difference between the first two groups (inequality statement is slightly informal here),

$$H_1 : \begin{pmatrix} \mu_{11} \\ \mu_{21} \\ \mu_{31} \\ \vdots \\ \mu_{j1} \end{pmatrix} \neq \begin{pmatrix} \mu_{12} \\ \mu_{22} \\ \mu_{32} \\ \vdots \\ \mu_{j2} \end{pmatrix} = \begin{pmatrix} \mu_{13} \\ \mu_{23} \\ \mu_{33} \\ \vdots \\ \mu_{j3} \end{pmatrix} \dots = \begin{pmatrix} \mu_{1p} \\ \mu_{2p} \\ \mu_{3p} \\ \vdots \\ \mu_{jp} \end{pmatrix} \quad (4.21)$$

#### 4.6.1 Suitability of MANOVA

When is it appropriate to perform a MANOVA over an ANOVA? The answer cannot simply be “when there is more than a single continuous response,” despite misleading cookbook approaches to data analysis sometimes advising on such a poor strategy. Indeed, analyzing sex differences on response variables IQ and amount of gasoline consumed in one’s vehicle in the past week would make little sense for the reason that the response “variate,” that is, the combination of variables on the response side, makes little if any substantive “sense.” Can it be analyzed? Yes, but on an applied level, is not a hypothesis we would likely be interested in. The key to MANOVA is not to analyze multiple responses simultaneously; it is to analyze multiple responses that in some sense “go together” to produce a *variate*. Simultaneous analysis of reading speed and grade point average would be a variate of more interest in an applied setting. Or, another example might be skull measurements as a function of prehistoric epoch (Anderson, 2003). The takeaway point is that when one combines multiple responses into a single model, one should have a *reason* for doing so over and above simply producing a multivariate model.

With that substantive rationale and caveat in place, there are *statistical* benefits to conducting a MANOVA over multiple ANOVAs, which include control over an inflating type I error rate. For instance, three different univariate ANOVAs conducted separately carries an increasingly large type I error rate (e.g., 0.05 per test,  $p = 0.05$ ). The probability of making *no type I errors* ( $r = 0$ ) across  $n = 3$  tests each set at 0.05 is a binomial random variable,

$$p(r) = \binom{n}{r} p^r (1-p)^{n-r}$$

$$p(0, 3, 0.05) = \binom{3}{0} (0.05)^0 (0.95)^3 = 0.86 \quad (4.22)$$

and hence the probability of at least one type I error is equal to  $1 - 0.86 = 0.14$ . More formally, the probability of *at least one type I error* across  $k$ -tests can be approximated by

$$p(\text{type I error} \geq 1) = 1 - (1 - \alpha_{pc})^k \quad (4.23)$$

A single MANOVA on the three response variables conducted simultaneously keeps the type I error rate fixed at whatever  $\alpha$  is set to for the MANOVA (e.g., 0.05). This is a *statistical* benefit to performing a MANOVA over several ANOVAs; however, if substantively the MANOVA does not make sense, as discussed, a MANOVA model makes no sense to conduct for statistical reasons alone. *The scientific or substantive rationale should dominate over the statistical here.*

#### 4.6.2 Extending the Univariate Model: Hotelling’s $T^2$

The easiest and most direct way of extending the univariate model to the multivariate one is via the simple between-subjects  $t$ -test and generalization to Hotelling’s  $T^2$ , which is the multivariate counterpart where we model more than a single continuous response variable. This essentially will generalize to more complex models, but we focus on the  $t$ -test because it emphasizes the core principles in going from a univariate to multivariate landscape.

Recall the univariate  $t$ -test discussed earlier in our review of essential statistics:

$$\begin{aligned}
t &= \frac{E(\bar{y}_1) - E(\bar{y}_2) - \delta_0}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}} \\
&= \frac{\mu_1 - \mu_2 - \delta_0}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}}
\end{aligned}$$

Consider now what happens when we square  $t$ ,

$$t^2 = \frac{(\bar{y}_1 - \bar{y}_2)^2}{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}$$

Next, we express the denominator as an inverse so that there is no denominator:

$$t^2 = (\bar{y}_1 - \bar{y}_2) \left[ \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \left( \frac{1}{n_1} + \frac{1}{n_2} \right) \right]^{-1} (\bar{y}_1 - \bar{y}_2)$$

Finally, we end up with

$$T^2 = \frac{n_1 n_2}{n_1 + n_2} (\bar{\mathbf{y}}_1 - \bar{\mathbf{y}}_2)' (\mathbf{S}^{-1}) (\bar{\mathbf{y}}_1 - \bar{\mathbf{y}}_2) \quad (4.24)$$

For the one-sample mean vector, the corresponding  $T^2$  is the following:

$$\begin{aligned}
T^2 &= (\bar{\mathbf{y}} - \boldsymbol{\mu}_0)' \left( \frac{1}{n} \mathbf{S} \right)^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}_0) \\
&= n(\bar{\mathbf{y}} - \boldsymbol{\mu}_0)' \mathbf{S}^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}_0)
\end{aligned} \quad (4.25)$$

$T^2$  is distributed as  $\frac{(n-1)p}{(n-p)} F_{p, n-p}$ , where  $p$  and  $n-p$  are degrees of freedom, where  $p$  is the number of variables and  $n$  is sample size. For details on likelihood ratio tests in relation to  $T^2$ , see Johnson and Wichern (2007, pp. 217–218). Confidence regions can also be computed for  $T^2$  and multivariate populations (see Johnson & Wicher, 2007, pp. 220–221).

#### 4.6.3 *Multivariate Test Statistics*

In the univariate ANOVA setting, there is only a single  $F$ -test to perform. In the multivariate setting, because of the alternative way covariances of response variables may be structured within groups on the independent variable(s), multiple multivariate tests have been developed and are available in virtually all software. These include *Wilk's lambda*, *Pillai's trace*, *Roy's largest root*, and the *Lawley-Hotelling's trace*. We provide only a cursory “glimpse” into the variety of multivariate tests. For a full development of how these tests come about, see Johnson and Wichern (2007).

Wilk's lambda,  $\Lambda$ , is given by

$$\Lambda = \frac{|\mathbf{E}|}{|\mathbf{H} + \mathbf{E}|} = \frac{|\mathbf{E}|}{|\mathbf{T}|} \quad (4.26)$$

where under the null hypothesis,

$$\Lambda = \frac{|\mathbf{E}|}{|\mathbf{0} + \mathbf{E}|} = \frac{|\mathbf{E}|}{|\mathbf{E}|} = 1.0$$

and where  $\mathbf{H}$  is the “hypothesis” matrix and  $\mathbf{E}$  the “error” matrix. Note that Wilk's is an *inverse criterion*, meaning that if there is no effect, we would expect  $\Lambda$  to equal 1.0. This is unlike the *F*-ratio in univariate ANOVA, where larger *F*'s are preferred over smaller.

On the opposite extreme, if  $\mathbf{H}$  accounts for all of the variation, then,

$$\Lambda = \frac{|\mathbf{0}|}{|\mathbf{H} + \mathbf{0}|} = \frac{|\mathbf{0}|}{|\mathbf{H}|} = 0$$

Wilk's lambda can also be expressed in terms of eigenvalues of  $\mathbf{E}^{-1}\mathbf{H}$ :

$$\Lambda = \prod_{i=1}^s \frac{1}{1 + \lambda_i} \quad (4.27)$$

where  $\lambda_i$  are the eigenvalues extracted from the analysis.

Pillai's trace is defined as

$$V^{(s)} = \text{tr}[(\mathbf{E} + \mathbf{H})^{-1}\mathbf{H}] = \sum_{i=1}^s \frac{\lambda_i}{1 + \lambda_i} \quad (4.28)$$

Note that Pillai's uses all eigenvalues (often called “roots”),  $\lambda_i$ . If we use only the *largest* of these roots, this gives rise to *Roy's largest root*. That is, Roy's largest root is simply the largest eigenvalue extracted and hence represents only the largest dimension underlying the data. Roy's largest root is especially useful if one dimension dominates all others, which translates into one eigenvalue being much larger than the others. When the distribution of eigenvalues is more spread out, Pillai's is more appropriate since presumably more than a single dimension is of interest.

Lastly, Lawley-Hotelling's trace is simply a sum of eigenvalues and is thus given by

$$U^{(s)} = \text{tr}(\mathbf{E}^{-1}\mathbf{H}) = \sum_{i=1}^s \lambda_i \quad (4.29)$$

#### 4.6.4 Evaluating Equality of Covariance Matrices: The Box-M Test

Recall that in univariate ANOVA, we were required to evaluate the assumption that all variances were equal. In the multivariate counterpart, we not only have variances to deal with; we also have *covariances*. Hence, the assumption now includes that all covariances across levels (or cells) of the independent variable are equal, that is,

$$H_0 : \Sigma_1 = \Sigma_2 = \Sigma_3 \dots = \Sigma_p \quad (4.30)$$

where  $\Sigma_p$  are the  $p$  covariance matrices for each population under levels of the independent variable.

There are different approaches to evaluating this assumption. One is simply a crude (yet still useful) visual approach to learn whether the covariance matrices are relatively similar to one another to deem their population counterparts at least approximately equal and proceed with the MANOVA. The *Box-M test* is another approach that evaluates this assumption and is available in R in the *biotools* package (da Silva, 2021) under the `boxM` test, which can be easily demonstrated directly from the package on the iris data:

```
> library(biotools)
> data(iris)
> boxM(iris[, -5], iris[, 5])
Box M-test for Homogeneity of Covariance Matrices
data: iris[, -5]
Chi-Sq (approx.) = 140.94, df = 20, p-value < 2.2e-16
```

The assumption under the null is that population covariance matrices are equal. For these data, we reject the null ( $p = 2.2e-16$ ) and infer the alternative hypothesis that at least two covariance matrices differ. When covariance matrices differ, it behooves the researcher to have a look at such matrices and to consider substantive reasons as to why they may differ. *Never obtain a statistical result without carefully considering what it might imply about your data on a scientific, not only statistical, level.* For additional theory on the test, see Johnson and Wichern (2007).

#### 4.6.5 MANOVA in Python

We can conduct a basic MANOVA in Python using `statsmodels`. In the following we conduct a MANOVA on some random data where we generate 20 samples (`n_samples = 20`) on five responses (`n_dim = 5`) with three levels on an independent variable (`n_classes = 3`):

```
import numpy as np
from statsmodels.multivariate.manova import MANOVA
n_samples = 20
n_dim = 5
n_classes = 3
```

We produce the random data using `np.random.randn()` where  $X$  is the independent variable (or “exogenous” variable) side of the equation and  $y$  the response variable (or “endogenous” variable):

```
y = np.random.randn(n_samples, n_dim)
X = np.random.randint(n_classes, size=n_samples)
```

Note that the number of classes for  $X$  (i.e., 0, 2, 0, 0, 0, 1, 2, etc.) is designated to equal the size of the `n_samples` specified in `y`, so that each case gets assigned a class (we print only a few cases for `y`):



```

y
Out [31]:
array([[ -0.68321194, -1.17229168,  0.86179734,  1.26790702,  0.53352012],
       [ -0.95366322, -1.06883607,  1.13404045, -0.71916984, -1.37089659],
       [  0.18602881, -0.22453474,  0.02700791,  0.17279279,  0.13939027],
       [ -1.45080321, -0.20265485,  0.57581133, -0.39974093,  0.93340395],
       ...])
X
Out [33]: array([0, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0, 2, 1, 0, 1, 2, 1, 0, 0])

```

In the following we specify the endogenous variable (i.e., the *response* variable) as well as the exogenous variable (the *independent* variable):

```

manova = MANOVA(endog=y, exog=X)
print(manova.mv_test())

```

```

-----
              x0          Value  Num DF  Den DF   F Value   Pr > F
-----
Wilks' lambda      0.7224  5.0000  15.0000   1.1526   0.3765
Pillai's trace      0.2776  5.0000  15.0000   1.1526   0.3765
Hotelling-Lawley trace 0.3842  5.0000  15.0000   1.1526   0.3765
Roy's greatest root  0.3842  5.0000  15.0000   1.1526   0.3765
=====

```

We see that none of the multivariate tests reject the null having  $p$ -values equal to 0.3765, and hence we do not have sufficient evidence to reject the null hypothesis of equality of mean vectors.

#### 4.6.6 MANOVA in R

Carrying out a MANOVA in R can be done in a variety of ways via several packages. In the following we conduct a MANOVA on the iris data. We construct the dependent variable “variate” by *concatenating* (i.e., joining) the iris features and identify species as the independent variable:

```

> data(iris)
> dv <- cbind(iris$Sepal.Length, iris$Sepal.Width, iris$Petal.Length,
iris$Petal.Width)
> iv <- iris$Species
> manova <- manova(dv ~ iv, data = iris)
> summary(manova)

Df Pillai approx F num Df den Df Pr(>F)
iv 2 1.1919 53.466 8 290 < 2.2e-16 ***
Residuals 147
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

We see that species yields a *statistically* (not necessarily *substantively*) significant effect ( $p < 0.001$ ).

To obtain effect sizes, we can use the package *effect size* (Ben-Shachar, 2020) to obtain a partial Eta-squared statistic:

```
> library(effectsize)
> effectsize::eta_squared(manova)
# Effect Size for ANOVA (Type I)
Parameter | Eta2 (partial) | 95% CI
-----
iv | 0.60 | [0.54, 1.00]
- One-sided CIs: upper bound fixed at [1.00].>
```

We see that the partial Eta squared associated with the *iv* is 0.60, meaning that species explains approximately 60% of the variance in the response variable *variate*, that is, the linear combination of sepal length, sepal width, etc. Next, we conduct a discriminant analysis on species, a kind of follow-up analysis to MANOVA in the way of generating discriminant functions:

```
> library(MASS)
> discrim <- lda(iv ~ dv, CV=F)
> discrim
```

Designating CV=F instructs not to perform leave-one-out cross-validation (CV = T will provide such validation).

```
Call:
lda(iv ~ dv, CV = F)
Prior probabilities of groups:
setosa versicolor virginica
0.3333333 0.3333333 0.3333333
Group means:

      dv1    dv2    dv3    dv4
setosa  5.006  3.428  1.462  0.246
versicolor 5.936  2.770  4.260  1.326
virginica  6.588  2.974  5.552  2.026
```

Coefficients of linear discriminants:

```
      LD1      LD2
dv1  0.8293776 -0.02410215
dv2  1.5344731 -2.16452123
dv3 -2.2012117  0.93192121
dv4 -2.8104603 -2.83918785
```

```
Proportion of trace:
LD1 LD2
0.9912 0.0088
```

We can see that the first discriminant function accounts for a much larger proportion of the trace (0.9912) than the second (0.0088), which implies the first function is more “important” statistically than the second and better differentiates flower groups (i.e., the function has a greater eigenvalue). We are also given coefficients of the linear discriminants. But why are we seeing a discriminant analysis here? Though we introduce and discuss discriminant analysis in more detail later in the book as a *classifier* (if you have never studied discriminant analysis, you may choose to skip to that section now), it behooves us to briefly discuss here how discriminant analysis can be considered the “reverse” of the multivariate analysis of variance.

#### 4.7 Linear Discriminant Analysis (as the “Reverse” of MANOVA)

The linear discriminant analysis model (that we have just briefly seen in our MANOVA example) features a binary or polytomous categorical response variable in which a linear combination of predictors is sought that maximizes group differences on the response. Discriminant analysis is sometimes referred to as a *filter model* and a *feature selection method* in the data mining literature (Aggarwal, 2015). While at first glance, this may be challenging for the reader to grasp or “see” the big picture, the LDA model can actually be regarded as a “hub” model, that is, a model from which many other models can be considered *scaled-up* or *scaled-down* versions of the LDA model structure.

For one, it can be operationalized in a few different ways, including being the “opposite” of the MANOVA model. This is the interpretation that most interests us here to supplement our discussion of MANOVA and also understand the above output provided in R by the MANOVA follow-up analysis or, crudely, “*post-hoc*” analysis (though it is not strictly a *post-hoc test* as we are not attempting to control for any error rates, we are using the word “*post-hoc*” here merely *descriptively*). MANOVA and LDA are often conceived as *opposites* of one another, and many times, a discriminant analysis will be performed as a follow-up to MANOVA to help learn more about the underlying function(s) (called the *discriminant function(s)*) responsible for separating groups so efficiently.

In the above output, we came across via our *post-hoc* of MANOVA a set of discriminant functions (technically, they are not discriminant *functions* yet until we weight the corresponding coefficients with the variables, which is why R calls them *linear discriminants*):

Coefficients of linear discriminants:

|     | LD1        | LD2         |
|-----|------------|-------------|
| dv1 | 0.8293776  | -0.02410215 |
| dv2 | 1.5344731  | -2.16452123 |
| dv3 | -2.2012117 | 0.93192121  |
| dv4 | -2.8104603 | -2.83918785 |

A full understanding of what a discriminant function is will have to wait until we survey discriminant analysis later in the book. However, for now, it is enough to know that we have obtained two discriminant functions as a result of there being three groups in our data (i.e., type of species). LD1 above is nothing more than a linear combination (called *linear discriminant* or more generally *linear discriminant function* after weighting by variables) that is weighted with values of sepal length, sepal width, petal length and petal width (i.e., dv1 through dv4). The linear combination generated by LD1 is thus:

$$\text{LD1} = 0.82(\text{sepal length}) + 1.53(\text{sepal width}) + (-2.20)(\text{petal length}) + (-2.81)(\text{petal width}).$$

In the machine-learning community, LDA is hardly ever conceived as the opposite of MANOVA but instead simply as a “classifier” of group membership and as a similar and potential alternative to logistic regression. However, even more generally, both LDA and logistic can be interpreted as special cases of a wider model, that of the *generalized linear model*, which we discuss later in the book. Under this umbrella, LDA is an application of the generalized linear model in which the response is (typically) binary, yet the model is still *linear*. That is, in this context, the *link function* is simply the *identity function*. This serves to contrast the LDA model with that of the logistic in which the link function is the *log of the odds*. For readers coming to this discussion with some background, this likely makes perfect sense. Without such background however, what we have just said will be entirely confusing and the reader will have to wait until further in the book when we survey discriminant analysis and the generalized linear model in more detail. She may wish to return to this section after that discussion.

We “unpack” the LDA model later in the book where we treat it in much more detail as a classifier. For now, however, we simply wish to note that the unifying aspects of LDA and MANOVA cannot be overstated. Where MANOVA seeks to find mean vector differences, the LDA model seeks to use those linear combinations to distinguish between groups. Extensions and special cases of these models are numerous, and pedagogically, understanding discriminant analysis gives rise to a variety of related models. Indeed, it is not an exaggeration to say that we could, if desired, title an entire book called *Linear Discriminant Analysis and Variations* because of the “hub” nature of LDA.

#### 4.8 Chapter Review Exercises

1. Describe or explain how *analysis of variance* (ANOVA) gets its name. How does a null hypothesis about means translate into a comparison of variances?
2. As degrees of freedom for *SS within* increase, all else equal, what effect does this have on the ensuing *F* statistic in ANOVA?
3. What role does  $n_j$  play in the computation of  $\eta^2$  in ANOVA? Does it play a similar or different role than in the computation of the *F* statistic?
4. State a *contrast* in which the first mean is hypothesized to be different from means two and three. How will you verify that the coefficients you selected are correct?
5. In R, conduct an ANOVA on the iris data where sepal length is hypothesized to be a function of species. That is, sepal length is the response variable while species is the predictor or independent variable. Evaluate the null hypothesis using a significance level of .05. Then, in Python, perform the same analysis and compare output.
6. Distinguish between *fixed-effects*, *random-effects*, *mixed*, and *multilevel models*. How are they similar? How are they different? Be specific in your comparison.
7. How does a *nonparametric* random-effects model differ from a *parametric* random-effects model? What is the key difference?
8. What is the probability of making *no type I errors* across six tests if the significance level used for each test is .05?
9. When is *Pillai's trace* equivalent to *Roy's largest root*?
10. Derive some random data on two response variables in Python and conduct a MANOVA where you hypothesize a three-group structure on that linear combination. Use a significance level of 0.05.

## 5 Simple Linear and Multiple Regression Models (and Extensions)

Linear regression has historically been the workhorse of statistics and is still so today (Stigler, 1986). What is more, many of the common techniques that are in use today are, in one way or another, *extensions* or *derivatives* of the basic linear regression model. Entire books and treatises are written on linear regression as well as entire courses that study the topic in outstanding depth. This is for good reason; *regression is literally at the center of the universe of statistical modeling and that of machine learning algorithms*. Excellent textbook-length sources on regression analysis alone include Fox (2016) and Draper and Smith (1998).

The history of regression ideas and concepts dates back to antiquity (Stigler, 1986). More recently, the origin of least-squares generally is said to lay with Legendre (1805) and later in the same century with Galton (1888), who applied regression to problems of heredity. The essential idea of regression is that of making predictions on one variable based on another (or more) while minimizing some function of the error in making those predictions. Least-squares regression is all about exploiting the correlation between two or more variables with the goal of reducing error in prediction in the one based on knowledge of the other(s). Regression, the *concept*, is the essence of what statistical modeling is all about.

### 5.1 Simple Linear Regression – Fixed Predictor Case

The simplest linear regression model features a single predictor whose values are regarded as fixed in advance. This classic simple linear regression model is given by

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad (5.1)$$

where  $y_i$  is the response variable,  $\beta_0$  is the intercept of the regression line,  $\beta_1$  is the slope,  $x_i$  is the explanatory or independent variable, and  $\varepsilon_i$  is the error term, that part of  $y_i$  that is unaccounted for by the systematic part of the model and where  $E(\varepsilon_i) = 0$ . The intercept is sometimes referred to as the *offset* or *bias* term. This terminology is common in neural networks discussed later in the book.

The systematic part of the model is  $\beta_0 + \beta_1 x_i$ , and thus the expectation of  $y_i$ , that is,  $E(y_i) = \beta_0 + \beta_1 x_i$  is equal to  $\beta_0 + \beta_1 x_i$ . The model has expectation  $E(y_i) = \beta_0$  when  $x_i = 0$  since

$$\begin{aligned} E(y_i) &= \beta_0 + \beta_1 x_i \\ &= \beta_0 + \beta_1(0) \\ &= \beta_0 \end{aligned}$$

We note that the model is linear in the parameters  $\beta_0$  and  $\beta_1$ . This is the meaning of *linear* in linear regression. Note that this definition of linearity does not require the model to be linear in the input variable  $x_i$ . Hence, the following model is still a linear model even though the input enters as a squared term:

$$E(y_i) = \beta_0 + \beta_1 x_i^2$$

In our model of regression in (5.1), as mentioned, we assume that our predictor is *fixed*. That is, if we were to refit the model on new data and thus replicate the experiment, we would use the same values on our predictor  $x_i$ . If  $x_i$  were random, we would presumably use different values since such values were presumably, at least on a theoretical level, drawn from a probability distribution. The theory we develop below assumes a fixed-effects model. For a treatment of random effects in regression, see Fox (2016, p. 228).

### 5.1.1 Parameter Estimates

Recall that we usually do not know values for  $\beta_0$  and  $\beta_1$  and hence typically require estimating them from sample data,

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_i \tag{5.2}$$

How shall values of  $\hat{\beta}_0$  and  $\hat{\beta}_1$  be chosen? Certainly not arbitrarily or at random. As is true for most statistical models, parameter estimates are chosen in such a way that they accomplish some goal, which is usually to *minimize* or *maximize* some objective function. In the case of simple linear regression, these estimates are chosen such that they minimize the sum of squared errors around the regression line in simple linear regression and regression *surface* in multiple regression. That is, we wish to minimize the difference between observed and values predicted by the function  $\hat{f}(x_i)$ :

$$\begin{aligned} \text{RSS} &= \sum_{i=1}^N (y_i - \hat{f}(x_i))^2 \\ &= \sum_{i=1}^N (y_i - \hat{\beta}_0 - \sum_{j=1}^p x_{ij} \hat{\beta}_j)^2 \end{aligned} \tag{5.3}$$

The general form of the above loss function is of the type  $(a - b)^2$  which is a *quadratic* term, and hence the loss is a *quadratic loss*, often referred to as an *L2 loss*. Regression itself in no way requires us to minimize the L2 loss specifically. We could have instead, for instance, minimized the L1 loss, which is that of taking the absolute value of differences,  $|a - b|$ . Regression that minimizes L1 is often referred to as *robust regression* for the reason that through taking absolute deviations, it places less weight on outlying or more extreme scores (i.e., it is more “robust” to such extremes compared to L2). This is in contrast to the L2 loss, which magnifies extreme deviations through squaring them. Deviations from medians instead of means can also make regression more robust. The takeaway point for now is that there are many possible loss functions one may choose. For instance, yet another loss function is the *Huber loss* (for details, see Murphy, 2022). The point is that the fact that we are minimizing the L2 loss in classic regression in no way implies that this is *the* one and only loss function that should dominate all work. However, it is undoubtedly the most

common among loss functions and is typically used as a default. Ideally, however, whatever loss function is chosen should depend on the problem at hand.

The least-squares estimates for  $\hat{\beta}_0$  and  $\hat{\beta}_1$  are found by differentiating the residual sums of squares (RSS) with respect to both  $\beta_0$  and  $\beta_j$ , of which the solutions can be shown to be equal to

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \quad (5.4)$$

for  $\hat{\beta}_0$  and

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (5.5)$$

for  $\hat{\beta}_1$ . It is easy to see that computing the slope requires variation in both  $x$  and  $y$ . For instance, without variation in  $x$ ,  $\sum (x_i - \bar{x})^2$  is equal to 0 and hence  $\hat{\beta}_1$  is undefined.

The slope, in the case of zero intercept, can also be written as the inner product of  $x$  and  $y$  relative to the inner product of  $x$ , that is,

$$\hat{\beta}_1 = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle} \quad (5.6)$$

As we will see when we survey multiple regression shortly, the variance of  $\hat{\beta}_1$  is equal to  $(\mathbf{X}^T \mathbf{X})^{-1} \sigma^2$  where  $\sigma^2$  is estimated by

$$\hat{\sigma}^2 = \frac{1}{N - p - 1} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (5.7)$$

where  $N$  is the number of cases and  $p$  the number of predictors in the model. To compute significance tests on  $\hat{\beta}_0$  and  $\hat{\beta}_1$  we of course require standard errors, which in scalar form can be calculated from

$$SE(\hat{\beta}_0)^2 = \hat{\sigma}^2 \left[ \frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right] \quad (5.8)$$

and

$$SE(\hat{\beta}_1)^2 = \frac{\hat{\sigma}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (5.9)$$

With standard errors computed, we can easily compute confidence intervals for both  $\beta_0$  and  $\beta_1$  as  $\hat{\beta}_0 \pm 2 \cdot SE(\hat{\beta}_0)$  and  $\hat{\beta}_1 \pm 2 \cdot SE(\hat{\beta}_1)$ , respectively.

A test of the null hypothesis that the slope  $\beta_1$  is equal to zero takes the form,

$$H_0: \beta_1 = 0 \quad (5.10)$$

against the statistical alternative hypothesis,

$$H_a: \beta_1 \neq 0 \quad (5.11)$$

To evaluate the null, we compute a  $t$ -statistic,

$$t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)} \quad (5.12)$$

evaluated on  $n - 2$  degrees of freedom.

Recall that the correlation  $r_{yx}$  between  $x$  and  $y$  is given by

$$Cor(x, y) = r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{\text{cov}_{xy}}{s_x s_y}$$

The square of  $r_{yx}$ , that is,  $r^2_{yx}$ , is the proportion of variance accounted for in  $y$  based on knowledge of  $x$ . For example, for  $r^2_{yx} = 0.70$ , this means that 70% of the variance in  $y$  is accounted for by  $x$ .

Note that many of the equations and formulas just reviewed feature the use of *scalar algebra* and not matrix algebra (though we did use matrices a tiny bit). This is because in the simple linear regression case working with scalars generally suffices. As we turn shortly to multiple regression however, we will require working with matrices, where it stands that all the scalar algebra just surveyed can be just as well presented via matrices. Most books on regression will present the theory in matrices because these books aim for the widest and most *general* presentation of linear regression possible, for which using scalars (e.g., simple linear regression) is but a special case.

### 5.1.2 Simple Linear Regression in R

Demonstrating a quick, simple linear regression in R is very easy. We consider the *cars* data in R, a data set featuring numerous characteristics for a variety of cars, including *mileage* and *price*. Our regression will model *mileage* as a function of *price*. First, we load the data and get a quick look at a few cases:

```
> data(cars)
> attach(cars)
> head(cars)
```



|   | Price       | Mileage | Cylinder | Doors     | Cruise | Sound | Leather | Buick | Cadillac | Chevy | Pontiac | Saab | Saturn |
|---|-------------|---------|----------|-----------|--------|-------|---------|-------|----------|-------|---------|------|--------|
| 1 | 22661.05    | 20105   | 6        | 4         | 1      | 0     | 0       | 1     | 0        | 0     | 0       | 0    | 0      |
| 2 | 21725.01    | 13457   | 6        | 2         | 1      | 1     | 0       | 0     | 0        | 1     | 0       | 0    | 0      |
| 3 | 29142.71    | 31655   | 4        | 2         | 1      | 1     | 1       | 0     | 0        | 0     | 0       | 1    | 0      |
| 4 | 30731.94    | 22479   | 4        | 2         | 1      | 0     | 0       | 0     | 0        | 0     | 0       | 1    | 0      |
| 5 | 33358.77    | 17590   | 4        | 2         | 1      | 1     | 1       | 0     | 0        | 0     | 0       | 1    | 0      |
| 6 | 30315.17    | 23635   | 4        | 2         | 1      | 0     | 0       | 0     | 0        | 0     | 0       | 1    | 0      |
|   | Convertible |         | coupe    | hatchback |        | sedan | wagon   |       |          |       |         |      |        |
| 1 |             | 0       | 0        |           |        | 0     | 1       | 0     |          |       |         |      |        |
| 2 |             | 0       | 1        |           |        | 0     | 0       | 0     |          |       |         |      |        |
| 3 |             | 1       | 0        |           |        | 0     | 0       | 0     |          |       |         |      |        |
| 4 |             | 1       | 0        |           |        | 0     | 0       | 0     |          |       |         |      |        |
| 5 |             | 1       | 0        |           |        | 0     | 0       | 0     |          |       |         |      |        |
| 6 |             | 1       | 0        |           |        | 0     | 0       | 0     |          |       |         |      |        |

We now specify the model via `lm()`:

```
> model <- lm(Mileage ~ Price)
> summary(model)
```

Call:

```
lm(formula = Mileage ~ Price)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-20814.9 -4769.1   856.8   5232.4  30662.0
```

Coefficients:

```
      Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.236e+04   6.815e+02  32.814   < 2e-16 ***
Price       -1.186e-01   2.898e-02  -4.093  4.68e-05 ***
---

```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 8117 on 802 degrees of freedom
```

```
Multiple R-squared:  0.02046, Adjusted R-squared:  0.01924
```

```
F-statistic: 16.75 on 1 and 802 DF, p-value: 4.685e-05
```

We see from the output that price is statistically significant ( $p < 0.001$ ). The multiple R-squared associated with the model is 0.02046, which means only approximately 2% of the variance in mileage is accounted for by price. If the effect is so small, why is price statistically significant? This is likely occurring due to the sample size of the data. The cars data has a total of 804 observations and hence rejecting the null hypothesis is relatively easy. Recall that the sensitivity of the  $p$ -value to sample size is something that should always be kept in mind when interpreting statistical tests. It may be tempting based on the statistical significance of price to say that price is an “important” predictor of mileage, but the multiple R-squared and Adjusted R-squared tell a different story with such small amounts of variance explained. The observed  $p$ -value is likely so small simply because of the large sample size.

### 5.1.3 Simple Linear Regression in Python

There are a variety of ways of conducting a linear regression in Python. For instance, one can use `sklearn.linear_model()` or `statsmodels()`. The following is a brief demonstration using `sklearn`:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import mglearn
X, y = mglearn.datasets.make_wave(n_samples=60)
```

We have requested a total of 60 samples, and so we should expect to see 60 values on `X` and `y` (though we only show the first few):

```
X
Out [33]:
array([[ -0.75275929],
       [ 2.70428584],
       [ 1.39196365],
       [ 0.59195091],
       [-2.06388816],
       [-2.06403288],
       ...])

y
Out [34]:
array([-1.18073311,  0.50015912,  0.13772889,  1.17396178, -1.32035898,
        -2.37364735, -0.70117304,  1.20319981,  0.29262945,  0.44971554,
        -0.48647165,  1.39515571,  1.0738403 , -1.30838148, -1.27708305,
        -0.94575843, -0.32691082,  0.26134393, -1.25635546, -0.74646948,
        ...])
```

We now fit the model on `X_train` and `y_train`. We will use the test data shortly to evaluate the model on new data:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
lr = LinearRegression().fit(X_train, y_train)
```

We can request the slope for the model via `lr.coef_`, which we see is equal to 0.39:

```
lr
Out [338]: LinearRegression()
print("lr.coef_:", lr.coef_)
lr.coef_: [0.39390555]
```

To obtain the intercept as well as the training and test set scores, we code:

```
print("lr.intercept_:", lr.intercept_)
lr.intercept_: -0.031804343026759746

print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
Training set score: 0.67

print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
Test set score: 0.66
```

We see from the above that the intercept of the regression equation is equal to  $-0.03$ . We see also that the training set score of 0.67, which is R-squared, is similar to the test set score of 0.66, suggesting that the model is not overfit. What is the test set score? This is the score resulting from running the model on test data, that is, “new data” that the model had not yet seen. Because both scores are similar, it implies the model is likely not overfit. Had the test set score been much smaller than the training set score, it would have likely been indicative of an overfit model.

We illustrate a simple linear regression using `statsmodels()`:

```
import pandas as pd
```

We create some data on X and Y:

```
data = {'X': [10, 12, 15, 13, 20],
        'Y': [24, 21, 36, 12, 19]}
```

```
data
```

```
Out[347]: {'X': [10, 12, 15, 13, 20], 'Y': [24, 21, 36, 12, 19]}
```

```
df = pd.DataFrame(data)
```

```
df
```

```
Out[349]:
```

|   | X  | Y  |
|---|----|----|
| 0 | 10 | 24 |
| 1 | 12 | 21 |
| 2 | 15 | 36 |
| 3 | 13 | 12 |
| 4 | 20 | 19 |

We now extract variables and add a constant, which will be represented by a column of “1” values:

```
y = df["Y"]
```

```
x = df["X"]
```

```
import statsmodels.api as sm
```

```
x = sm.add_constant(x)
```

```
x
```

```
Out[359]:
```

|   | const | X  |
|---|-------|----|
| 0 | 1.0   | 10 |
| 1 | 1.0   | 12 |
| 2 | 1.0   | 15 |
| 3 | 1.0   | 13 |
| 4 | 1.0   | 20 |

```

model = sm.OLS(y, x).fit()
print_model = model.summary()
print(print_model)

OLS Regression Results
=====
Dep. Variable:          Y R-squared:          0.000
Model:                  OLS Adj. R-squared:    -0.333
Method:                  Least Squares F-statistic:      5.515e-16
Date:                    Fri, 15 Mar 2024 Prob (F-statistic):    1.00
Time:                    11:48:49 Log-Likelihood:    -17.406
No. Observations:        5 AIC:                38.81
Df Residuals:            3 BIC:                38.03
Df Model:                1
Covariance Type:         nonrobust
=====
                        Coef      std err      t      P>|t|    [0.025    0.975]
-----
Const                22.4        19.207     1.166     0.328   -38.725   83.525
X                    8.88E-16      1.333   6.66E-16     1.000    -4.242    4.242
=====

Omnibus:              nan Durbin-Watson:          2.778
Prob(Omnibus):         nan Jarque-Bera (JB):        0.331
Skew:                  0.556 Prob(JB):              0.848
Kurtosis:              2.409 Cond. No.             61.2
=====

```

We can see from the above that the coefficient for X is equal to 8.882e-16 with an associated *p*-value of 1.000. The R-squared for the model is 0.000, and in this unique situation, explains zero percent of the variance.

### 5.2 Multiple Linear Regression

In a simple linear regression there is only a single response variable and a single predictor. In a multiple regression model, there are several predictor variables that we model simultaneously in predicting the single response. In a *multivariate multiple regression*, there are not only several predictor variables but also several responses modeled simultaneously.

The multiple linear regression model is given by

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon_i \tag{5.13}$$

where  $E(\varepsilon_i) = 0$  and thus with expectation equal to

$$E(y_i) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

or when expressed as conditional on  $x$ ,

$$\begin{aligned} y &= E(y \mid x_1, \dots, x_p) \\ &= \beta_0 + \sum_{j=1}^p x_j \beta_j \end{aligned}$$

We can more concisely state the above model more generally as a function  $f(x)$ ,

$$f(x) = \beta_0 + \sum_{j=1}^p x_j \beta_j \quad (5.14)$$

where “ $f(x)$ ” represents the function  $f$  given the input of  $x$ . The summation  $\sum_{j=1}^p x_j \beta_j$  allows for as many predictors that we wish to include into the regression model.

As was true for the simple linear regression model, parameter values  $\beta_0, \beta_1, \beta_2, \dots, \beta_p$  are not known and must be estimated from the sample data; hence the *estimated* multiple regression model is given by

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p \quad (5.15)$$

Parameter values are chosen via ordinary least-squares, which is an optimization problem to minimize the sum of squared errors. The least-squares solution is the following, where  $\mathbf{X}$  is a matrix of predictors and  $\mathbf{y}$  a vector of responses:

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (5.16)$$

Though the matrix representation for multiple regression looks more complex than for simple regression, it is not difficult to appreciate the solution. Recall that taking inverses is conceptually equivalent to division in ordinary scalar algebra, and hence if we were to *naïvely* (though mathematically incorrectly) rewrite (5.16) in the most “familiar” way, we could express it as

$$\begin{aligned} \hat{\beta} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \frac{\mathbf{X}^T \mathbf{y}}{(\mathbf{X}^T \mathbf{X})} \end{aligned}$$

We now easily see that the numerator contains in a general way a product of  $\mathbf{X}$  and  $\mathbf{y}$  whereas the denominator contains only a product of  $\mathbf{X}$ . Hence, in words, the solution for  $\hat{\beta}$  is answering the question of the extent to which  $\mathbf{X}$  and  $\mathbf{y}$  “go together” relative to only  $\mathbf{X}$ . If  $\mathbf{X}$  and  $\mathbf{y}$  are highly *linearly* related (emphasis on *linear* here), then  $\mathbf{X}^T \mathbf{y}$  will be much larger than  $\mathbf{X}^T \mathbf{X}$  and hence the coefficient for  $\hat{\beta}$  will be larger than not. On the opposite extreme, if  $\mathbf{X}$  and  $\mathbf{y}$  are not very much linearly related, then  $\mathbf{X}^T \mathbf{y} < \mathbf{X}^T \mathbf{X}$  and  $\mathbf{X}^T \mathbf{y}$  will be only a *fraction* of the total “cross-variation” present in  $\mathbf{X}^T \mathbf{X}$  in the hypothetical division of quantities. Note that this matrix representation then is nothing more than a generalization of the single variable case in which

$$\hat{\beta} = \frac{Cov_{xy}}{S_x^2}$$

where in this simpler case  $Cov_{xy}$  is analogous, in concept at least, to  $\mathbf{X}^T \mathbf{y}$  and  $S_x^2$  to  $\mathbf{X}^T \mathbf{X}$ .

The term

$$\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T, \quad (5.17)$$

that is, with  $\mathbf{y}$  dropped, is the left *pseudo-inverse* of the nonsquare matrix  $\mathbf{X}$  (Murphy, 2022, p. 367). Hence, in cases where the inverse does not exist, a pseudo-inverse may be an option. This is especially useful for nonsquare matrices since ordinary inverses exist only for square matrices. Pseudo-inverses are best computed using the singular value decomposition, or “SVD” (Murphy, 2022).

### 5.2.1 Minimizing Squared vs. Absolute Deviations

Recall that instead of squared deviations from predicted values, one could also take least absolute deviations of the kind

$$|y_i - \hat{y}_i| \quad (5.18)$$

But what is the relationship between minimizing  $(y_i - \hat{y}_i)^2$  vs. minimizing  $|y_i - \hat{y}_i|$ ? Minimizing  $|y_i - \hat{y}_i|$  instead of  $(y_i - \hat{y}_i)^2$  places less emphasis on extreme scores. That is, when we square  $y_i - \hat{y}_i$ , differences are magnified due to the squaring. If we instead minimize  $|y_i - \hat{y}_i|$ , large deviations are not so emphasized.

Minimizing  $|y_i - \hat{y}_i|$  is featured in *least-absolute-value regression*, or “LAV” regression for short. Historically, however,  $(y_i - \hat{y}_i)^2$  is usually first choice in a regression context as squares are usually more tractable mathematically than are absolute value deviations (Fox, 2016).

### 5.2.2 Hypothesis-Testing in Multiple Regression

The null hypothesis under test in the multiple regression model is the following:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0 \quad (5.19)$$

against the statistical alternative hypothesis that at least one of the coefficients,  $\beta_j$ , is not equal to 0. To evaluate, as we did in analysis of variance (ANOVA), we compute an  $F$ -ratio, but not identical to that in ANOVA,

$$F = \frac{(TSS - RSS) / p}{RSS / (n - p - 1)} \quad (5.20)$$

which is evaluated on  $p$  and  $n - p$  degrees of freedom, where  $p$  is the number of predictors and  $n$  is the number of data points.  $TSS$  and  $RSS$  are the total and residual sums of squares, respectively. If the probability of  $F$  under the null hypothesis of  $\rho^2 = 0$  is small, we reject the null hypothesis of no effect and conclude  $\rho^2 \neq 0$  in the population from which the data were drawn.

### 5.2.3 Multiple Linear Regression in Python

We can conduct a multiple linear regression in Python using *sklearn*. As an example, we fit a regression on 100 data points with four features:

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
import pandas as pd

X,y = make_regression(n_samples=100, n_features=4,
n_informative=3, noise=20, random_state=99)

df = pd.DataFrame(X,columns = ['x1','x2','x3','x4'])
df['y'] = y

df
Out [73]:
```

|     | x1        | x2        | x3        | x4        | y          |
|-----|-----------|-----------|-----------|-----------|------------|
| 0   | 0.755180  | -0.154622 | -0.069031 | 0.825647  | 3.974386   |
| 1   | -0.062135 | -0.256100 | 0.865294  | -0.367090 | 21.401357  |
| 2   | -1.966333 | -0.389262 | 1.400000  | 2.171340  | -20.381524 |
| 3   | 1.396620  | -1.133224 | -0.279086 | 0.032219  | -15.004750 |
| 4   | 0.449099  | -1.247421 | 0.065488  | -0.683553 | -56.170348 |
| ... | ...       | ...       | ...       | ...       | ...        |
| 95  | 1.763478  | 1.700284  | -0.528785 | -1.121608 | 47.390283  |
| 96  | 0.199602  | 0.558044  | -1.739733 | 1.360776  | -34.496664 |
| 97  | -0.331549 | 1.775028  | 1.026608  | 0.722101  | 77.484661  |
| 98  | -0.257728 | -0.838072 | 0.488592  | -1.317476 | 21.973845  |
| 99  | 0.751760  | 1.313417  | -0.597966 | -0.415810 | 15.014818  |

```
mlr = LinearRegression()
mlr.fit(df[['x1','x2','x3','x4']], y)

mlr.coef_
Out [40]: array([ 8.33369861, 29.1717497 , 26.6294007 , -1.82445836])
```

Above are printed the coefficients for the model, 8.33, 29.17, 26.63, -1.82. Next we obtain the multiple R-squared for the model, equal to approximately 0.84:

```
mlr.score(df[['x1', 'x2', 'x3', 'x4']], y)
Out [41]: 0.8465893941639528
```

With *statsmodels*, we can fit the same regression as above:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

lm = ols('y ~ x1 + x2 + x3 + x4',df).fit()
lm.params
```

The following are the parameter estimates:

Out [42] :

```
Intercept  -0.740399
x1          8.333699
x2         29.171750
x3         26.629401
x4         -1.824458
```

dtype: float64

```
anova_table = anova_lm(lm)
anova_table
```

Out [44] :

|          | df   | sum_sq        | mean_sq       | F          | PR(>F)       |
|----------|------|---------------|---------------|------------|--------------|
| x1       | 1.0  | 10394.554366  | 10394.554366  | 28.605241  | 6.110239e-07 |
| x2       | 1.0  | 113541.846572 | 113541.846572 | 312.460911 | 8.531356e-32 |
| x3       | 1.0  | 66267.787822  | 66267.787822  | 182.365304 | 7.899193e-24 |
| x4       | 1.0  | 298.584632    | 298.584632    | 0.821688   | 3.669804e-01 |
| Residual | 95.0 | 34521.039456  | 363.379363    | NaN        | NaN          |

We can see that all predictors are statistically significant ( $p < 0.001$ ) except for  $x_4$  ( $p = 0.3669$ ).

#### 5.2.4 Multiple Linear Regression in R

We will perform a multiple regression on the *mtcars* data, which contains characteristics of different makes of automobiles. First, a brief look at the data:

```
> data(mtcars)
> attach(mtcars)
> head(mtcars)
```

|                   | mpg  | cyl | disp | hp  | drat | wt    | qsec  | vs | am | gear | carb |
|-------------------|------|-----|------|-----|------|-------|-------|----|----|------|------|
| Mazda RX4         | 21.0 | 6   | 160  | 110 | 3.90 | 2.620 | 16.46 | 0  | 1  | 4    | 4    |
| Mazda RX4 Wag     | 21.0 | 6   | 160  | 110 | 3.90 | 2.875 | 17.02 | 0  | 1  | 4    | 4    |
| Datsun 710        | 22.8 | 4   | 108  | 93  | 3.85 | 2.320 | 18.61 | 1  | 1  | 4    | 1    |
| Hornet 4 Drive    | 21.4 | 6   | 258  | 110 | 3.08 | 3.215 | 19.44 | 1  | 0  | 3    | 1    |
| Hornet Sportabout | 18.7 | 8   | 360  | 175 | 3.15 | 3.440 | 17.02 | 0  | 0  | 3    | 2    |
| Valiant           | 18.1 | 6   | 225  | 105 | 2.76 | 3.460 | 20.22 | 1  | 0  | 3    | 1    |

We will model *wt* as a function of *disp* and *hp*:

```
> fit <- lm(wt ~ disp + hp)
> summary(fit)
```

Call:

```
lm(formula = wt ~ disp + hp)
```

Residuals:

| Min      | 1Q       | Median  | 3Q      | Max     |
|----------|----------|---------|---------|---------|
| -0.78294 | -0.32249 | 0.02302 | 0.28926 | 0.67179 |



Coefficients:

|             | Estimate  | Std. Error | t value | Pr(> t ) |     |
|-------------|-----------|------------|---------|----------|-----|
| (Intercept) | 1.675818  | 0.195742   | 8.561   | 1.98e-09 | *** |
| disp        | 0.007737  | 0.001089   | 7.108   | 8.04e-08 | *** |
| hp          | -0.001662 | 0.001968   | -0.845  | 0.405    |     |

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4596 on 29 degrees of freedom

Multiple R-squared: 0.7936, Adjusted R-squared: 0.7793

F-statistic: 55.75 on 2 and 29 DF,  $p$ -value: 1.158e-10

We see from the output that `disp` is statistically significant ( $p = 8.04\text{e-}08$ ) but that `hp` is not ( $p = 0.405$ ). The model as a whole explains approximately 79% of the variance in `wt` (i.e., multiple R-squared is equal to 0.7936). Adjusted R-squared is, expectantly, a bit less at 0.7793, as it penalizes the model slightly (we discuss adjusted R-squared shortly).

### 5.3 Geometry of Least-Squares

A picture is worth a thousand words, and this holds true for visualizing how least-squares works. Figure 5.1 is a depiction of the projection of a least-squares line based on two predictors,  $x_1$  and  $x_2$ .

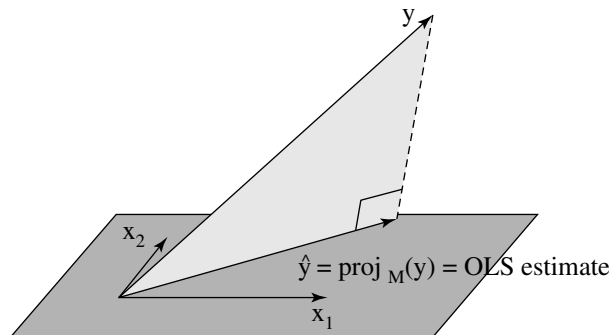


Figure 5.1 Geometry of least-squares regression, where the response variable  $y$  is projected onto the space spanned by predictors  $x_1$  and  $x_2$ . Used with permission of Springer Nature BV, from *Modern Multivariate Statistical Techniques* by Alan J. Izenman, © 2008; permission conveyed through Copyright Clearance Center, Inc.

We can see the response  $y$  and its estimated values  $\hat{y}$  as a function of  $x_1$  and  $x_2$ , the space *spanned* by these predictors. The projection is the ordinary least-squares estimate. The extent to which  $x_1$  and  $x_2$  overlap is the extent to which there may be *collinearity* among the predictors and hence they are taking up a similar space. This overview just scratches the surface regarding visualizing regression and is only meant to whet the reader's appetite should one wish to follow up on the interesting topic of the *geometry of linear models*. For an excellent and much more detailed treatment, see Fox, *The Vector Geometry of Linear Models* (2016, Chapter 10).

### 5.4 Gauss-Markov Theorem: What We Like about Least-Squares Estimates

Why do we like least-squares estimators so much? What is so attractive about them? What is most important about them can be summarized in the *Gauss-Markov theorem*, which asserts that the least-squares estimates of the parameters  $a^T \hat{\beta}$  have the smallest variance among all linear unbiased estimators when the errors are independently distributed with zero expectation and constant variance (Fox, 2016, p. 212). Least-squares estimators are often called “BLUE” which stands for *best linear unbiased estimator*. As noted earlier, the least-squares estimate of  $a^T \hat{\beta}$  is

$$a^T \hat{\beta} = a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (5.21)$$

for which the expectation is equal to

$$\begin{aligned} E(a^T \hat{\beta}) &= E(a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}) \\ &= a^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X} \beta \\ &= a^T \beta \end{aligned} \quad (5.22)$$

The Gauss-Markov theorem holds that for any other linear estimator for which the expectation is  $a^T \beta$  (unbiased), for example,  $E(\mathbf{w}^T \hat{\mathbf{y}}) = \mathbf{w}^T \mathbf{y}$ , the variance of the least-squares estimates  $a^T \hat{\beta}$  is less than or equal to the variance of  $\mathbf{w}^T \mathbf{y}$ . For a proof of this theorem, see Fox (2016, p. 213). An implication of the theorem is that the least-squares estimator has *the smallest mean-squared error out of all the linear estimators we could consider that have no bias* (Hastie et al., 2009).

#### 5.4.1 Are Unbiased Estimators Always Best?

As we will see later, the restriction to unbiased estimators in regression and other methods is not necessarily always a wise one. Recall that the bias-variance trade-off is always at work in estimation. Most of the time, we would like to minimize bias in estimation. At times, however, it may be beneficial to permit some bias into an estimator for the purpose of reducing variance.

Biased estimators such as the *ridge estimator* in *ridge regression* may at times be preferred over an unbiased estimator. One such case when a biased estimator may be preferred is when least-squares estimates are *unstable* (Izenman, 2008). That is, if  $\mathbf{X}$  in  $(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$  is less than full rank, then *singularity* will result, which means the least-squares estimates will no longer be *unique*. When the assumptions of regression do not hold due to collinearity or other problems (e.g., correlated errors), alternative solutions for estimation must be sought. And while exact collinearity may rarely occur, variables that are “almost collinear” can happen quite often depending on the research context. A *biased* estimator can be a better choice when faced with these kinds of conditions as well as for data that are ill-conditioned, meaning that estimated quantities are rather sensitive to small changes in the data (Izenman, 2008). Collinearity and instability often go hand in hand, as the effects of collinearity are such that it can produce overly large positive or negative estimated coefficients whose very signs can be reversed under rather negligible changes to the data. Standard errors could be grossly inflated as well, which could yield non-statistically significant effects, whereas if collinearity were not present, those same effects may have been statistically significant.

In addition to the above, for some data sets, the number of input variables in  $\mathbf{X}$  might greatly exceed the number of observations in the data. Under these conditions, the OLS estimator still exists, but is not *unique*, and hence a biased estimator may be useful in this case. We discuss biased estimators such as ridge and the lasso shortly in the book.

### 5.5 Time Series: An Example of Correlated Errors

The standard linear model assumes independently and normally distributed errors, but the assumption of independence is rarely (if ever) quite right on empirical data. Just like most things in statistics, it is often a reasonable *approximation*, however. Recall that approximations are what statistical modeling is all about. Rarely, if ever, are any assumptions for any model *exactly* met.

When measurements are taken across a *time period*, the assumption of independent errors is usually not satisfied. That is, when the observations comprise a *time series*,  $\varepsilon_i$  are usually *correlated*. Indeed, dependencies among the errors can be very strong in such time-dependent data.

How is this dependency represented in the linear model? Recall in the classic linear model,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where  $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \Sigma_{\varepsilon\varepsilon})$  and where  $\Sigma_{\varepsilon\varepsilon}$  is the covariance matrix of errors, symmetric and positive semi-definite. However, unlike in the typical non-time series least-squares regression model, the expectation for  $\Sigma_{\varepsilon\varepsilon}$  is not one having zeroes on the off-diagonal. Instead, because measurements are usually related, we expect *nonzero diagonal elements* in the off-diagonal representing the correlated errors, which implies  $\Sigma_{\varepsilon\varepsilon}$  will have  $n(n+1)/2$  distinct elements. The dependency of errors in a time series model requires us to incorporate the covariances between time points similar to how we did so in the classic repeated measures models surveyed earlier in the book.

Time series is an enormous topic and we have only scratched the surface here in our cursory overview to give an example of where errors are often correlated in data. Books on time series include those by Chatfield and Xing (2019) and Cryer and Chan (2008). The package *timeseries* (Wuertz et al., 2024) can be used for time series in R, especially applicable to financial time series. The package *pyts* can be used to fit time series models in Python. Other packages such as *sktime* and *tslearn* can also be used. Fox (2016) provides a good, if not somewhat technical introduction to time series and comes highly recommended.

### 5.6 Model Selection in Regression: Is There an “Optimal” Model?

One of the biggest problems in regression analysis and related models is that of determining which model to maintain or keep. *Model selection* in statistics in general is a huge problem and is usually not quite as simple as using statistical criteria alone to determine which model to adopt. Research and scientific criteria should always enter the discussion. Too often researchers adopt a “kitchen-sink” approach (i.e., throw in everything but the kitchen sink) to variable selection which often leads to overly complex models, problems of collinearity, and poor generalizability of test data.

Among the statistical approaches, there are generally two common strategies used for selecting a model. Recall that in choosing a model, we are not interested in the model that minimizes *training error*; we are interested in the model that minimizes *test error*. The most direct way to evaluate test error would be to employ a *cross-validation* approach. For example, we could fit our model to 80% of the data and test on 20%. This gives us a direct estimate of the test error because it tells us exactly how the model performs on new data, that is, data the model has yet to “see.” This is a

first strategy to choosing a model, that of evaluating the fitted model on test data to learn how it performs.

For many reasons, however, we may not want to follow the above direct cross-validation approach. Some of these reasons include the fact that historically, data may have been expensive and sometimes very inconvenient to collect in the given field, and hence training a model on anything less than the full data set is sometimes simply not feasible. Indeed, before computers came along, the idea of setting aside data to test on via fast computation was, of course, not even an option. Hence, what if we decided to not put aside test data and wanted simply to estimate the test error on the training data itself, even if imperfectly? Since we only have the training data, fortunately we can *adjust* the training error rate to, for some measures at least, better reflect test error. We consider these options next.

### 5.7 Effect Size, Model Selection, and Adjusting the Training Error Rate

A rejection of the null hypothesis in regression, crudely, suggests that the regression is predicting better than chance. However, on its own, a statistically significant  $p$ -value does not necessarily tell us much more than that, unless of course we have specific knowledge of the inputs to the computed  $p$ -value such as sample size, variance, and so on, to help make sense of the  $p$ -value finding for the given data set. As is usually the case with significance testing, it is entirely possible to reject a null hypothesis for regression yet it be a *trivial* scientific effect. Such can occur often when sample size is extremely large and hence even miniscule effects cause us to reject the null hypothesis. This is nothing new, as we have discussed in many other places in the book the influence that sample size has on computed  $p$ -values. As usual, what is needed is a measure of *effect size*. A good effect size is not sample size dependent. The problem with effect size, however, is that if we compute it only on the training data then we usually get an overly *optimistic* size of effect. This is not surprising and is in line with our understanding of how models fit to training data are generally too optimistic overall.

In what follows we first consider an effect size for training data, then survey a few strategies for measuring size of effect as well as *model selection measures* that better evaluate the parsimony of statistical models, and in some cases (e.g., *Akaike's Information Criterion*), also help us better estimate test error.

#### 5.7.1 $R^2$ , Adjusted $R^2$ , $C_p$ , AIC, BIC

Recall that  $R^2$  in regression is given by

$$R^2 = \frac{\sum_{i=1}^n (y_i' - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (5.23)$$

The denominator,  $\sum_{i=1}^n (y_i - \bar{y})^2$  is simply a sums of squares of actual data values from  $\bar{y}$ . This quantity will be large or small depending simply on how much variation we have in the current data set. However, look more closely at the numerator,  $\sum_{i=1}^n (y_i' - \bar{y})^2$ ; this term expresses the distance between predicted values,  $y_i'$  and the mean of the data,  $\bar{y}$ . If this distance is zero, then

$\sum_{i=1}^n (y_i' - \bar{y})^2$  is equal to 0 and  $R^2$  will equal 0. As this distance increases,  $R^2$  becomes larger, and the quadratic (i.e., squared) nature of sums of squares amplifies the magnitude, especially for deviations that are distant from the mean. Hence, it would appear at first glance that this  $R^2$  statistic is all we really need to choose a model, such that *the larger the value of  $R^2$ , the better the model*. But this logic is problematic, since in almost all cases  $R^2$  will necessarily mathematically increase as the number of predictors entered into the model also increases. Hence, models with a greater number of predictors will result in larger  $R^2$  values. The value of  $1 - R^2$  will give us a measure of what is typically called the *apparent error rate*. What we really want to know is the *actual error rate*.

One way around this difficulty is to interpret, instead of  $R^2$ , an *adjusted  $R^2$  statistic* given by the following:

$$R_{adj}^2 = 1 - (1 - R^2) \left( \frac{n-1}{n-p} \right) \quad (5.24)$$

where “ $p$ ” here is the number of parameters in the model, including the intercept term. From (5.24) we can see that as  $p$  increases relative to a stable  $R^2$ ,  $R_{adj}^2$  *decreases* instead of automatically increasing as was the case for  $R^2$ . Hence,  $R_{adj}^2$  essentially “penalizes” the user for including predictors that are not “worth their salt” in increasing  $R^2$ .

The  $C_p$  statistic (“Mallow’s  $C_p$ ”) is another approach to penalizing model fit,

$$C_p = \frac{1}{n} (RSS + 2d\hat{\sigma}^2) \quad (5.25)$$

where, as before,  $n$  is the number of cases or observations,  $RSS$  is the residual sums of squares on the training data,  $d$  now is the number of predictors (recall that  $p$  was the number of *parameters* in  $R_{adj}^2$ ), and  $\hat{\sigma}^2$  is an estimate of error variance using all predictors, usually MS residual. The nature of the penalty in  $C_p$  is by way of  $2d\hat{\sigma}^2$ . Smaller values of  $C_p$  are preferred over larger values since smaller  $RSS$  implies a better-fitting model. However, this is offset by  $2d\hat{\sigma}^2$ , where increasing  $d$  has the effect of increasing  $2d\hat{\sigma}^2$ . Hence,  $C_p$ , if it could speak, would say, “Congrats on getting  $RSS$  down, but that decrease will be penalized by how many  $d$  it required to do so. If any of your  $d$  are not sufficiently contributing to the observed decrease in  $RSS$ , you may want to consider removing them instead of being penalized for including them.”

A similar statistic to the  $C_p$ , and one that is more popular, is that of the AIC (*Akaike’s Information Criterion*) statistic,

$$AIC = \frac{1}{n\hat{\sigma}^2} (RSS + 2d\hat{\sigma}^2) \quad (5.26)$$

which is very similar in logic to that of the  $C_p$ : the only difference is the multiplier  $\frac{1}{n\hat{\sigma}^2}$  instead of  $\frac{1}{n}$  in  $C_p$ . The penalty is still equal to  $2d\hat{\sigma}^2$ , however. In this formulation, increasingly smaller values are preferred over larger values. The following is another version of AIC, the original version given by Akaike (1974),

$$-2\ln p(D|\theta) + 2m \quad (5.27)$$

which is the familiar log-likelihood,  $\ln p(D|\theta)$ , however pre-multiplied by  $-2$ . For this version of AIC, smaller is also better than larger, as the penalty  $m$  (number of *parameters* in the model) acts to increase AIC, and if the decrease in  $-2\ln p(D|\theta)$  isn't "worth" the increase by  $m$ , it makes adding the extra parameter(s) less worthwhile in terms of maintaining model parsimony. However, "smaller is always better" does not always pertain to the report of an AIC statistic; *it depends on how it is formulated*. AIC could, in theory, appear as simply  $\ln p(D|\theta) - 2m$  where now a *larger* than not value is preferred. Notice that now the log-likelihood is positive without pre-multiplying it by  $-2$  and as before we are subtracting the number of parameters in the way of  $2m$ . *Be sure you are clear on how AIC (or BIC) is specifically defined for your particular application (e.g., software, package, etc.) before you interpret it.*

A derivative of both the  $C_p$  and AIC is the Bayesian Information Criterion (BIC),

$$BIC = \frac{1}{n\hat{\sigma}^2} (RSS + \log(n)d\hat{\sigma}^2) \quad (5.28)$$

The BIC often penalizes to a greater extent than the AIC. The key difference in the two statistics is in what follows  $RSS$  in their respective formulations. In the AIC, that term is  $2d\hat{\sigma}^2$ , where in the BIC, it is  $\log(n)d\hat{\sigma}^2$ . All else equal, as sample size  $n$  increases, so does  $\log(n)$  and BIC will typically "punish" more than AIC for the same  $RSS$  and  $\hat{\sigma}^2$ . As with AIC, be sure you know how the author is defining BIC before concluding that "smaller is better" (e.g., see Schwarz, 1978; Fraley & Raftery, 1998, where BIC is translated into a "larger is better" statistic).

As the name suggests, BIC originates from Bayesian decision theory and hence is derived from Bayesian principles. It can be shown that choosing a model with the minimum BIC is equivalent to choosing the model with the highest approximate posterior probability (Hastie et al., 2009). What is more, BIC is asymptotically consistent as a selection criterion, which implies that the probability that BIC will select the correct model increases toward unity (i.e., 1) as the amount of information (sample size) on which it is based increases. This is not true for AIC, which will too often select a model that is too rich or complex as information on which it is based increases (i.e.,  $n \rightarrow \infty$ ). For smaller sample sizes, BIC typically penalizes much more severely than AIC and hence may choose a model that is too simple (Hastie et al., 2009).

An additional measure of effect size that is sometimes recommended in regression is that of the *semi-partial correlation* coefficient (Hays, 1994). The semi-partial correlation partials out variance due to other predictors in the model before relating a predictor to the response variable. That is, it is the bivariate correlation between an adjusted variable (predictor) and an unadjusted variable (response). For details, see Hays (1994).

### 5.7.2 Comparing $R_{adj}^2$ , $C_p$ , AIC, and BIC to Cross-Validation

We have surveyed above a few ways of penalizing training data to assess model fit and model selection. Which measure is best? Should one use any of the  $R_{adj}^2$ ,  $C_p$ , AIC, or BIC options, or should one use cross-validation? Cross-validation is usually first choice, since it is a direct estimate of the test error, whereas only  $C_p$  and AIC rather efficiently, though *indirectly*, estimate test error. Cross-validation also makes fewer assumptions about the underlying model (James et al., 2013).

Historically, before high-speed computing power and very large data sets (at least in some fields), performing cross-validation directly was computationally quite difficult or borderline impossible and hence computing measures such as  $C_p$ ,  $R_{adj}^2$ , etc. was more feasible and convenient. Today, however, with high-speed computing, cross-validation can be performed with relative ease. Hence, given the availability of relatively cheap computing speed, cross-validation is often the preferred option for model validation. Of course, nothing stops the investigator from simultaneously computing many competing effect-size measures and comparing them to see if they *triangulate* on the same data set. There is no “type I” error rate when computing effect sizes; the researcher can compute as many of them as she would like.

## 5.8 Assumptions for Regression

Virtually all statistical models are founded on a set of assumptions. Verifying assumptions for regression is important because least-squares estimates and inference are only as good as the assumptions on which they are based. Further, techniques have been developed that allow for users to evaluate the impact that data points have on the estimated least-squares solution. Thus, in addition to verifying assumptions such as *linearity* and *homoscedasticity of variance*, residual analyses that evaluate such things as leverage, raw, standardized, and studentized residuals are generally recommended.

Residual analysis in regression is a huge topic and due to space limitations, we do not survey it here. Defining such things as *leverage* and *influence* of data points requires more theoretical development than we can afford here. We also assume the reader may already have at least some exposure to this area in prior courses. We refer the reader to Fox (2016) for a thorough and excellent treatment of residual analyses, which can become quite complex and is an art unto itself. We cannot escape the topic of assumption analysis entirely, however, and must survey one of the most important areas of assumption-checking in regression and related models, that of collinearity.

### 5.8.1 Collinearity

Collinearity can have serious and deleterious effects on a regression solution. If the columns of  $\mathbf{X}$  are not linearly independent, this equates to  $\mathbf{X}$  not being of full rank, which implies that  $\mathbf{X}^T \mathbf{X}$  is singular, the determinant is equal to 0, and the least-squares coefficients, though they theoretically can be estimated, are not *uniquely* defined. Hence, collinearity in regression is a serious problem and we need a way to address it.

#### 5.8.1.1 Variance Inflation Factor

There are a few common ways of assessing collinearity. The first we survey is probably the most popular, that of computing a statistic called the *variance inflation factor*, or “VIF” for short. To understand VIF, we must first consider how we can write the variance of the  $i$ th partial regression coefficient (Cohen et al., 2003, p. 423):

$$s_{\beta_i}^2 = \frac{s_y^2}{s_{x_i}^2} \left( \frac{1 - R_{y,12\dots p}^2}{n - p - 1} \right) \left( \frac{1}{1 - R_{i,12\dots(i)\dots p}^2} \right) \quad (5.29)$$

where  $s_{\beta_i}^2$  is the variance of the  $\beta_i$  coefficient,  $s_y^2$  is the variance of the response,  $s_{x_i}^2$  is the variance of the given predictor  $x_i$  corresponding to  $\beta_i$ , and  $R_{y.12\dots p}^2$  is the coefficient of determination for the entire model with all predictors  $y.12\dots p$ . The variance inflation factor is the last term in (5.29):

$$\text{VIF} = \left( \frac{1}{1 - R_{i.12\dots(i)\dots p}^2} \right) \quad (5.30)$$

where  $R_{i.12\dots(i)\dots p}^2$  represents the explained variance of the predictor in question regressed on the rest of predictors in the model. The extent to which  $R_{i.12\dots(i)\dots p}^2$  is large is the extent to which the denominator  $1 - R_{i.12\dots(i)\dots p}^2$  goes toward zero, and hence is the extent to which VIF will be larger than not. That is, the extent to which the predictor in question shares variance with other predictors in the model is the extent to which the variance inflation factor increases. The variance inflation factor, as the name suggests, measures *the factor by which the variance of the given regression coefficient increases due to its collinearity with other variables in the model*.

A second popular way of assessing ill-conditioning is through the *condition number*. The *condition number*  $k$  is the square root of the ratio of the largest to the smallest nonzero eigenvalues, that is,  $k = \sqrt{\lambda_1 / \lambda_2}$ , where  $\lambda_1$  is the largest eigenvalue and  $\lambda_2$  is the smallest (Fox, 2016). If this ratio is large,  $\mathbf{X}$  is considered *ill-conditioned*. It makes sense that a comparison of eigenvalues here can be used as an indicator of ill-conditioning. Consider what it means to have a very wide range between the largest and smallest eigenvalues. It implies that an increasing amount of variance is in one “component” (i.e., *principal component*) only in the data, which implies that there exists a large degree of correlation among variables “coagulating” into that component. Hence, the condition number can serve as an aid in detecting collinearity.

### 5.8.2 Collinearity Necessarily Implies Redundancy Only in Terms of Variance

In our discussion of collinearity thus far, it should be emphasized that two variables that are nearly perfectly collinear are redundant only in terms of *shared variance*, and not necessarily in terms of what they are actually individually measuring. A common misunderstanding when considering collinearity is to assume that two collinear variables are “getting at the same thing” on a substantive level simply because they are highly correlated. And while in some cases this may be true, it is not necessarily so across all cases.

For example, consider the classic case of correlating height and weight. Suppose the correlation is near 0.90. The shared variance between variables is relatively high, but we would all agree that weight is not the same thing as height. More generally, *a statistical measure, whatever the kind, in no way necessarily reflects substantive reality*. Whenever we apply an arbitrary statistical function to any “real object,” it is never a one-to-one relationship. *The statistics are never enough; you have to know what the statistics are being applied to in order to draw the proper scientific conclusion*. A lack of understanding of what statistics *can* vs. *cannot* tell you about empirical data is probably the greatest pitfall for users of statistics who have not considered these wider and deeper methodological and philosophical issues.

## 5.9 Variable Selection Methods: Building the Regression Model

Previously we discussed model fit and model selection measures, such as  $R^2$ ,  $R_{adj}^2$ , AIC, etc. While we discussed how to potentially choose the best model, we did not discuss *how* to select



variables for a given model. This latter concern falls under the category of *variable selection* rather than the previously discussed topic of *model selection*. One way to conceive of variable selection in a very general sense is as suggested by Breiman (2001), such that a variable is important *if dropping it from the model seriously affects prediction*. And while variable selection may at first glance appear to be simple, it is an extremely complex problem, just as complex as model selection overall. Most variable selection procedures are quite unstable as well, such that tweaking or perturbing the original data even slightly can have a rather drastic effect on the final model (Breiman, 1996).

Variable selection methods include *forward*, *backward*, and *stepwise* techniques, where stepwise can be considered a hybrid version of forward and backward. Stepwise and selection algorithms often go by the name of *wrapper models* in the data-mining literature (Aggarwal, 2015). Versions of backward elimination algorithms are sometimes referred to as *recursive feature elimination algorithms* in the data and machine learning environments, and though there are distinctions between these types of algorithms, they pretty much get at the same thing in the end.

Other techniques include *all-possible-subsets*, *forward-stagewise regression*, and others. Both ridge and lasso regression, as we will see in the next chapter, can be considered in some sense variable selection techniques as well, especially the lasso since coefficients are explicitly driven to values of zero, while for ridge coefficients they are simply “shrunk” toward zero. For this reason, ridge is often not considered a “pure” variable selection procedure. However, nothing inhibits the user from simply dropping shrunken coefficients as they near zero or are much less in magnitude than other coefficients in the model. We will return to this discussion when we discuss both techniques in the following chapter.

In typical forward regression, predictors are entered based on some entry criteria, such as statistical significance (e.g., attaining  $p < 0.05$ ). The procedure halts when no more predictors reach that criterion. In backward regression, all predictors begin in the model and are removed once they no longer reach  $p < 0.05$  (or whatever level one chooses). Stepwise regression is typically a mix of both forward and backward approaches; however there are many stepwise programs and packages and it behooves the reader to study the details of the package they are using to clearly understand what the stepwise program does or does not do. The algorithms in various packages can become quite complex, which if not understood, provides ample opportunity for drawing false substantive conclusions from the stepwise results.

### 5.9.1 Forward, Backward, and Stepwise in R

There are several ways (and packages) for conducting forward, backward, or stepwise regressions in R. Many packages and routines will often combine routines, so as already emphasized, it is essential that one thoroughly familiarize oneself with the given package to know exactly what is being computed.

For our example here, we feature the *mtcars* data in R once more, on which we conduct both a *forward* and *backward* stepwise regression. Recall that the *mtcars* data contains numerous vehicle specs such as gas mileage (`mpg`), number of cylinders (`cyl`), and displacement (`displ`), among other characteristics of vehicles. We will use the *StepReg* package (Li et al., 2024) to carry out stepwise regression:

```
> library(StepReg)
> data(mtcars)

> head(mtcars)
```

```
      Mpg  cyl  disp  hp  drat    wt   qsec vs  am  gear carb
Mazda RX4         21.0   6  160  110   3.90 2.620 16.46 0   1     4     4
Mazda RX4 Wag     21.0   6  160  110   3.90 2.875 17.02 0   1     4     4
Datsun 710        22.8   4  108   93   3.85 2.320 18.61 1   1     4     1
Hornet 4 Drive    21.4   6  258  110   3.08 3.215 19.44 1   0     3     1
Hornet Sportabout 18.7   8  360  175   3.15 3.440 17.02 0   0     3     2
Valiant           18.1   6  225  105   2.76 3.460 20.22 1   0     3     1
```

```
> formula <- mpg ~ .
```

In specifying `mpg ~ .`, we are implicitly requesting modeling `mpg` as a function of all other variables. We will now specify that we wish to conduct a stepwise, specifically:

```
> fit <- stepwise(formula = formula,
+ data = mtcars,
+ type = "linear",
+ strategy = c("forward", "bidirection", "backward"),
+ metric = c("AIC", "BIC", "SL"))
> fit
```

We have requested above that forward, bidirection, and backward elimination be performed, and hence our output will be quite long. We have also requested AIC, BIC, and SL as the metrics for evaluating the regressions, where “SL” corresponds to “significance level.” Indeed, if we choose to look at the results from the stepwise (`fit`), we will get a listing of the model selection variables in the data set that is quite long and provides much detail. To at least illustrate some of the output, we obtain a listing of the AIC values under the *forward* algorithm (Table 5.1):

Table 5.1 AIC Value for Steps Using a Forward Algorithm

Summary of selection process under forward with AIC

| Step | EffectEntered | NumberParams | AIC        |
|------|---------------|--------------|------------|
| 1    |               | 1            | 149.94345  |
| 2    | wt            | 2            | 107.217363 |
| 3    | cyl           | 3            | 97.197999  |
| 4    | hp            | 4            | 96.664562  |

We can see that at step 4, AIC is the smallest under a forward algorithm. Recall that, typically, the smaller the AIC, the better. The reason we say “typically” here is because recall that AIC can also at times be defined such that increasing values of AIC suggest a better-fitting model. *Always check how AIC is being defined in your specific software application or package before concluding what denotes an increasingly better value.*

The benefit to using *StepReg* is that we can also visualize the results of the selection process where in Figure 5.2 is shown the effect of *variable removal* (e.g., (-) cyl) on each of AIC, BIC, and SL, this time resulting from the *backward* algorithm:

```
> plot(fit)
```

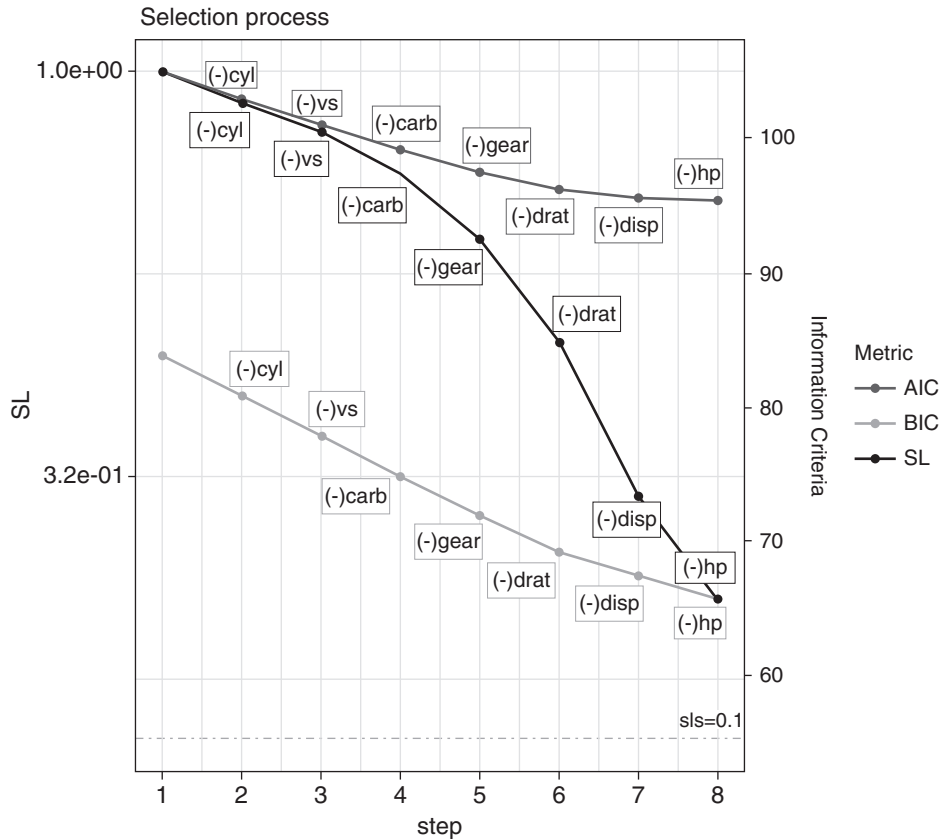


Figure 5.2 AIC, BIC, and SL as a function of step, where the effect of removing variables (e.g., (-)cyl) is displayed resulting from the backward algorithm.

We see from the plot that at each step of the function, AIC, BIC, and SL drop in value as predictors are removed. The plot is a powerful way of visualizing the selection process. We see that SL drops rather dramatically toward the latter steps. BIC follows an almost perfect negative linear path across steps.

In Python, forward and stepwise regression can be performed using `sklearn's SequentialFeatureSelector()` though not demonstrated here. For details, see Python's `sklearn` site.

## 5.10 Mediated and Moderated Regression Models

Mediation has a relatively long history in statistics and has been motivated primarily by the substantive interests of social scientists, primarily psychologists. Despite its popularity and hype in

some fields, it is important to understand what statistical mediation *can* vs. *cannot* do. Moderation, on the other hand, is most generally simply another word for *interaction*. In ANOVA-type models, we usually refer to such as interactions, whereas in regression models it has come to be known as moderation. It is vitally important to realize at the outset that *mediation and moderation do not, by themselves, imply a physical process*. That is, conducting a test of statistical mediation and concluding *Z mediates the relationship between X and Y* is only necessarily true *statistically*; it is not true *substantively* unless one has a substantive rationale or argument outside of the statistical model for inferring such a mediational process. There are countless published articles, even in “high impact factor” journals, where researchers conclude a mediational process simply based on *statistical* mediation, which exemplifies one of many philosophical and methodological abuses of statistics. We revisit and discuss this issue more extensively later.

### 5.10.1 Statistical Mediation

We purposely title this section by the name of *statistical* mediation for the reason that, as mentioned, far too often researchers tend to believe that if they apply a mediation model to data, it can thereby “extract” a mediational *process*. This is simply not the case. In addressing this important issue, let’s first very briefly survey how mediation (and moderation) is defined in the literature. For example, Little et al. (2007, p. 207) offer the following definition and distinction:

Generally speaking, a mediator can be thought of as the carrier or transporter of information along the causal chain of effects. A moderator, on the other hand, is the changer of a relationship in a system.

Pearl (2014, pp. 1–2) claims, “Scientifically, mediation tells us “how nature works” and, practically, it enables us to predict behavior under a rich variety of conditions and interventions.”

The classic mediation model is diagrammed in Figure 5.3, where  $X$  is the predictor,  $Y$  is the response, and  $M$  is the mediator.

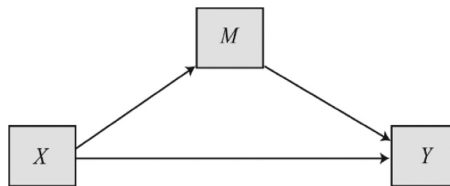


Figure 5.3 Classic mediation model where “ $M$ ” is a mediator of the relationship between  $X$  and  $Y$ .

The conditions for “classic” or traditional mediation typically include the following (MacKinnon et al., 2007):

1.  $X$  should bear a relationship to  $Y$ .
2.  $X$  should bear a relationship to  $M$ .
3.  $M$  should be related to  $Y$  when both  $X$  and  $M$  are predictors of  $Y$ .
4. The relationship of  $X$  to  $Y$  should *diminish* (in absolute value) when  $M$  is included in the model, presumably as a result of  $M$  “taking up” the correlation that would have otherwise been “transmitted” (in necessarily some *statistical* sense only) from  $X$  to  $Y$ .

A more complex combination of mediator models is illustrated in Figure 5.4 where we now have three mediators,  $z_1$ ,  $z_2$ , and  $z_3$ , and where  $\varepsilon_1$ ,  $\varepsilon_2$ , and  $\varepsilon_3$  are errors.

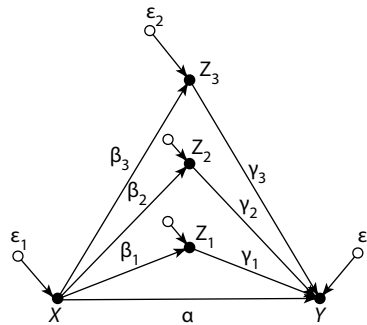


Figure 5.4 A three-mediator model. Used with permission of Springer Nature BV, from *The Causal Mediation Formula—A Guide to the Assessment of Pathways and Mechanisms* by Judea Pearl, © 2012; permission conveyed through Copyright Clearance Center, Inc.

While including more than a single mediator may seem initially a good idea, it comes with many caveats. The models are only as valid as the assumptions imposed on those models. Indeed, as again noted by Pearl (2012, pp. 2–3), whom we paraphrase:

The validity of these two methods depends of course on the *assumption* [added emphasis] that the error terms ... are uncorrelated, otherwise, some of the structural parameters would produce biased results. In randomized trials, where error can be identified with the randomization device, we are assured that error is uncorrelated with other errors and so the regression estimates will be unbiased ... However, randomization does not remove error from  $e_2$  and  $e_3$ .

There is no end to how complex we can formulate a mediation model. For example, Figure 5.5 features a mediation model with a confounder that predicts the treatment, mediator, and outcome. Other than the confounder, the model is a classic three-variable mediation model.

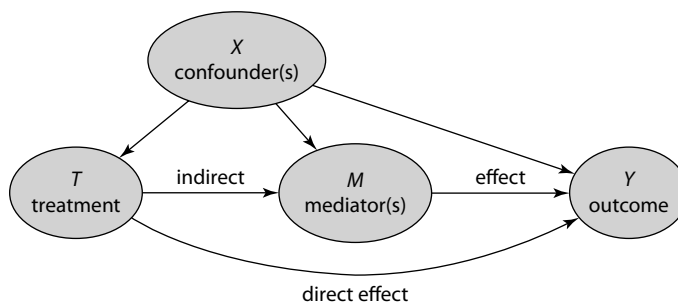


Figure 5.5 Mediation model with confounder(s).

Other than for rather simple mediation models, models such as these are often best analyzed as *structural equation models*, and hence we will revisit them to some degree later in the book when we survey SEM models in more detail. Estimation in mediation models includes the traditional *coefficient product estimator* (Hayes, 2018). Simulation-based estimators have also been proposed as well as semi-parametric estimators such as the *inverse probability weighting estimator* (Zhang,

2023). The *multiply-robust estimator*, as well as the *double-machine learning for mediation estimator*, have also been proposed. For details, see Abécassis et al. (2023).

### 5.10.2 Statistical Moderation

As mentioned at the outset of our discussion of mediation and moderation, moderation analysis is nothing more than analyzing interactions. That is, quite literally, “moderation” is a fancy word for statistical *interaction*. When you conduct an analysis of variance and model the interaction, you are performing a moderation analysis. However, there is so much “hype” about moderation analyses in fields such as psychology and others that it is often advertised as being able to accomplish so much more than it can. Again, to reiterate, nowhere in the *statistical* procedure of moderation analysis is anything necessarily “moderating” anything else in a *physical* or *substantive* sense unless the variables you are working with and your theory suggest such a “physical” process. *Statistical moderation alone does not afford that process in any way whatsoever*. In fact, in many cases, statistical moderation can be rather anticlimactic.

As a basic example of moderation, consider the plot in Figure 5.6.

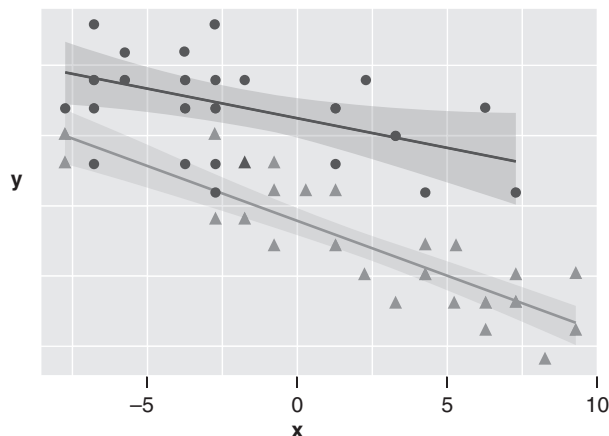


Figure 5.6 Illustration of moderation for two groups; slopes are nonparallel.

As a result of the slopes being different (*nonparallel*), we have a moderation effect. Of course, in the *sample*, there will virtually *always be a moderation effect*. Do you really think slopes could ever be *exactly* parallel in the *sample*? The real question is whether such an effect can be inferred toward the population, which is the inferential question we are ultimately interested in. We will survey an example of moderation later in the chapter using software.

### 5.10.3 Moderated Mediation

Just as in the test for any other interaction effect, one may be interested in learning whether an entire mediation model holds across values of a moderating variable. More specifically, we may be interested in learning whether an *indirect effect* (i.e., the effect of going “through” the mediator, so to speak) is conditional on values of a moderating variable. This is referred to as *moderated mediation* and more generally is found under the name of *conditional process models* (Hayes, 2018), or again, the search for *conditional indirect effects*. That is, the indirect (or “mediating”) effect is conditional upon some other third variable, illustrated in Figure 5.7.

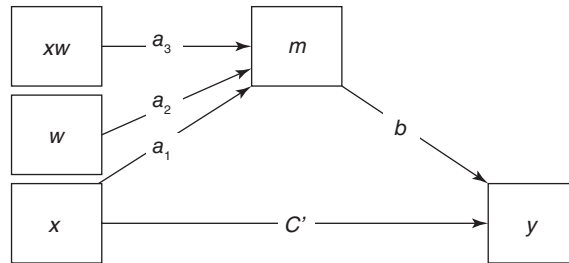


Figure 5.7 Three-variable mediation model conditional upon  $w$ .

We see in Figure 5.7 the classic three-variable mediation model, however conditional upon the  $w$  variable as indicated by the  $xw$  interaction. Note that both  $w$  and the interaction term  $xw$  are included due to the *principle of marginality* (Fox, 2016) which holds that an interaction term should not be included without also including the individual terms that make up the interaction, which in this case is simply the  $w$  variable.

A *mediated moderation* model is when an interaction between an independent variable and a moderator variable “affects” (statistically) a mediator variable that in turn “affects” (again, only necessarily *statistically*) the outcome variable. According to Hayes (2018, p. 465), “mediated moderation is neither interesting nor meaningful” and ends up being mathematically equivalent under most conditions to moderated mediation. As such, Hayes recommends that questions of mediated moderation should be recast as *moderated mediational hypotheses*.

#### 5.10.4 Mediation in Python

Mediation and moderation analyses are probably best left to R or similar software specifically designed for statistics (e.g., SPSS also has macros devoted to performing mediation). However, Python does afford some capabilities for this as well. The following is an example of mediation using the *pingouin* package (Vallat, 2018) in Python. The *pingouin* package is an open-source package originally written in Python 3 and uses primarily *pandas* and *numpy*.

We first install *pingouin* via *pip* and then import the `mediation_analysis` function:

```
pip install pingouin
from pingouin import mediation_analysis, read_dataset
df = read_dataset('mediation')
```

```
df
```

```
Out [216]:
```

|    | X  | M  | Y  | Mbin | Ybin | W1  | W2  |
|----|----|----|----|------|------|-----|-----|
| 0  | 6  | 5  | 6  | 0    | 1    | 0.0 | 1.0 |
| 1  | 7  | 5  | 5  | 1    | 0    | 0.1 | 1.0 |
| 2  | 7  | 7  | 4  | 0    | 0    | 0.2 | 1.0 |
| 3  | 8  | 4  | 8  | 0    | 1    | 0.3 | 1.0 |
| 4  | 4  | 3  | 5  | 0    | 1    | 0.4 | 1.0 |
| .. | .. | .. | .. | ...  | ...  | ... | ... |
| 95 | 5  | 2  | 2  | 1    | 0    | 1.0 | 1.0 |
| 96 | 8  | 4  | 1  | 1    | 0    | 1.0 | 1.0 |
| 97 | 7  | 4  | 5  | 0    | 0    | 1.0 | 1.0 |
| 98 | 2  | 2  | 1  | 0    | 1    | 1.5 | 1.0 |
| 99 | 5  | 4  | 5  | 0    | 0    | 2.0 | 1.0 |

In the following, we will designate  $X$  as the predictor,  $M$  as the mediator, and  $Y$  as the response. We set the significance level ( $\alpha$ ) at 0.05:

```
mediation_analysis(data=df, x='X', m='M', y='Y', alpha = 0.05, seed = 42)
Out [214] :
```

|   | path     | coef     | se       | pval         | CI [2.5%] | CI [97.5%] | sig |
|---|----------|----------|----------|--------------|-----------|------------|-----|
| 0 | M ~ X    | 0.561015 | 0.094480 | 4.391362e-08 | 0.373522  | 0.748509   | Yes |
| 1 | Y ~ M    | 0.654173 | 0.085831 | 1.612674e-11 | 0.483844  | 0.824501   | Yes |
| 2 | Total    | 0.396126 | 0.111160 | 5.671128e-04 | 0.175533  | 0.616719   | Yes |
| 3 | Direct   | 0.039604 | 0.109648 | 7.187429e-01 | -0.178018 | 0.257226   | No  |
| 4 | Indirect | 0.356522 | 0.083313 | 0.000000e+00 | 0.219818  | 0.537654   | Yes |

We can see from the above that while the direct path is not statistically significant, the *total* and *indirect* paths are. The *direct effect* is from the exogenous variable to the endogenous one, which for our data is from  $X$  to  $Y$ . The *indirect effect* is the pathway through the mediator. The *total effect* is the sum of the direct and indirect effects.

The penguin package can also easily fit mediation with binary variables as well as models with multiple mediators. Mediation models can also be fit with `statsmodels.stats.mediation()`, though we do not explore these usages here. The package *moderate.mediation* in R (Qin, 2024) can be used to fit rather complex mediation models and comes highly recommended for moderated mediation modeling. The package *mediation* (Tingley et al., 2014) also fits a variety of mediation models. Sensitivity analyses for mediation effects can also be performed. See Imai et al. (2010) for details. As a summary on mediation and moderation analyses in R, the package *mediation* (Tingley et al., 2014) is the “go-to” source for all types of mediation analyses using R.

### 5.11 Further Directions and a Final Word of Warning on Mediation and Moderation

Mediation and moderation in statistics is an enormous topic with books devoted to these methods (e.g., see Hayes, 2018). Recent work on mediation includes that of Ge et al. (2023) who develop a reinforcement learning framework for *dynamic mediation analysis*. In many cases, treatments are assigned over time and this work seeks to evaluate mediators along a longitudinal trajectory as well as sequential decision-making over an infinite number of decision points. Pearl (2012) critically assesses the habitual way of performing mediation, which assumes linearity and independence of errors, and proposes an alternative approach for modeling nonlinear relationships, which includes that with categorical variables. Judd et al. (2001) give details on estimating mediation and moderation models in within-subjects designs.

As mentioned at the outset of our discussion, however, mediation and moderation, if used and more importantly, *interpreted* incorrectly, can do more harm than good. Far too often in the social sciences are mediational and moderation processes concluded simply because statistical evidence supports it.

To reiterate, and as any good methodologist or philosopher of science will tell you, *no statistical method, whatever the kind, can alone indicate a physical process* (Fiedler et al., 2011). The problem with using these “power” and “action” words to describe statistical processes is *dangerous*. “Causal modeling” as mediational analysis has sometimes come to be known, is not causal at all unless there is a substantive *context* to induce such causality. There is absolutely nothing wrong with conducting a mediation or moderation analysis, but if more researchers actually understood how *little* it truly tells them about their data instead of recklessly imposing *powers* on their variables that they do not



*necessarily* possess, the methods would probably be used less than they currently are. *It behooves all researchers to understand the limitations of the statistical procedures they use before using and interpreting them on empirical data. Researchers regularly “empower” statistics to say much more from their data than they are able. Mathematical and statistical models are first and foremost abstractions; they have no knowledge of your empirical observations, they are simply using those numbers as inputs in their computations. Whether the statistical model says anything substantive about those empirical observations is a much more complex scientific and philosophical issue than simply fitting the statistical model. Narratives rarely correspond to “reality” one-to-one.*

## 5.12 Principal Components Regression

Recall that in the typical multiple regression problem, a linear combination of predictors  $x_1$  through  $x_p$  is sought such that it maximizes the correlation with the response variable,  $y$ . This maximum correlation is expressed via fit statistics such as  $R^2$ . As we know already, multicollinearity can have detrimental effects on a regression model. Having several predictors that have high correlations with one another can, for one, *inflate standard errors*. Inflated standard errors reduce statistical power for detecting effects and hence make it more difficult to reject null hypotheses for predictors.

Statistical issues are not the only reason for disliking collinearity. From a scientific point of view, having correlated predictors is a waste of resources. For example, recording your birth date in months and then again in days is a waste of time, as one variable essentially does the job of two. They are essentially duplicates of one another. More precisely, one variable is a linear combination of the other. In such a case, the most likely solution is to simply drop one of the variables from the regression and simplify the model, both to eliminate or reduce collinearity but also to substantively reduce the scientific complexity of the model.

There are alternatives to simply eliminating variables from the set of predictors, however. One relatively popular alternative approach to this problem is to reduce the dimensionality of  $x_1$  through  $x_p$  *before* making a prediction on  $y$ . Especially if  $p$  (the number of predictors) is very large, it may make more sense to transform  $p$  into a smaller number of *components* and then regress  $y$  onto this smaller number of components. *Principal components regression* attempts to do just this. To understand principal components regression, however, we must first get an idea of what *principal components analysis* is all about.

### 5.12.1 What Is Principal Components Analysis?

A more extensive discussion of principal components analysis (PCA) is reserved for later in the book where we discuss dimension-reduction techniques and manifold learning in more detail, devoting part of an entire chapter to the PCA technique. For now, we only survey the method briefly to learn how it serves the PCA regression technique.

Principal components analysis derives *orthogonal linear combinations* of the variables equal to the number of variables, such that successive linear combinations account for less and less variance in the observed variables. Recall that a linear combination is defined as

$$\ell_i = a_1 y_1 + a_2 y_2 + \cdots + a_p y_p$$

That is, a linear combination is a *weighted sum of products*, where  $a_1$  through  $a_p$  are the weights applied to observed variables  $y_1$  through  $y_p$ . For our data on  $x_1, \dots, x_p$ , the linear combination becomes

$$z_i = a_1x_1 + a_2x_2 + \dots + a_px_p$$

where now  $z_i$  is the name of the linear combination and  $x_1$  through  $x_p$  are the observed variables. A principal component is a unique kind of linear combination such that the weights  $a_1$  through  $a_p$  correspond to values of allied eigenvectors extracted from an eigen analysis performed on the covariance or correlation matrix of variables  $x_1, \dots, x_p$ . Hence, if we denote the weights  $a_1$  through  $a_p$  by  $e_1$  through  $e_p$  to represent eigenvector weights, the first principal component is defined as

$$z_1 = e_1x_1 + e_2x_2 + \dots + e_px_p \quad (5.31)$$

The first principal component accounts for the most variance in the observed variables. The second principal component accounts for the second-most variance in the observed variables but is orthogonal to the first. The goal of PCA is to capture as much variance in the first few components as possible to account for the variance in the set of observed variables. *Principal components regression* uses the first few principal components as predictors in a least-squares sense. In this way, the number of components retained in the PCR (*principal components regression*) can be likened to a *tuning parameter* (Boehmke & Greenwell, 2020). As we will see when we survey methods such as ridge and lasso regression, tuning parameters are useful in controlling the balance between the variance and bias of a statistical learning method and are useful in selecting a “best” model from a wider set of possibilities.

Principal components regression, useful as it may at first appear, makes one important assumption that may with empirical data not always hold. By regressing  $y$  on the set of components, *we are implicitly assuming that the directions of most variance in  $x_1$  through  $x_p$  are those same directions that are associated with  $y$* . While this assumption may not always hold, it is often considered a useful approximation for most data and PCA regression often gives reasonable results (James et al., 2013). As more precisely stated by Izenman (2008, p.132):

The extraction of the principal components is accomplished without any reference to the output variable  $Y$ , we have no reason to expect  $Y$  to be highly correlated with any of the principal components, in particular those having the largest eigenvalues. Indeed,  $Y$  may actually have its highest correlation with one of the last few principal components or even the last one.

Or, as even better elucidated by Artigue and Smith (2018, p. 9):

The promise of principal components regression is that it is an efficient way of selecting a relatively small number of explanatory variables from a vast array of possibilities, based on the correlations among the explanatory variables. The problem is that the eigenvector weights on the candidate variables have nothing to do with their relationship to the variable being predicted. Mildly important variables may be given larger weights than important variables. Nuisance variables may be given larger weights than the true explanatory variables. The coefficients of the true explanatory variables may be given the wrong signs.

Even though principal components regression provides a simple way to use less than the full spectrum of predictors for the problem, it should *not* be regarded as a feature or variable selection method because of the lack of guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response. In fact, it is entirely possible to have variables only weakly related to the response, yet as the number of these variables increases,  $R$ -squared of 1.0 is nonetheless approached. In general, adding useless predictors, though resulting in an increase in dimensionality, will do so at the risk of overfitting and an increase in test error.

A more in-depth discussion of the above problems can be found in Jolliffe (1982). The key takeaway point is that *though PCA regression is guaranteed to derive orthogonal components that maximize variance in the set of predictors, this will not necessarily be equivalent to maximizing  $R^2$  between  $y$  and those components that account for the most variance in  $x_1$  through  $x_p$* . For this reason, PCA regression should not be considered a variable selection technique (James et al., 2013) in the same spirit as techniques such as *best-subsets* or *forward regression*. PCA regression is also not scale-invariant, and hence standardization of predictors is typically (though not *always*) recommended.

### 5.12.2 PCA Regression and Singularity

As we have seen, PCA regression can be useful in situations where we wish to reduce the dimensionality of the predictors before using resulting components as predictors of a response. More to the point, PCA regression can be useful when predictors are highly correlated with one another.

There is another instance where PCA regression may be especially useful, and that is when the matrix of predictors,  $\mathbf{X}$ , is *singular* and hence *ill-conditioned*. Recall that for a singular matrix  $\mathbf{X}$ , it is implied that one row or column is a linear combination of one or more rows or columns. This condition always implies that the determinant of the matrix will equal zero, which implies the inverse of  $\mathbf{X}$  cannot be computed. In such a case, one can substitute a *generalized inverse* when computing least-squares estimates. There are many generalized inverses, but the most common is that of the *Moore-Penrose inverse*, the use of which will yield the generalized inverse regression estimator (Izenman, 2008).

The number of components to keep in PCA regression on the side of  $\mathbf{X}$  can be determined by *Kaiser's rule*, which says to keep all those components with eigenvalues greater than 1 for the correlation matrix, or by using a cross-validation technique (Wold, 1978). The substantive meaning of derived components can also play a significant role in their selection contingent on whether or not the scientific nature of components matters to the scientist.

### 5.12.3 Principal Components Regression in R

The package *pls* (Liland et al., 2023) can be used for principal components regression. We perform a *pls* on the iris data:

```
> require(pls)
> set.seed(1000)
```

We will first remove the species variable:

```
> iris.2 <- iris[c(-5)]
> head(iris.2)
```

|   | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|---|--------------|-------------|--------------|-------------|
| 1 | 5.1          | 3.5         | 1.4          | 0.2         |
| 2 | 4.9          | 3.0         | 1.4          | 0.2         |
| 3 | 4.7          | 3.2         | 1.3          | 0.2         |
| 4 | 4.6          | 3.1         | 1.5          | 0.2         |
| 5 | 5.0          | 3.6         | 1.4          | 0.2         |
| 6 | 5.4          | 3.9         | 1.7          | 0.4         |

We now run the principal components regression model:

```
> pcr_model <- pcr(Sepal.Length~., data = iris.2, scale = TRUE, validation = "CV")
```

In the model statement, `Sepal.Length~.` implies the response as a function of all other variables in the iris data, and `scale = TRUE` requests the data to be standardized. The statement `validation = "CV"` requests cross-validation on the model. To request leave-one-out cross-validation specifically, one would code `validation = "LOO"`:

We obtain a summary:

```
> summary(pcr_model)

Data: X dimension: 150 3
      Y dimension: 150 1
Fit method: svdpc
Number of components considered: 3

VALIDATION: RMSEP
Cross-validated using ten random segments.

      (Intercept)  1 comps  2 comps  3 comps
CV          0.8308  0.5432  0.3867  0.3177
adjCV       0.8308  0.5429  0.3865  0.3173

TRAINING: % variance explained

      1 comps  2 comps  3 comps
X          74.05  98.84 100.00
Sepal.Length 57.97  78.47  85.86
```

The CV number is the *cross-validation*, whereas the adjCV is a bias-corrected CV estimate (Mevik & Wehrens, 2024, p. 5). RMSEP is the *root mean-squared error of prediction*. We also see the *% variance explained* increasing with the number of components (up to three), as it should. We can plot the RMSEP for sepal length on those components (Figure 5.8):

```
> plot(RMSEP(pcr_model), legendpos = "topright")
```

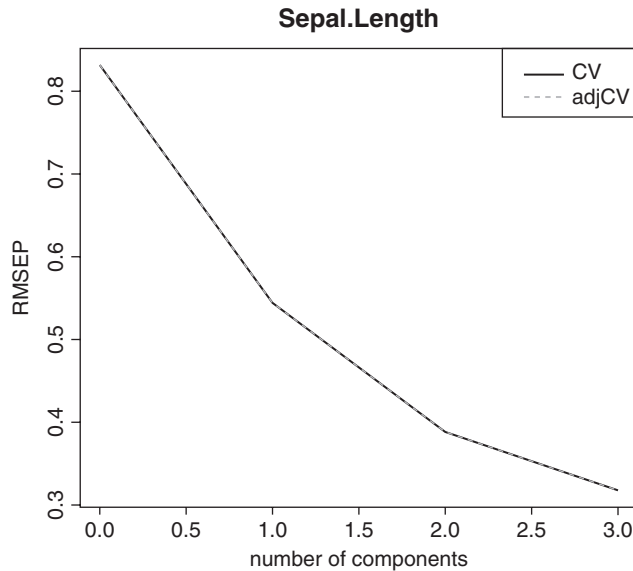


Figure 5.8 RMSEP as a function of number of components. CV and adjCV are virtually the same on this data (the overlap may be difficult to see in the plot).

We can clearly see that RMSEP decreases as the number of components increases, which makes intuitive sense. We could have obtained the above plot through the following code as well (output not shown), where `val.type = "MSEP"` is also requested, where MSEP = mean-squared error of prediction):

```
> validationplot(pcr_model)
> validationplot(pcr_model, val.type="MSEP")
> predplot(pcr_model)
> coefplot(pcr_model)
```

The `predplot()` function reveals a plot (Figure 5.9, left) of predicted values against measured (observed) values for the case of three components. The right plot in Figure 5.9 plots the regression coefficients for each predictor, where `coef(pcr_model)` will yield the corresponding coefficients for sepal width (variable = 1), petal length (variable = 2) and petal width (variable = 3):

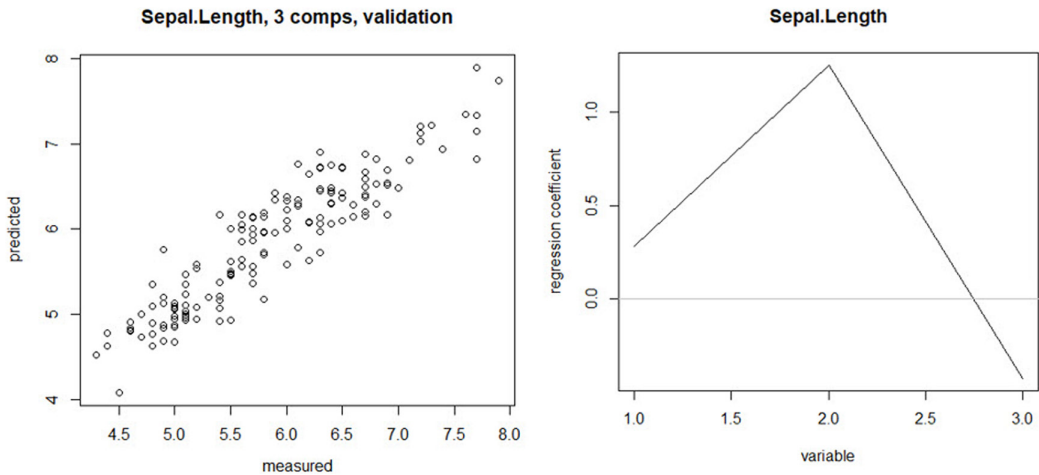


Figure 5.9 Plot of predicted values against measured values (left). Plot of regression coefficients against each predictor (right).

### 5.13 Partial Least-Squares Regression

Whereas principal components regression only uses  $x_1$  through  $x_p$  in its construction of components, *partial least-squares* seeks out dimensionality or directions in the data that incorporates both the predictors *and* the response variable. That is, the components found in  $x_1$  through  $x_p$  are not just relevant to the predictor side, but are also relevant to the response  $y$ . Partial least-squares, or “PLS” for short, is often considered a supervised alternative to principal components regression (James et al., 2013) since PLS will incorporate the response when determining the best “directions.”

As well described by Izenman,

PLS searches for a set of components (called latent vectors) that performs a simultaneous decomposition of  $\mathbf{X}$  and  $\mathbf{Y}$  with the constraint that these components explain as much as possible of the covariance between  $\mathbf{X}$  and  $\mathbf{Y}$ . In this manner, the derived variables in PLS are constructed in such a way as to retain most of the information in the  $X_1$  through  $X_p$  variables and is typically obtained via algorithm rather than optimization procedure (Izenman, 2008, p. 133).

To unpack what PLS actually accomplishes, recall once more what a linear combination looks like in the typical PCA:

$$z_1 = e_1x_1 + e_2x_2 + \cdots + e_px_p$$

In PLS, the coefficients in  $z_1$ , those of  $e_1$ , etc., are set to the coefficients resulting from a simple linear regression of  $y$  on  $x$  for each predictor. This coefficient (e.g., for  $e_1$ ) is proportional to the correlation between  $y$  and  $x_j$ , the given predictor we are considering. That is, in computing the first component, variables  $x$  that are most strongly related to  $y$  (in a univariate sense) receive the most weight.

To obtain the second component,  $z_2$ , each of the variables for  $z_1$  are first adjusted by regressing each variable on  $z_1$  and taking residuals of what is “left over.” If the first component accounts for all the information in the predictors, the residual will equal zero. More than likely, the first

component will not account for all information, and hence the residual will be greater than zero. The second component  $z_2$  is computed using these residuals, which are *orthogonal* to the first component. The iterative approach continues for ensuing directions (i.e., components) of  $z$ .

It is clear then why the method is referred to as *partial* least-squares. It is so because it capitalizes on residuals that are orthogonal to previous component values. Hence, if PLS could talk, it would say, “After the predictive part is removed (or ‘partialled’), let’s now regress the new component on what is left over.” Partial least-squares, as is true of principal components regression, is not scale-invariant, and hence standardizing predictors is often advised. However, as noted elsewhere in the book, the decision to standardize should not be made without careful consideration of the nature (and specifically, *scales*) of the variables under study.

### 5.13.1 Partial Least-Squares in R

The *caret* package (Kuhn, 2008) allows us to fit a variety of models, including that of partial least-squares. The following is a partial least-squares *discriminant analysis* where the response variable is binary:

```
> library(caret)
> library(mlbench)
> data(Sonar)
```

We print only a few variables and cases of the data:

```
> head(Sonar)
  V1      V2      V3      V4      V5      V6      V7      V8      V9      V10     V11     V12     V13     V14
1 0.0200 0.0371 0.0428 0.0207 0.0954 0.0986 0.1539 0.1601 0.3109 0.2111 0.1609 0.1582 0.2238 0.0645
2 0.0453 0.0523 0.0843 0.0689 0.1183 0.2583 0.2156 0.3481 0.3337 0.2872 0.4918 0.6552 0.6919 0.7797
3 0.0262 0.0582 0.1099 0.1083 0.0974 0.2280 0.2431 0.3771 0.5598 0.6194 0.6333 0.7060 0.5544 0.5320
4 0.0100 0.0171 0.0623 0.0205 0.0205 0.0368 0.1098 0.1276 0.0598 0.1264 0.0881 0.1992 0.0184 0.2261
5 0.0762 0.0666 0.0481 0.0394 0.0590 0.0649 0.1209 0.2467 0.3564 0.4459 0.4152 0.3952 0.4256 0.4135
6 0.0286 0.0453 0.0277 0.0174 0.0384 0.0990 0.1201 0.1833 0.2105 0.3039 0.2988 0.4250 0.6343 0.8198

  V56      V57      V58      V59      V60      Class
1 0.0167 0.0180 0.0084 0.0090 0.0032      R
2 0.0191 0.0140 0.0049 0.0052 0.0044      R
3 0.0244 0.0316 0.0164 0.0095 0.0078      R
4 0.0073 0.0050 0.0044 0.0040 0.0117      R
5 0.0015 0.0072 0.0048 0.0107 0.0094      R
6 0.0089 0.0057 0.0027 0.0051 0.0062      R
```





```
method = "pls",
preProc = c("center", "scale")
)
```

Under `method =`, we could have chosen a different method, such as `rpart` (*recursive partitioning*) and decision trees. However, since we are performing partial least-squares, we choose `pls`. The data have also been centered (`center`) and scaled (`scale`) to have a mean equal to zero and standard deviation equal to one. We now look at the output of the fitted model:

```
> plsFit$bestTune
  ncomp
2      2

> plsFit

Partial Least Squares
157 samples
 60 predictor
  2 classes: 'M', 'R'
Pre-processing: centered (60), scaled (60)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 141, 141, 142, 142, 141, 142, ...
Resampling results across tuning parameters:
```

| ncomp | ROC       | Sens      | Spec      |
|-------|-----------|-----------|-----------|
| 1     | 0.8052910 | 0.7259259 | 0.6904762 |
| 2     | 0.8479084 | 0.7495370 | 0.8005952 |
| 3     | 0.8488426 | 0.7638889 | 0.7476190 |
| 4     | 0.8355241 | 0.7652778 | 0.7357143 |
| 5     | 0.8124173 | 0.7481481 | 0.7547619 |
| 6     | 0.7887566 | 0.7236111 | 0.6988095 |
| 7     | 0.7938161 | 0.7439815 | 0.6892857 |
| 8     | 0.8012235 | 0.7393519 | 0.6982143 |
| 9     | 0.7932126 | 0.7578704 | 0.6767857 |
| 10    | 0.7895916 | 0.7412037 | 0.6904762 |
| 11    | 0.7868386 | 0.7416667 | 0.7101190 |
| 12    | 0.7773479 | 0.7365741 | 0.7148810 |
| 13    | 0.7715608 | 0.7375000 | 0.7000000 |
| 14    | 0.7678902 | 0.7175926 | 0.6904762 |
| 15    | 0.7679729 | 0.7148148 | 0.6898810 |

We see from the above output that the largest ROC value (i.e., *area under the ROC curve*, where larger values are “better” than not) occurs when there are three components, with a sensitivity of 0.76 and specificity of 0.74. We can also visualize these results in a plot (Figure 5.10), noting the highest “peak” for ROC occurs when there are two to three components:

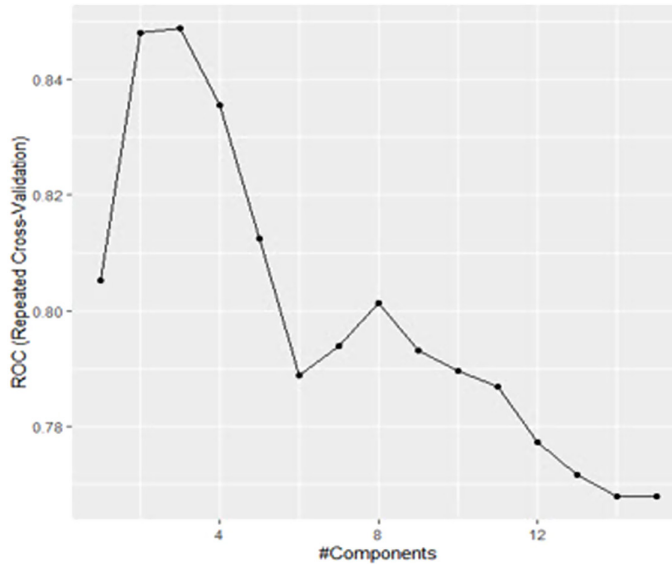


Figure 5.10 ROC as a function of number of components.

The following will compute predicted classes via predicted probabilities (we print only a few cases of both below), which we also summarize in the confusion matrix to follow:

```
> ggplot(plsFit)
> plsClasses <- predict(plsFit, newdata = testing)

> plsClasses
[1] R M M R R R R R R R M R M R R R R R R R M M R M R R M M M M R R R M M R
[39] M M M M M M M M M M M M M
Levels: M R
> plsProbs <- predict(plsFit, newdata = testing, type = "prob")
> head(plsProbs)
```

|    | M         | R         |
|----|-----------|-----------|
| 6  | 0.3526672 | 0.6473328 |
| 8  | 0.5916628 | 0.4083372 |
| 9  | 0.5851591 | 0.4148409 |
| 15 | 0.4297217 | 0.5702783 |
| 26 | 0.4023577 | 0.5976423 |
| 27 | 0.4527808 | 0.5472192 |

The confusion matrix has an accuracy of  $(21 + 18)/51 = 0.76$ , which is also noted in the output, along with a 95% confidence interval. A Kappa statistic (0.5278) is also reported, which assesses the degree of agreement, where higher than not values are preferred. Sensitivity, specificity, and positive and negative predictive values are also noted in the output.

```
> confusionMatrix(data = plsClasses, testing$Class)
```

| Prediction |    | Reference |   |
|------------|----|-----------|---|
|            |    | M         | R |
| M          | 21 | 6         |   |
| R          | 6  | 18        |   |

Accuracy : 0.7647  
 95% CI : (0.6251, 0.8721)  
 Kappa : 0.5278  
 Sensitivity : 0.7778  
 Specificity : 0.7500  
 Pos Pred Value : 0.7778  
 Neg Pred Value : 0.7500

Figure 5.11 (left) is a plot of root mean square error (RMSE) as a function of the number of components. We see that as the number of components increases, RMSE, expectantly, continually decreases.

```
> plot(plsFit)
```

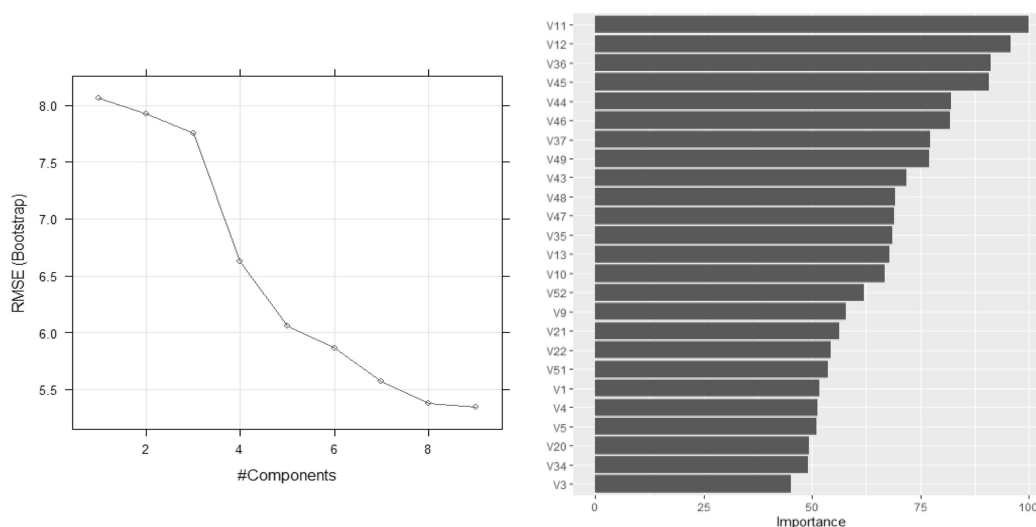


Figure 5.11 RMSE as a function of number of components (left). Variable importance (right).

We can obtain variable importance (Figure 5.11, right) for the first 25 features by

```
> library(vip)
> vip(plsFit, num_features = 25, method = "model")
```

The variable importance measure (`vip`) for `pls` represents the contribution of corresponding coefficients weighted proportionally to their reduction in residual sums of squares (Boehmke & Greenwell, 2020, p. 102). We can see from Figure 5.11 that V11 is the most important variable, while V3 is the least out of the first 25 variables requested (`num_features = 25`).

### 5.13.2 Partial Least-Squares in Python

Partial least-squares regression can be fit in Python using `PLSRegression`. The following is a simple example with easy data:

```
from sklearn.cross_decomposition import PLSRegression
X = [[0., 0., 1.], [1., 0., 0.], [2., 2., 2.], [2., 5., 4.]]
Y = [[0.1, -0.2], [0.9, 1.1], [6.2, 5.9], [11.9, 12.3]]
```

We will request a 2-component solution:

```
pls2 = PLSRegression(n_components=2)
pls2.fit(X, Y)
```

We now obtain predicted values based on the two-component solution:

```
Out [5]: PLSRegression()
Y_pred = pls2.predict(X)

Y_pred
Out [7]:
array([[ 0.26087869,  0.15302213],
       [ 0.60667302,  0.45634164],
       [ 6.46856199,  6.48931562],
       [11.7638863, 12.00132061]])
```

The predicted values on each component extracted above are to be read *vertically* in a column. That is, the first component has predicted values of 0.26, 0.60, 6.46, and 11.76.

## 5.14 Multivariate Reduced-Rank Regression

Multivariate reduced-rank regression (Izenman, 1975) is a very general framework for the multivariate regression model and we mention it here only for pedagogical inclusion into the general regression framework. The classic multivariate linear regression model can be stated in its most general way as

$$\mathbf{Y} = \mu + \mathbf{C}\mathbf{X} + \varepsilon \quad (5.32)$$

where  $\mathbf{C}$  is a matrix of regression coefficients or parameters on which there exists a set of linear constraints. These linear constraints in the case of reduced rank for  $\mathbf{C}$  imply that  $\mathbf{C} = \mathbf{A}\mathbf{B}$ , where both  $\mathbf{A}$  and  $\mathbf{B}$  are *nonunique full-rank matrices* (Izenman, 2008, p. 177), such that we can rewrite the classical multivariate model as

$$\begin{aligned} \mathbf{Y} &= \mu + \mathbf{C}\mathbf{X} + \varepsilon \\ &= \mu + (\mathbf{A}\mathbf{B})\mathbf{X} + \varepsilon \end{aligned} \quad (5.33)$$

where in (5.33), we have simply substituted  $\mathbf{A}\mathbf{B}$  for  $\mathbf{C}$ . That is, the reduced-rank multivariate model generalizes the classical multivariate model by, as summarized by Izenman (2008), relaxing the implicit constraint on the rank of  $\mathbf{C}$ , which by consequence, allows for dimension reduction. The theme is already familiar by now; a matrix of less than full rank may very well have underlying

dimensionality “disguised” within it. By estimating  $\mathbf{A}$  and  $\mathbf{B}$ , we effectively “treat” the reduced-rank condition of  $\mathbf{C}$ . For a proof of why  $\mathbf{A}$  and  $\mathbf{B}$  are nonunique, see Izenman (2008, p. 177).

Several statistical models, including *principal components analysis*, *exploratory factor analysis*, *canonical correlation*, *discriminant analysis*, and even *neural networks*, can be conceptualized as special cases of the reduced-rank multivariate model. What is more, the Gauss-Markov theorem for regression discussed earlier in the book generalizes to the multivariate reduced-rank counterpart. Profile analysis and time series can also be conceptualized within the reduced-rank multivariate model.

### 5.15 Canonical Correlation

The canonical correlation is the bivariate correlation between sets of variables. It is, like Pearson  $r$ , a *linear* correlation. It is the linear correlation between linear combinations of variables. That is, for linear combinations  $\ell_1$  and  $\ell_2$ , the correlation between them is given by:

$$r_{\ell_1, \ell_2} = \frac{\text{cov}_{\ell_1, \ell_2}}{\sqrt{s_{\ell_1}^2 s_{\ell_2}^2}} \quad (5.34)$$

where  $s_{\ell_1}^2$  and  $s_{\ell_2}^2$  are variances of the respective linear combinations (Denis, 2021a) and  $\text{cov}_{\ell_1, \ell_2}$  is the covariance between linear combinations  $\ell_1$  and  $\ell_2$ . The nonzero squared population canonical correlations are the nonzero eigenvalues  $\lambda_i$  of the following (Bilodeau and Brenner, 1999, p. 177; Rencher and Christensen, 2012, p. 387):

$$\Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy}$$

As a result of the eigenvalues providing the *squared* canonical correlations, the *unsquared* canonical correlations are thus equal to the *square root* of the eigenvalues, that is,  $\sqrt{\lambda_i}$ .

How are the squared canonical correlations found? There are a few ways of obtaining them, but perhaps the easiest way is through an eigen analysis of  $\Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy}$ , where such represents the matrix being analyzed in  $\mathbf{A}$  in the classic form of the eigen equality:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Recall that to find  $\lambda$  we set  $|\mathbf{A} - \lambda\mathbf{I}| = 0$ , where now instead of simply a matrix  $\mathbf{A}$ ,

$$\mathbf{A} = \Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy}$$

and so we have

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \quad \rightarrow \quad |\Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy} - \lambda\mathbf{I}| = 0 \quad (5.35)$$

and hence since  $|\mathbf{A} - \lambda\mathbf{I}|\mathbf{x} = 0$ , where  $\mathbf{x}$  is an eigenvector associated with  $\lambda$ , the respective eigenvectors  $\mathbf{a}$  and  $\mathbf{b}$  of  $\Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy}$  are found by

$$(\Sigma_{yy}^{-1} \Sigma_{yx} \Sigma_{xx}^{-1} \Sigma_{xy} - \lambda\mathbf{I})\mathbf{a} = 0 \quad (5.36)$$

and

$$(\Sigma_{yy}^{-1}\Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy} - \lambda\mathbf{I})\mathbf{b} = 0 \quad (5.37)$$

That is, the first linear combination is now  $\ell_1 = \mathbf{a}'\mathbf{x}$  and the second  $\ell_2 = \mathbf{b}'\mathbf{y}$ . Corresponding canonical variates for each dimension are *uncorrelated*, and each subsequent canonical dimension extracted is uncorrelated to previous dimensions. Izenman (2008, p. 216) summarizes canonical correlation:

The CVA technique ensures that every bit of correlation is wrung out of the original X and Y variables and deposited in an orderly fashion into pairs of new variables ... which have a special correlation structure. If the notion of correlation is regarded as the primary determinant of information in the system of variables, then CVA is a major tool for reducing the dimensionality of the original two sets of variables.

Not surprisingly, canonical correlation can be interpreted as a special case of the aforementioned reduced-rank regression (Izenman, 2008). Variations of canonical correlation include *deep variational canonical correlation* (Wang et al., 2017) that extends and generalizes linear canonical correlation to nonlinear models within a deep neural network parameterization. *Probabilistic canonical correlation* is discussed by Bach and Jordan (2006). *Tensor canonical correlation* has also been proposed, where each observation in data consists of multidimensional arrays (tensors) (Min et al., 2020). Relationships between multivariate reduced rank, PCA, and CVA can be found in Izenman (1975).

### 5.15.1 Canonical Correlation in R

The package *CCA* in R (González & Déjean, 2023) can be used for canonical correlation as well as a version of regularized canonical correlation. The package also allows for numerous graphical visualizations of canonical correlation. Below is an example adapted from the package's documentation on the *nutrimouse* data, where gene data is correlated to lipids on a group of mice:

```
> library(CCA)
> data(nutrimouse)
> X = as.matrix(nutrimouse$gene[,1:3])
> Y = as.matrix(nutrimouse$lipid[, 1:3])
```

Above, we designate variables X and Y, where X is gene and Y is lipid. The X variable is defined by variables 1 through 3 of the gene dataframe housed in *nutrimouse*. According to the package details, the gene dataframe contains a  $40 \times 120$  matrix of numerical variables, and thus we are only selecting the first three (1:3). Likewise, we are only selecting variables 1:3 of the dataframe of the 40 rows of lipid (`nrow(nutrimouse$lipid)`) that are in the data. Given that we have three variables on both sides of the equation, we should get three canonical correlations in our output as the number of canonical correlations possible is equal to the lesser of the number of variables on either side of the equation:

```
> head(X)
      X36b4    ACAT1    ACAT2
1 -0.42    -0.65    -0.84
2 -0.44    -0.68    -0.91
3 -0.48    -0.74    -1.10
4 -0.45    -0.69    -0.65
5 -0.42    -0.71    -0.54

> head(Y)
      C14.0    C16.0    C18.0
1  0.34    26.45    10.22
2  0.38    24.04     9.93
3  0.36    23.70     8.96
4  0.22    25.48     8.14
5  0.37    24.80     9.63
```

```
> res.cc = cc(X, Y)
```

The output from `res.cc` will provide a host of canonical correlation output (far too long to list here). It will include such output as  $x$  and  $y$  coefficients as well as  $x$  and  $y$  canonical scores. Most relevant will be the canonical correlations which we can get from

```
> res.cc = cc(X, Y)
> res.cc
$cor
[1] 0.4362942 0.2078600 0.1100969
```

as well as the coefficients for  $X$  and  $Y$ , corresponding to the linear combinations being correlated. There are three sets of coefficients, each set corresponding to the first, second, and third canonical correlations:

```
$xcoef
      [,1]      [,2]      [,3]
X36b4 -2.097072 -3.196288 -15.486386
ACAT1 -6.886185 -15.246034  11.241307
ACAT2 -5.379544  2.058381  1.252734

$ycoef
      [,1]      [,2]      [,3]
C14.0 -0.4372447  1.24826248  1.2095311
C16.0 -0.2332235 -0.06834084 -0.3046945
C18.0  0.2839601  0.53574645  0.2444486
```

The first canonical correlation is equal to 0.43, the second 0.20, and the third 0.11. We can obtain a plot (Figure 5.12) of the above canonical correlations to visualize them in descending order. The `type =` function here controls the appearance of the line in the plot (try `type = "a"` to see the difference):

```
> plot(res.cc$cor, type = "b")
```



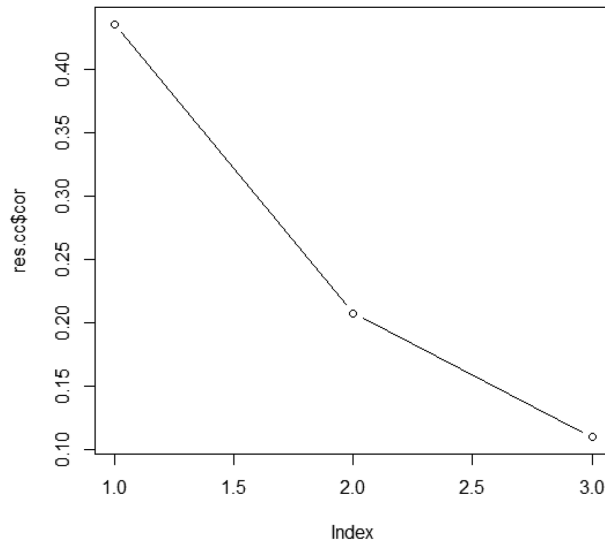


Figure 5.12 Plot of canonical correlations in magnitude of descending order.

The plots in Figure 5.13 allow us to visualize the canonical loadings (a, b) and scores (c) on the first two dimensions:

```
> plt.cc(res.cc)
```

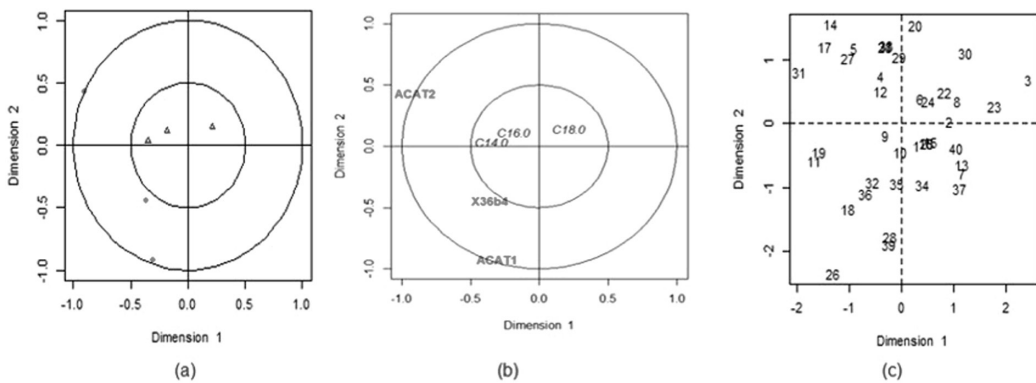


Figure 5.13 Plot of canonical loadings (a, b) and canonical scores (c).

We can easily add variable names to the plot (Figure 5.13, (b)), where we now request the dimension numbers specifically (`d1 = 1`, `d2 = 2`, try `d1 = 2` and `d2 = 3` to plot dimensions two and three, `type = "v"` is for “variables,” try `type = "i"` for “individuals”):

```
> plt.cc(res.cc, d1=1, d2=2, type = "v", var.label = TRUE)
```

The package *mulSEM* (Cheung, 2024) also performs canonical correlation via the function `cancorr()`.

**5.16 Chapter Review Exercises**

1. Calculate the intercept for a simple linear regression model in which the estimated slope is equal to 0, then explain why the resulting intercept makes intuitive sense.
2. What role does sample size play in the computation of the estimated standard error for  $\beta_0$ ?
3. For the *cars* data in R, estimate a simple linear regression where *price* is hypothesized to be a function of *mileage*. Is this the same regression that would result from regressing *mileage* on *price*?
4. What is the consequence of  $(\mathbf{X}^T \mathbf{X})$  not being *invertible* in the computation of  $\beta_1$ ?
5. What is the potential benefit of considering a *biased* estimator in place of an *unbiased* estimator? Explain.
6. Compare the  $C_p$  statistic to the AIC statistic. How are they similar? Different? In the BIC statistic, what role is played by  $\log(n)$ ?
7. Distinguish between *mediation* and *moderation*, and explain the dangers of inferring substantive effects based solely on statistical jargon.
8. Compare *principal components regression* to *partial least-squares regression*. How are these two procedures similar and different from one another?
9. Contrive some data in Python on two variables on ten cases and run a partial least-squares regression. Obtain and report predicted values for the regression.
10. Distinguish between *multivariate reduced-rank regression* and *canonical correlation*. Can one model be subsumed under the other? Why or why not? Justify your answer.

## 6 Regularization Methods in Regression

### Ridge, Lasso, and Elastic Net

In the previous chapter we provided an overview of simple and multiple linear regression methods. We saw that the estimator of choice was an unbiased one, that is, a “BLUE” estimator. The quality of being *unbiased* seemed like a desirable feature to possess. Indeed, from elementary statistics courses onward, students are regularly taught that an unbiased estimator is one of the most favorable qualities of an estimator, in addition to other characteristics such as consistency, sufficiency, etc.

In some cases, however, especially where the matrix of predictors is ill-conditioned (e.g., collinearity), it may be worthwhile to drop the requirement of an unbiased estimator and instead *adopt an estimator that is biased*. At first glance, this may seem statistically unintuitive and borderline reckless. To actually *prefer* a biased estimator seems to go against all essential statistical teaching for favoring unbiased estimators. However, as we will see, choosing a biased estimator in regression has the effect of driving coefficients of predictors toward or to 0, thus potentially deemphasizing predictors that are not essential to the regression solution. By driving coefficients to or toward 0, using a biased estimator can facilitate variable selection in multiple regression. In fact, we will demonstrate in this chapter that the residual sums of squares of ordinary least-squares is nothing more than a special case of regularized regression for a specific choice of hyperparameter.

Three popular estimators that are known to regularize a model are the *ridge*, *lasso*, and *elastic net*. Before discussing each of these, a general and broad introduction and overview of the *principle of regularization* is in order. Regularization as a general concept is not exclusively associated with any particular estimator per se, and shows up in many different statistical and machine learning methods. That is, regularization is an *idea*, not any specific particular methodology. This point is perhaps the most important to understand as it concerns regularization.

#### 6.1 The Concept of Regularization

The main idea of regularization is getting a “handle” on a model to simplify its complexity so that it potentially generalizes better to new test data and hence minimizes overfitting. Equivalently, we can say that the goal of regularization is to improve performance on the test set by adding a penalty to the training loss (Kroese et al., 2020) so that the learner we do select is one that does not overfit on the training data and that generalizes relatively well on the test data. *Regularization is all about preventing overfitting and increasing generalizability*. It is often associated with so-called *shrinkage* and techniques such as ridge regression and the lasso, but it is important to understand that it is much more general and broader than this. One can regularize a model without implementing any specific algorithm. Regularization, the *concept* at least, is very simple and quite general.

To demonstrate the essence of regularization, consider an exceedingly simple example of regularization in the basic linear model:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad (6.1)$$

The above models a linear relationship between response  $y_i$  and predictor  $x_i$ . But suppose the observed sample data exhibit nothing at all like a linear relationship. Suppose instead the plot of  $y_i$  on  $x_i$  is as depicted in Figure 6.1, with a correlation close to 0.

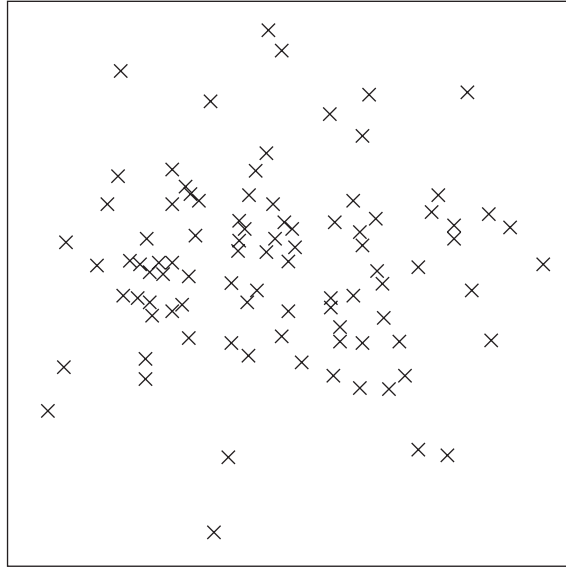


Figure 6.1 Arbitrary plot of data with estimated Pearson Product-Moment correlation equal to (approximately) 0.

Now, consider if we fit the above linear model to this data. Would it fit well? No, and if it does not fit well, it likely will not *generalize* well to test data. To use statistical jargon, even a simple linear model on this data can be considered *overfitting*, and overfitting rarely leads to good generalizability. Hence, one solution is to simplify the model so that it no longer overfits and generalizes better. One way to simplify the model would be to simply constrain the slope parameter to equal 0, or equivalently, not estimate it:

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_i + \varepsilon_i \\ &= \beta_0 + (0)x_i + \varepsilon_i \\ &= \beta_0 + \varepsilon_i \end{aligned} \quad (6.2)$$

This “simpler” model is now just an estimate of the intercept (or “bias” term, to use neural network terminology) along with some error,  $\varepsilon_i$ . Simple as this example may be, *we have just engaged in an act of regularization of a statistical model*. The effect of simplifying the model has another implication; it expresses our belief that the true function is much less complicated than the linear function in this case. Hastie et al. (2009, p. 34) describe this idea beautifully:

Regularization methods, express our prior belief that the type of functions we seek exhibit a certain type of smooth behavior, and indeed can usually be cast in a Bayesian framework.

Their point is that on a technical level, penalties (or *regularization*) reflect to some extent *prior belief*, which is a Bayesian idea. In our example of the line turned to intercept only, the penalty is implicit, and reflects our belief that the true function underlying the data is not as “complicated” as including the slope would suggest. It’s much simpler in that the intercept suffices to account for the true function. This is the *idea* or *spirit* of regularization.

*Regularization can be very informal in practice and can be as simple as deciding to not model the slope in a simple linear regression problem. From a Bayesian point of view, we are imposing our belief that the true functional form underlying the data is simpler than hypothesizing a non-horizontal slope in the regression.*

### 6.1.1 Regularization in Regression and Beyond

Consider for a moment the extent to which the above basic and simple concept might generalize to other modeling activities and methods. For example, when we constrain a parameter estimate in structural equation modeling so as to not overfit and hence improve generalizability, we are, in effect, engaged in model regularization. Likewise, when we decide on a simpler principal component solution instead of a more complex one for the purpose of generalization (such as in *principal components regression*, for instance), we are also regularizing the model. As we will see later, there are a variety of ways of regularizing a neural network, perhaps the most obvious or natural way by reducing the number of hidden layers or the number of nodes. Even *early stopping* of the iterative algorithm in neural networks constitutes a simple way to regularize a neural network.

In general then, anytime a model or method is simplified in some sense, it denotes an act of regularizing that model if such is done in an effort for it to generalize better to test data. In terms of regression, there are three more advanced and common ways of imposing a regularization parameter that go beyond our simple example of constraining the slope above. These include *ridge*, *lasso*, and *elastic net* regression. With the “spirit” of regularization now in place, we now survey these methods.

## 6.2 Ridge Regression

We begin with *ridge regression* because it is among the oldest and most significant and influential of the regularization techniques. These techniques often fall under the umbrella of “shrinkage,” where partial regression coefficients are driven toward or to values of 0. At first glance, it may seem questionable why you might ever want to “shrink” coefficients toward 0, but in the end, shrunken coefficients can greatly simplify, and hence regularize, a model. The ridge penalty (or similar, such as the *lasso*) is also generally known as *weight decay* and the  $L_2$  parameter norm penalty and ridge regression is commonly known as *Tikhonov regularization* in many circles (Goodfellow et al., 2016). The lasso penalty is commonly known as  $L_1$  regularization. Where the ridge penalty is in squared units, that is, a *quadratic* penalty, the lasso penalty is in absolute value units and is hence nonquadratic. The difference in penalties can have important consequences, especially for data in which the number of cases is greater than the number of variables. For an example of this latter point, see Hastie et al. (2009, p. 660). An advantage of  $L_1$  penalization is that such allows for feature selection through driving weights *to*, rather than simply *toward*, 0.

Ridge (and lasso, elastic net) can be applied to situations where one might have literally thousands of predictors, and hence shrinking becomes a good strategy for simplifying the model and making it more interpretable. However, just as regularization is a wide-ranging concept, so is

shrinkage. Shrinkage is merely one way in which a model can be regularized. Be sure this is clear, as there is often some degree of misunderstanding that regularization is *equivalent* to shrinkage, but this is not the case. Subset selection, stepwise regression, forward regression, and even less formal techniques are all methods designed to regularize a model as well. *A regularization method need not be associated with shrinkage.*

As mentioned, another benefit to shrinking parameter estimates relates to the ratio of sample size  $n$  to the number of predictors  $p$ . If  $n$  is not that much larger than  $p$ , there can result inflated variability in the least-squares fit, which can result in overfitting and possibly poor predictions of future observations not used in the training. What is more, if  $p > n$ , there is no longer at all a reliable least-squares estimate, since the variance will be infinite. Under this situation, there exists simply too many ways to project the response variable onto the set of predictors (Hastie et al., 2009). Shrinkage is a tremendously valuable technique in such situations, since by shrinking the estimated coefficients in these cases, both when  $n = p$  and  $p > n$ , we can essentially reduce the variance at the cost of a negligible increase in bias.

### 6.2.1 Mathematics of Ridge Regression

To understand how ridge regression works, first recall ordinary least-squares regression. In least-squares regression,  $\beta_0, \beta_1, \dots, \beta_p$  are estimated via  $\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$  such as to minimize the residual sums of squares (RSS) function,

$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \quad (6.3)$$

Not surprisingly, in ridge regression we also seek to minimize a quantity, but instead through the ordinary least-squares (OLS) estimator

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (6.4)$$

the ridge estimator adds a small quantity  $\lambda$  to the trace of  $\mathbf{X}^T \mathbf{X}$  such that

$$\hat{\beta}_{ridge} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \quad (6.5)$$

The quantity minimized in ridge regression is similar to that in least-squares regression, but with an extra term added:

$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \quad (6.6)$$

Note that the only difference between the two quantities that are minimized is in (6.6), we have added the trailing term  $\lambda \sum_{j=1}^p \beta_j^2$ , where  $\beta_j^2$  is simply the square of  $\beta_j$  which we are already familiar. However, this quantity is “weighted” by  $\lambda$ , which is a *tuning parameter* that is imposed on the sum of the squared regression weights (i.e.,  $\sum_{j=1}^p \beta_j^2$ ). Taken together, the trailing term  $\lambda \sum_{j=1}^p \beta_j^2$  is known as a *shrinkage penalty*. But why add this penalty term to the original residual sums of squares? What is the theoretical consequence of doing so? We explore that motivation now.

### 6.2.2 Consequence of Ridge Estimator

We first note that the residual sums of squares for ridge regression reduces to ordinary least-squares regression when the value for the tuning parameter is equal to 0, since

$$\begin{aligned} RSS &= \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \\ &= \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + 0 \sum_{j=1}^p \beta_j^2 \\ &= \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \end{aligned}$$

For values of  $\lambda$  greater than 0, we see that it has the effect of increasing the value of the residual sums of squares, since we are then adding on at least *some* of  $\sum_{j=1}^p \beta_j^2$  to the least-squares residual.

Hence, ridge regression has the *apparent* effect of *increasing the residual sums of squares over what it would have been had we simply used least-squares regression*. This effect can seem contradictory to the purpose of conducting regression in the first place, as up to now, the goal in regression analysis has always been to *minimize* the objective function, which for classical least-squares has typically been the sum of squared errors. But now, with ridge regression, we are purposely adding on a “penalty” term that would seem to have the effect of *increasing* the sums of squared errors.

Why would we want to do this? We should have a good reason for doing so. It turns out that we do, and that is because we are *no longer using the best unbiased estimator*. Instead, ridge regression uses a *biased estimator*. Ridge regression has an explicit analytic solution, and so numerical methods are not required. Numerical methods will be required when we encounter a related technique called the *lasso*, as a closed-form solution for the lasso does not exist.

To understand why using a biased estimator may be advantageous over employing an unbiased one, we need to revisit the bias-variance tradeoff discussed earlier in the book.

### 6.2.3 Revisiting the Bias-Variance Tradeoff: Why Ridge Is Useful

To appreciate why ridge regression is effective and why using a biased estimator has an advantage over an unbiased one, we need to recall the bias-variance tradeoff concept:

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}\left(\hat{f}(x_0)\right) + \left[\text{Bias}\left(\hat{f}(x_0)\right)\right]^2 + \text{Var}(\varepsilon) \quad (6.7)$$

where  $E\left(y_0 - \hat{f}(x_0)\right)^2$  is the expected test mean squared error (i.e., MSE) at a given value of  $x$ , that of  $x_0$ . The goal is to get  $E\left(y_0 - \hat{f}(x_0)\right)^2$  down or reduced. All else equal in the equality, we see that an increase in  $\left[\text{Bias}\left(\hat{f}(x_0)\right)\right]^2$  will have the effect of increasing  $E\left(y_0 - \hat{f}(x_0)\right)^2$ , not decreasing it, again, *all else equal*. But here’s the key to understanding ridge regression: *not all else is equal*! For an increase in bias, the hope is that the variance of the function, that of  $\text{Var}\left(\hat{f}(x_0)\right)$ , will decrease, which overall, for a constant  $\text{Var}(\varepsilon)$  implies that  $E\left(y_0 - \hat{f}(x_0)\right)^2$  will decrease overall, not *increase*. Cutting right to the chase, *ridge regression works by purposely introducing bias into the estimator to offset a reduction in variance*, which we can summarize succinctly below:

$$\begin{aligned}
E(y_0 - \hat{f}(x_0))^2 &= \downarrow \text{Var}(\hat{f}(x_0)) + \uparrow [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\varepsilon) \\
&= \downarrow E(y_0 - \hat{f}(x_0))^2
\end{aligned} \tag{6.8}$$

That is, the net effect of using a ridge estimator is desirable because it typically results in a smaller MSE on test data. This last point is important. *The smaller MSE is expected (and desired) on test data, not necessarily training data.*

When performing ridge regression, it is common to recommend standardization of input variables prior to carrying out the procedure, despite the recommendation not being accepted by everyone (Izenman, 2008; Thisted, 1976).

#### 6.2.4 A Visual Look at Ridge Regression

Ridge regression can be understood visually and is often referred to as *penalized* least-squares regression. The original motivation for the ridge estimator, as discussed, was the observation made by Hoerl and Kennard (1970) that potential instability in the OLS estimator could be tracked by adding a small constant  $k$  to the diagonal entries of  $\mathbf{X}^T \mathbf{X}$  before taking its inverse. This is what results in the *ridge estimator*, or sometimes called the *ridge rule*. This has the effect of regularizing ordinary least-squares by shrinking toward a multiple of the identity matrix (Friedman, 1989).

But what is the significance of adding this small constant  $k$ ? To understand this, consider the depiction in Figure 6.2 (left) featuring that of concentric ovals or ellipses for OLS, along with the search space for ridge estimators. We can see that the ridge estimate lies on the outer oval of the OLS ellipse, for both  $\beta_1$  and  $\beta_2$  (note that  $\beta_1$  and  $\beta_2$  are on the  $x$  and  $y$  axes, respectively).

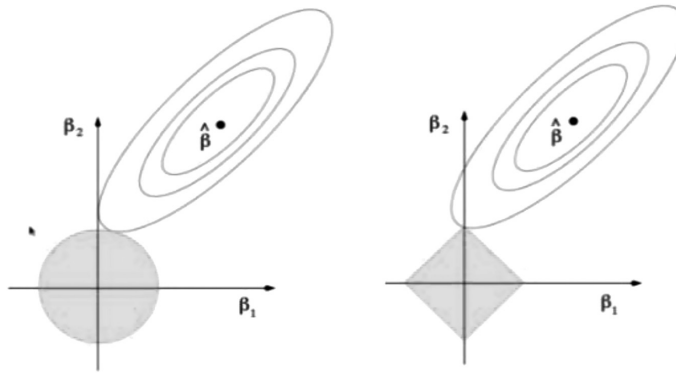


*Figure 6.2* Concentric RSS ellipses for OLS. The ridge estimate is a shrunken version of the OLS estimate moving coefficients toward 0 as a function of a tuning parameter (left). For a constant concentric ellipse, the ridge estimates for  $\beta_1$  and  $\beta_2$  lie as indicated in the plot where both coefficients  $\beta_1$  and  $\beta_2$  have been shrunken toward, but not to, 0. That is, where the ellipse “touches” the circle (it is a circle or “disk” and not another shape as a consequence of the ridge constraint region) is at a point on the circle unequal to 0. In contrast, the lasso solution (right) generates a shape of a diamond constraint region with sharp edges, and hence the lasso solution for  $\beta_1$  and  $\beta_2$  is more likely to have a given coefficient equal to 0.



The concentric ellipses in Figure 6.2 represent possible and constant values of residual sums of squares for coefficients  $\beta_1$  and  $\beta_2$  for each ellipse. That is, no matter where RSS is on a given particular ellipse, it is the *same* value. Ellipses further away from the estimated coefficients have greater RSS. The OLS estimate (and corresponding RSS) in the plot is that RSS produced in regression for a combination of  $\beta_1$  and  $\beta_2$  values. Notice where the corresponding ridge estimate is; it is at a location where both  $\beta_1$  and  $\beta_2$  values have “shrunk” and are thus smaller than the OLS estimates (situated in the center ellipse), though typically not driven all the way to 0 since geometrically ridge features a *circular constraint*. That is, the circular constraint region is effectively the search space for ridge coefficients. That search is most likely to terminate on an edge of the circular region as shown in Figure 6.2 (left). The lasso, on the other hand, features a *diamond* search space with sharp edges instead (as a consequence of the L1 penalty), which illustrates that coefficients will not only be driven smaller, but may be “shrunk” to 0. That is, the search space is this time within the diamond, which means RSS will intersect a hard edge of the diamond rather than a smooth part of the circular region as in ridge. Hence, the lasso coefficient “lands” on a 0, such as for  $\beta_1$ , as the first point that touches the lasso constraint region, that is, the lasso search space. For a more thorough discussion related to interpreting ridge (and lasso) regression this way, see James et al. (2013) for a deeper explanation of the geometry of ridge vs. lasso regression.

### 6.2.5 Ridge Regression in Python

Ridge regression can be performed very easily in Python. We create a simple example from *sklearn*’s ridge library:

```
from sklearn.linear_model import Ridge
import numpy as np
n_samples, n_features = 10, 5
```

Note from the above that we have designated the number of samples equal to 10 and the number of features equal to 5, which we confirm below:

```
n_samples
Out[97]: 10
n_features
Out[98]: 5
```

We now produce some random data on those samples and features, where `rng.randn()` generates data from a standard normal distribution:

```
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)

y
Out[103]:
array([ 1.76405235,  0.40015721,  0.97873798,  2.2408932 ,  1.86755799,
        -0.97727788,  0.95008842, -0.15135721, -0.10321885,  0.4105985 ])
```

Note that the  $y$  values consist of a single vector, while the  $X$  values below comprise five different variables, each having ten cases (to generate standardized data, one can use `StandardScaler()`, more on this later):

```
X
Out [104]:
array([[ 0.14404357,  1.45427351,  0.76103773,  0.12167502,  0.44386323],
       [ 0.33367433,  1.49407907, -0.20515826,  0.3130677 , -0.85409574],
       [-2.55298982,  0.6536186 ,  0.8644362 , -0.74216502,  2.26975462],
       [-1.45436567,  0.04575852, -0.18718385,  1.53277921,  1.46935877],
       [ 0.15494743,  0.37816252, -0.88778575, -1.98079647, -0.34791215],
       [ 0.15634897,  1.23029068,  1.20237985, -0.38732682, -0.30230275],
       [-1.04855297, -1.42001794, -1.70627019,  1.9507754 , -0.50965218],
       [-0.4380743 , -1.25279536,  0.77749036, -1.61389785, -0.21274028],
       [-0.89546656,  0.3869025 , -0.51080514, -1.18063218, -0.02818223],
       [ 0.42833187,  0.06651722,  0.3024719 , -0.63432209, -0.36274117]])

clf = Ridge(alpha = 1.0)
clf.fit(X, y)
Out [146]: Ridge()
```

The default  $\alpha$  in the above is 1.0. We could adjust it to other values. The smaller the value for  $\alpha$  (*regularization* or *tuning* parameter), the more the solution will resemble ordinary least-squares regression. Recall that *for an  $\alpha$  of 0, the two solutions will converge*. As sample size increases and  $\alpha \rightarrow 0$ , both ridge and linear regression will also converge to generate the same output. With enough training data, the impact of regularization usually also becomes less prominent.

Now, let's define some new data on each of the five features and predict a value based on that new data. The new data we will define by the name `row`:

```
row = [0, 2, 3, 4, 5]
yhat = clf.predict([row])
print('Predicted: %.3f' % yhat)
Predicted: 4.059
```

We see that for the above new data on `row`, the predicted value for  $y$  is equal to 4.059.

We now cross-validate the solution. The `RepeatedKfold` cross-validator in Python performs  $k$ -fold  $n$  times with different randomization in each repetition. We will request ten splits (i.e., “folds”) with three repeats:

```
import numpy as np
from sklearn.model_selection import RepeatedKfold
cv = RepeatedKfold(n_splits=10, n_repeats=3, random_state=1)
grid = dict()
grid['alpha'] = np.arange(0, 1, 0.01)
```

The above values from 0 to 1 in increments of 0.01 are values of the regularization parameter  $\alpha$ . We will now run a *grid search* that scans the parameter values, finding the one that gives the smallest mean absolute error:

```

search = GridSearchCV(clf, grid, scoring='neg_mean_absolute_error',
cv=cv, n_jobs=-1) results = search.fit(X, y)
print('MAE: %.3f' % results.best_score_)
MAE: -0.620

print('Config: %s' % results.best_params_)
Config: {'alpha': 0.13}

```

Above, we see that the *negative mean absolute error* (i.e., MAE) is equal to  $-0.620$  based on the most optimal parameter, equal to an alpha of  $0.13$ .

Python also offers *Bayesian ridge regression*, where regularization parameters are updated systematically. To implement Bayesian ridge regression, see the `BayesianRidge()` function in Python. Also, for the example featured here, because we drew random data from a standard normal distribution (i.e., the data drawn were on the same *scale*), we did not *standardize* the resulting data to have a mean equal to 0 and standard deviation equal to one. However, such does not imply that standardization would not be worthwhile here. This is especially true in situations where variables occur naturally on *different scales* where one does not want the “importance” of one variable to exceed that of another simply due to the scale on which it is being measured. On the other hand, it is possible that this kind of “Bayesian prior” in terms of the *original scale on which measurements are taken* is precisely what is desired depending on the research setting and context, in which case the decision to standardize or not is a more difficult decision (Izenman, 2008). Either way, it is a good idea to compare results both with and without standardization when conducting most regularization methods. If one must choose one option over the other, it is probably a good idea to standardize (Hastie et al., 2009, p. 63) in most cases, but such should not be done without great care and consideration.

Indeed, beyond regularization methods, the decision of whether or not to standardize must be made in many other methods as well (e.g., principal components analysis). Offering a “cookbook” recommendation to “always standardize” is not good statistical teaching, especially without carefully considering whether or not it is desirable that the original variances of the measurement units play a role in the ensuing statistical analyses. Once more with the Bayesian prior concept, sometimes we want those variances to impact the solution, whereas other times we do not. The key point is that neither solution is definitively right or wrong, and the decision to keep measurements in their original units or transform them to standardized units needs to be made on a case-by-case basis.

### 6.2.6 Ridge Regression in R

Ridge regression in R can be performed using the *lmridge* package (Imdad & Aslam, 2018). For an excellent summary of this package, see Ullah et al. (2018). The package is *outstanding* in its coverage of all things “ridge.” In the following, we fit a ridge regression to the *Hald* data. The data relate to an engineering application that was concerned with the effect of the composition of cement on heat evolved during hardening. What is most relevant about this data for our purposes is that the features are relatively highly correlated and hence well-suited for ridge regression. Consult the package for further details on this specific example.

Let’s first briefly survey the Hald data:

```

> library(lmridge)
> data(Hald)
> head(Hald)

```

|      | Y     | X1 | X2 | X3 | X4 |
|------|-------|----|----|----|----|
| [1,] | 78.5  | 7  | 26 | 6  | 60 |
| [2,] | 74.3  | 1  | 29 | 15 | 52 |
| [3,] | 104.3 | 11 | 56 | 8  | 20 |
| [4,] | 87.6  | 11 | 31 | 8  | 47 |
| [5,] | 95.9  | 7  | 52 | 6  | 33 |
| [6,] | 109.2 | 11 | 55 | 9  | 22 |

We will fit a model where  $y$  is hypothesized as a function of predictors  $X1$  through  $X4$ :

```
> mod <- lmridge(y ~ X1 + X2 + X3 + X4, data = as.data.frame(Hald),
+ scaling = "sc", K = seq(0, 1, 0.001))
```

The “sc” for scaling scales the predictors to correlation form. The option `scaling = “centered”` would have simply centered all predictors. The option `K =` designates the ridge biasing *tuning* parameter, where in the above we chose options 0, 1, 0.001, which means R will compute coefficients starting at 0 up to 1, at steps or increments of 0.001.

We now call the fitted model which shows parameter estimates for values of  $K$  0–0.004. These are the *raw* or “de-scaled” weights fitted by the ridge regression:

```
> mod
Call:
lmridge.default(formula = y ~ X1 + X2 + X3 + X4, data = as.data.frame(Hald),
  K = seq(0, 1, 0.001), scaling = “sc”)
```

|         | Intercept | X1      | X2      | X3       | X4       |
|---------|-----------|---------|---------|----------|----------|
| K=0     | 62.40537  | 1.55110 | 0.51017 | 0.10191  | -0.14406 |
| K=0.001 | 71.32028  | 1.45679 | 0.41874 | 0.00650  | -0.23358 |
| K=0.002 | 75.32580  | 1.41283 | 0.37795 | -0.03741 | -0.27346 |
| K=0.003 | 77.60713  | 1.38657 | 0.35493 | -0.06322 | -0.29591 |
| K=0.004 | 79.08396  | 1.36858 | 0.34021 | -0.08058 | -0.31023 |

The following will yield *scaled* coefficients:

```
> mod$coef
```

|    | K=0       | K=0.001     | K=0.002     | K=0.003    | K=0.004    | K=0.005    |
|----|-----------|-------------|-------------|------------|------------|------------|
| X1 | 31.607147 | 29.6853546  | 28.7895380  | 28.254405  | 27.887857  | 27.613799  |
| X2 | 27.500315 | 22.5718760  | 20.3730129  | 19.132537  | 18.339056  | 17.790004  |
| X3 | 2.261167  | 0.1442803   | -0.8300639  | -1.402819  | -1.787926  | -2.070135  |
| X4 | -8.353056 | -13.5438292 | -15.8561502 | -17.157745 | -17.987880 | -18.560165 |

We can obtain predicted values as follows:

```
> predict(mod)
```

|   | K=0       | K=0.01    | K=0.02    | K=0.03    | K=0.04    | K=0.05    | K=0.06    |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 1 | 78.49535  | 78.49063  | 78.65269  | 78.81280  | 78.96609  | 79.11210  | 79.25112  |
| 2 | 72.78893  | 73.10007  | 73.26551  | 73.41722  | 73.56132  | 73.69948  | 73.83256  |
| 3 | 105.97107 | 106.39463 | 106.38297 | 106.34900 | 106.30959 | 106.26726 | 106.22416 |
| 4 | 89.32720  | 89.48225  | 89.48739  | 89.48752  | 89.48842  | 89.49055  | 89.49423  |
| 5 | 95.64939  | 95.70913  | 95.84146  | 95.96400  | 96.07594  | 96.17731  | 96.26972  |
| 6 | 105.27469 | 105.27361 | 105.21815 | 105.16162 | 105.10662 | 105.05234 | 104.99962 |
| 7 | 104.14885 | 104.10473 | 104.09527 | 104.08482 | 104.07302 | 104.05858 | 104.04282 |

Many other statistics are available in the ridge package for the fitted model. For instance, we can obtain a plot, the so-called *ridge trace plot* of the fitted model as given in Figure 6.3.

```
> plot(mod)
```

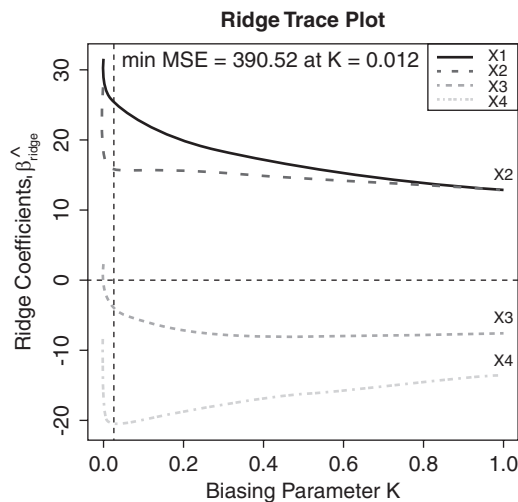


Figure 6.3 Ridge trace plot featuring ridge coefficients as a function of the biasing parameter  $K$ .

The plot in Figure 6.3 shows the value of ridge coefficients for various values of  $K$ . We can see that the minimum MSE occurs at  $K = 0.012$ . We can also generate a summary of the model via `summary(mod)`, which will produce parameter estimates, standard errors, and other model information.

Cross-validation plots can also be obtained using `cv.plot()`, and AIC and BIC values can be obtained as in Figure 6.4, showing where AIC and BIC values are at their minimums as a function of degrees of freedom (DF):

```
> info.plot(mod)
```

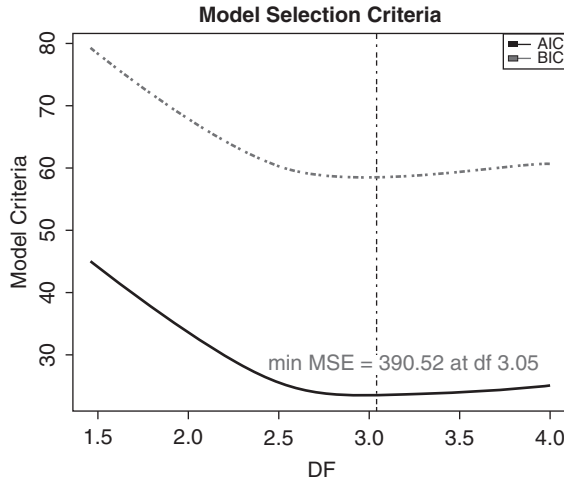


Figure 6.4 AIC and BIC as a function of degrees of freedom.

We can also get vectors of AIC and BIC values by  $K$  values via the following (output not shown): `infocr(mod) [, 1]`, `infocr(mod) [, 2]`. Our survey of the capabilities of *lmridge* only scratches the surface of what the package is capable of generating. See the package's details for a wealth of further options.

### 6.3 Lasso Regression

The ridge regression just featured is only one way that we may shrink parameter estimates. A second popular and modern way is through lasso regression. In lasso regression, the idea of minimizing  $E(y_0 - \hat{f}(x_0))^2$  over and above what OLS regression would accomplish remains through imposing a penalty term, but this time, instead of the penalty  $\lambda \sum_{j=1}^p \beta_j^2$  as in ridge, the penalty term is  $\lambda \sum_{j=1}^p |\beta_j|$  so that we are now minimizing the following:

$$RSS = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \quad (6.9)$$

That is, instead of the tuning parameter being applied to the sum of squared regression weights,  $\sum_{j=1}^p \beta_j^2$ , it is now applied to the sum of the *absolute value* regression weights,  $\sum_{j=1}^p |\beta_j|$ . Be sure to note however that whether adding a squared term in ridge regression or absolute value term in lasso regression, the general principle is the same, and that is to include a penalty term to the least-squares residual sum of squares to seemingly increase the error function. But again, it is worth asking why we would want to do such a thing. As before with ridge, it becomes “worth it” if such an addition of the penalty reduces MSE on test (not necessarily *train*) data. The only difference between ridge and lasso is the nature of the penalty. In both cases, as lambda  $\lambda$  increases, all coefficients in the regression will eventually go to 0, and hence a “stopping point” will be required; otherwise performing ridge or lasso regression would be trivial and meaningless. That is, while we would like to shrink some coefficients to or toward 0, driving all coefficients to 0 would be

counterproductive. As we will soon see, in practice, the stopping point is typically selected via cross-validation.

According to Müller and Guido (2017), ridge is usually the first choice between that of ridge vs. lasso. Lasso might be most useful if you have a large number of predictors and expect only a few of them to be important, so that the rest can drop out. Remember, lasso can be used for variable selection because it drives some coefficients all the way to 0. This is the key distinguishing feature, in an applied sense, between using ridge vs. the lasso.

### 6.3.1 Lasso Regression in Python

We can perform lasso regression in Python using `sklearn.linear_model` by importing the `lasso` function. As with ridge, the tuning parameter is adjusted via `alpha()`:

```
import numpy as np
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha = 0.1)
```

Let's first build some random data. We will request three columns for  $X$  and a single column for  $y$ :

```
X = 2*np.random.rand(10, 3)
```

The designation  $(10, 3)$  requests 10 rows by 3 columns, and hence 3 variables, each having 10 values:

```
X
Out [171]:
array([[1.83514679e-03, 1.22586651e+00, 1.36877003e+00],
       [1.98425878e+00, 1.27078700e+00, 3.05682926e-01],
       [1.67601853e+00, 1.91988490e+00, 1.14380535e+00],
       [6.85308266e-01, 1.74952610e+00, 2.58223320e-01],
       [1.92985175e+00, 1.33611596e+00, 1.69223164e+00],
       [1.70564639e+00, 9.98607007e-01, 2.89554744e-01],
       [1.91331572e+00, 1.39363873e+00, 1.78534834e+00],
       [9.52425306e-01, 4.84902989e-01, 1.77936410e+00],
       [1.95151855e+00, 1.89191263e+00, 6.02149503e-01],
       [9.86734995e-01, 1.14731519e+00, 1.30875638e+00]])
```

```
y = np.random.randn(10, 1)
```

```
y
Out [167]:
array([[ 0.89068823],
       [ 0.55519131],
       [-1.23040405],
       [-1.24752468],
       [-0.20588354],
       [-0.86695762],
       [-1.10404141],
       [-0.74561481],
       [-2.19565951],
       [ 1.28035892]])
```

```
lasso_reg.fit(X, y)
```

We can now implement the lasso regression and obtain predicted values on  $y$  for a set of values on  $X$ , 1.5, 1, 2:

```
lasso_reg.predict([[1.5, 1, 2]])
Out[177]: array([-0.39427847])
```

We see the predicted value for the set of values entered for predictors is equal to  $-0.39$ . Lasso regression can also be fit using `statsmodels` in Python.

### 6.3.2 Lasso Regression in R

Lasso regression in R can be fit using `glmnet()` in the *glmnet* package (Friedman et al., 2023). As was true for *lmridge*, *glmnet* is an outstanding package. The options are endless for performing lasso regression and a host of other regression models. Hastie et al. (2023) also published *An Introduction to glmnet*, which is equally outstanding and strongly recommended for getting started with *glmnet*.

We feature an example from the package:

```
> library(glmnet)
> data(QuickStartExample)
> x <- QuickStartExample$x
> y <- QuickStartExample$y
```

There are four predictors in  $x$  and a single response in  $y$ . We now fit the model (printing only a few cases):

```
> fit <- glmnet(x, y)
> fit
Call: glmnet(x = x, y = y)
```

|   | Df | %Dev  | Lambda  |
|---|----|-------|---------|
| 1 | 0  | 0.00  | 1.63100 |
| 2 | 2  | 5.53  | 1.48600 |
| 3 | 2  | 14.59 | 1.35400 |
| 4 | 2  | 22.11 | 1.23400 |
| 5 | 2  | 28.36 | 1.12400 |

The number of nonzero coefficients is indicated by DF, with %Dev indicating the percent of deviance explained for various values of  $\lambda$ , where the higher the deviance, the more that is explained by that particular case. The percent deviance explained is relative to the null deviance, which is a baseline model containing only the intercept term.

The `plot()` function (Figure 6.5) reveals the coefficient growth as a function of  $\lambda$  and consequently the L1 norm  $\sum |\beta_j|$  of the coefficient vector, where the top axis (0–9) shows the number of nonzero coefficient for the given value of L1:

```
> plot(fit)
> plot(fit, label = TRUE)
```



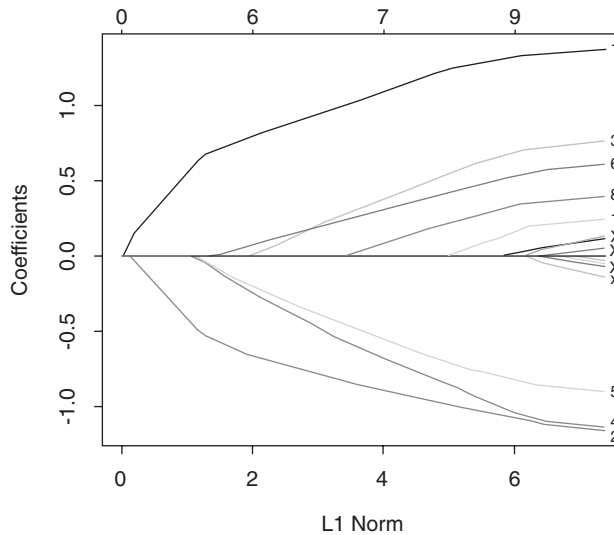


Figure 6.5 Estimated coefficients as a function of L1 norm. At L1 norm = 0, all coefficients have been driven to 0 and lambda (i.e., tuning parameter) is large, whereas as L1 norm increases, the OLS solution is approximated for large values and where lambda is increasingly smaller.

Note that as values of L1 decrease toward 0, the “extreme” lasso solution is obtained with all coefficients equal to 0 implying that the number of nonzero coefficients (upper axis) is equal to 0. The right-most axis corresponds to the variable numbers. As evident in the plot, variable 1 has the largest coefficient. The second-largest (in magnitude) is variable 14 (the plot has been shrunk somewhat and so it cannot be seen; variable 14 is at the extreme lower right of the plot).

We can examine different models contingent upon values of the regularization parameter,  $s$ . For instance, in the following, we look at coefficients given a value of  $s = 0.1$  (try different values for  $s$  to see what you get; if you select a very large value, all coefficients will be driven to 0, which is not the point of performing the lasso):

```
> coef(fit, s = 0.1)
21 x 1 sparse Matrix of class "dgCMatrix"
```

```

              s1
(Intercept)  0.150928072
V1           1.320597195
V2           .
V3           0.675110234
V4           .
V5          -0.817411518
V6           0.521436671
V7           0.004829335
V8           0.319415917
V9           .
V10          .
V11          0.142498519
V12          .
V13          .
```

```

V14      -1.059978702
V15      .
V16      .
V17      .
V18      .
V19      .
V20      -1.021873704

```

Coefficients above that have been driven to 0 as a result of the lasso are simply represented by periods. For instance, V2 and V4 have been driven to 0. We can obtain a plot of variable importance (Figure 6.6) by the following, where such importance is defined by the magnitude of standardized coefficients (Boehmke & Greenwell, 2020, p. 136):

```

> library(vip)
> vip(fit)

```

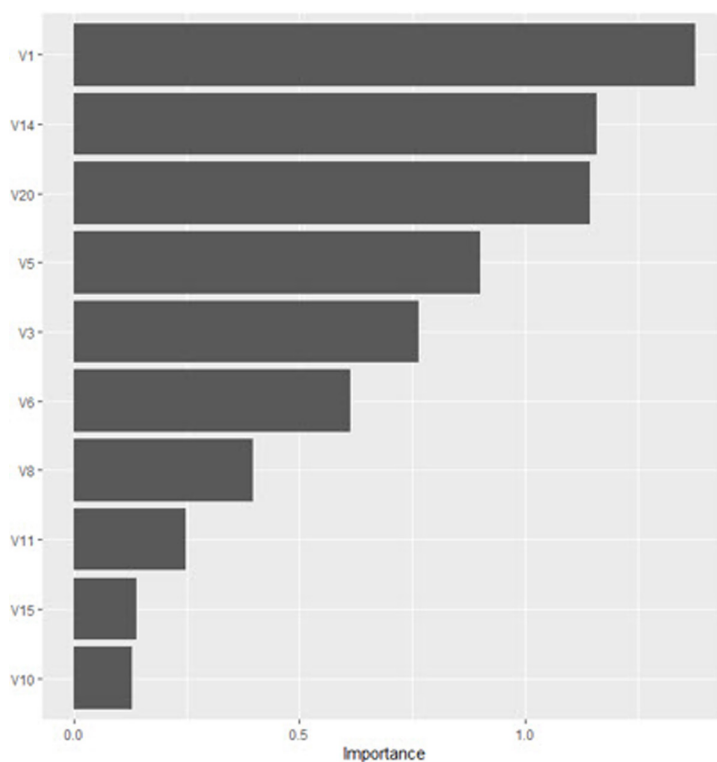


*Figure 6.6* Plot of variable importance for lasso regression, where “importance” is the value of the standardized coefficients. For these data, variable 1 is the most important variable, followed by variable 14.

We can also perform cross-validation directly using the `glmnet()` function (Figure 6.7), which performs  $k$ -fold cross-validation (ten-fold by default).

```
> set.seed(123)
> cvfit <- cv.glmnet(x, y)
> plot(cvfit)
```

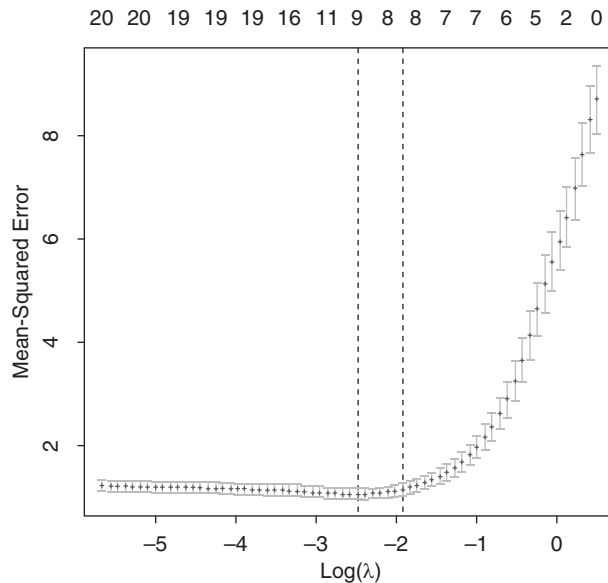


Figure 6.7 Mean squared error as a function of (log) lambda. The left vertical line represents the value of lambda for which mean squared error is a minimum.

We can extract the horizontal line value (left vertical line) along with the corresponding coefficient for this value:

```
> cvfit$lambda.min
[1] 0.07569327
```

The value of lambda that generates the minimum is thus equal to 0.076. The natural log of this number is equal to  $-2.58$ , which is the left line shown in the above plot (Boehmke and Greenwell, 2020, p. 131). The vertical line to the right is that of `lambda.1se`, which is lambda equal to one standard error of the aforementioned minimum (`cvfit$lambda.1se`). We now obtain coefficients based on this minimum value (we print only the first few):

```
> coef(cvfit, s = "lambda.min")
21 x 1 sparse Matrix of class "dgCMatrix"
```

```

              s1
(Intercept)  0.14867414
V1           1.33377821
V2           .
V3           0.69787701
V4           .
V5          -0.83726751
```

Hence, via cross-validation, we have settled on a set of coefficients that we could deem “final” for our lasso regression, or could have also explored coefficients in the neighborhood of `lambda.min`. Automating the tuning parameter can further be performed using the *caret* package (Kuhn, 2008). Boehmke and Greenwell (2020, p. 133) provide an excellent summary of how to use the *caret* package for this purpose, where a *grid search* over values of `lambda` for various values of `alpha` is performed, where the `alpha` setting indicates whether we have performed ridge, lasso, or elastic net (to be surveyed next). That is, such a search will reveal the model with both the best (i.e., lowest RMSE) `alpha` parameter *and* `lambda` combined. A similar approach for searching for both `alpha` and `lambda` can be performed using `cv.glmnet` (see the package for details).

## 6.4 Elastic Net

A third option for shrinking parameters is that of *elastic net*, with penalty equal to

$$\sum_{j=1}^p (\alpha |\beta_j| + (1-\alpha)\beta_j^2) \quad (6.10)$$

Elastic net is considered a compromise of sorts between the ridge and lasso, a kind of “hybrid” of the aforementioned estimators (Hastie et al., 2009). It can be shown that elastic net actually employs a linear combination of the ridge and lasso penalty functions (see Izenman, 2008, p. 153 for details).

Note that elastic net features two coefficients,  $\beta_j^2$  and  $|\beta_j|$ . To understand the role of  $\alpha$ , the tuning parameter in this context, note that it is being applied to both  $\beta_j^2$  and  $|\beta_j|$ , both the losses that are used in the ridge and lasso, respectively (i.e., quadratic and absolute function losses). Hence, we can see that the role of  $\alpha$  is to differentially weight  $\beta_j^2$  and  $|\beta_j|$ . This is why elastic net is interpreted as a “compromise” between ridge and lasso, and such compromise depends on the chosen value for  $\alpha$  in differentially weighting the ridge and lasso L2 and L1 losses, respectively.

### 6.4.1 Elastic Net in Python

The library `scikit-learn` can perform elastic net. In what follows we fit an elastic net model on two features:

```
from sklearn.linear_model import ElasticNet
from sklearn.datasets import make_regression
X, y = make_regression(n_features=2, random_state=0)
regr = ElasticNet(random_state=0)
```

We print only a few cases of  $X$  and  $y$  below:

```
X
Out [39]:
array([[ 1.46935877,  1.53277921],
       [-1.60205766,  0.62523145],
       [-1.34775906,  1.8831507 ],
       [ 1.22244507, -0.40317695],
       [ 1.5430146 , -0.739563 ],
       [ 0.70657317,  0.3563664 ],
       [ 0.37816252,  0.15494743],

y
Out [40]:
array([ 1.90370562e+02,  1.33395696e+01,  1.41772669e+02, -3.06962683e+00,
       -2.60624653e+01,  5.49225405e+01,  2.59528810e+01, -4.33086794e+01,
        2.47135869e+02, -1.07140916e+02, -8.57587584e+01, -3.79043198e+01,
       -7.95749661e+01,  1.86961487e+01,  4.93789716e+01,  5.32933539e+00,
       -1.16562784e+02,  5.63423160e+01, -8.36280817e+01,  7.68252760e+01,
```

We fit the elastic net regression and obtain coefficients below:

```
regr.fit(X, y)
Out [689]: ElasticNet(random_state=0)

print(regr.coef_)
[18.83816048  64.55968825]

print(regr.intercept_)
1.4512607561654032
```

We can see that the regression coefficients are equal to 18.84 and 64.56 for each feature, respectively, while the intercept is equal to 1.45. Let's now obtain a prediction for values of 0 and 0 on each feature:

```
print(regr.predict([[0, 0]]))
[1.45126076]
```

Not surprisingly, we see that the predicted value is equal to the intercept of 1.45 (try other values to see what predicted values you get).

As we did for the lasso, we can use the package *glmnet* to fit elastic net in R. We can also use a recently-proposed package *fasterElasticNet* (Ma et al., 2022) that purports to fit elastic net (and related models) very efficiently.

### 6.5 Which Regularization Penalty Is Better?

To conclude our discussion of these regularization approaches, it should be noted that neither ridge nor lasso nor elastic net will universally dominate each other. *There is no free lunch!* Lasso might perform better when a few coefficients are close to 0 and the rest quite substantial, where ridge often performs best when all coefficients are approximately equal, or at minimum, of equal importance to the investigator. However, there are no guarantees. The use of *cross-validation* in some capacity is recommended to determine which approach is better for a given data set, with tuning parameters selected via *grid search* on different values and cross-validation employed to observe which gives the lowest *test error*. To reiterate, we want to minimize the *test error*, not necessarily *train error*. This is why cross-validation is so important to choosing an optimal tuning parameter. See Frank and Friedman (1993) for a comparison of ordinary least-squares to not only ridge regression, but also principal components regression and partial least-squares.

### 6.6 Least-Angle Regression

As we have seen when surveying variable selection techniques, forward regression selects variables in the candidate pool one at a time that improves model fit. *Least-angle regression* can be seen as a related technique to forward regression as it still selects based on a forward strategy, but does not necessarily select the *entire* variable. Instead, what enters into the selection is as much of the variable that is *merited* to the prediction, not all of it as in traditional stepwise routines. Hastie et al. (2009, pp. 73–74) describe the algorithm most succinctly:

At the first step it [least-angle regression] identifies the variable most correlated with the response. Rather than fit this variable completely, LAR (least-angle regression) moves the coefficient of this variable continuously toward its least-squares value (causing its correlation with the evolving residual to decrease in absolute value). As soon as another variable “catches up” in terms of correlation with the residual, the process is paused. The second variable then joins the active set, and their coefficients are moved together in a way that keeps their correlations tied and decreasing. This process is continued until all the variables are in the model, and ends at the full least-squares fit.

The algorithm begins with standardizing predictors to have a mean equal to 0 and variance equal to 1 (i.e., *z-scores*). At this initial step, the residual  $r_i$  is computed having no predictors in the model, that is,  $r_i = y_i - \bar{y}$ , with all coefficients in the regression set to 0. That is,  $\beta_1 = \beta_2, \dots, \beta_p = 0$ . Next, the predictor having the largest correlation with this residual is entered into the model; however, *not all of it*. The predictor’s weight moves *toward* its least-squares coefficient until another predictor “catches up” and has as much correlation with the current residual as the first predictor. At this point, both  $\beta_1$  and  $\beta_2$  move in the direction of their *joint* least-squares coefficient until another predictor “catches up” to the current residual. The algorithm proceeds in this way until all predictors are entered and hence the least-squares solution is achieved for the model. In the case where the number of predictors is greater than sample size (more specifically,  $N-1$ ), the algorithm will reach a zero residual condition after  $N-1$  steps (Hastie et al., 2009, p. 74).

Hence, least-angle regression builds up a regression model by piecewise linear “partial” steps, adding single elements at each step by correlating the new variable with the current residual. As another variable “catches up,” the algorithm moves in a direction *equiangular* (i.e., having the

same angle) to predictors. For this reason, least-angle regression has been called the “democratic” version of stepwise (Hastie et al., 2009). Both the lasso and forward stagewise regression models can be computed through the *least-angle regression algorithm* (Izenman, 2008). The key point to emphasize is that in least-angle regression, predictors essentially “catch up” to others in the selection process, then those variables move together. In Python, the `Lars` estimator can be used to fit a least-squares regression. The LARS lasso can also be used which implements the LARS estimator while driving some coefficients to 0, as is characteristic of the lasso.

### 6.6.1 Least-Angle Regression in R

We can fit a least-angle regression in R using the *lars* package (Hastie and Efron, 2022). The *lars* package fits least-angle regression, lasso, and forward stagewise regression. We demonstrate on the *diabetes* data, which contains several blood chemistry measures of glucose tolerance and insulin in 145 adults:

```
> library(glmxdiag)
> library(lars)
> data(diabetes)
> attach(diabetes)
```

There are a total of 20 variables on  $x$  (we print only the first eight) and a single variable on  $y$ :

```
> head(x)

      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]
[1,] 0.2738562 -0.0366722 0.8547269 0.9675242 1.4154898 0.5234059 0.5626882 1.11122333
[2,] 2.2448169 -0.5460300 0.2340651 -1.3350304 1.3130758 0.5212746 -0.6100346 -0.86139651
[3,] -0.1254230 -0.6068782 -0.8539217 -0.1487772 -0.6646828 0.6066164 0.1617207 -0.86272165
[4,] -0.5435734 1.1083583 -0.1042480 1.0165262 0.6999042 1.6550164 0.4899635 0.02338209
[5,] -1.4593984 -0.2744945 0.1119060 -0.8517877 0.3152839 1.0507493 1.3863575 0.28450104
[6,] 1.0632081 -0.7535232 -1.3825534 1.0762270 0.3700331 1.4987212 -0.3604525 -0.21421571

> head(y)

      [,1]
[1,] -1.2748860
[2,] 1.8434251
[3,] 0.4592363
[4,] 0.5640407
[5,] 1.8729633
[6,] 0.5275317
```

If we simply run `lars`, we will obtain a lasso regression:

```
> object <- lars(x,y)
> summary(object)
```

LARS/LASSO

Call: `lars(x = x, y = y)`

|   | Df | Rss    | Cp       |
|---|----|--------|----------|
| 0 | 1  | 852.48 | 812.2901 |
| 1 | 2  | 809.19 | 768.0660 |
| 2 | 3  | 543.04 | 485.8674 |
| 3 | 4  | 542.61 | 487.4075 |
| 4 | 5  | 478.70 | 421.1627 |
| 5 | 6  | 398.84 | 337.8830 |

We can obtain a plot of standardized coefficients (Figure 6.8), which provides us with the lasso coefficients, which, as the ratio of the L1 norm to least-squares coefficients ( $x$ -axis) approaches 1.0, the least-squares coefficients are obtained:

```
> plot(object)
```

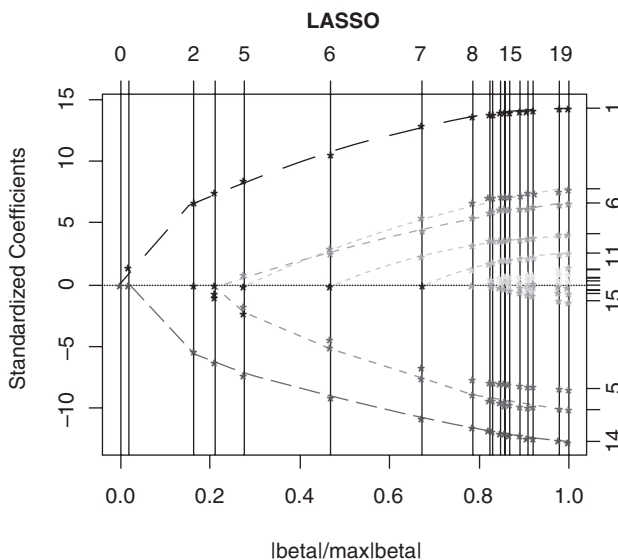


*Figure 6.8* Standardized coefficients as a function of the ratio of the L1 norm to least-squares. As this ratio nears 1.0 ( $\lambda = 0$ ), the least-squares coefficients are obtained as the L1 norm is the same as that of OLS. As  $\lambda$  increases, the L1 norm decreases and lasso drives coefficients to 0. The plot reveals when each coefficient is driven to 0. At the extreme left of the plot, there remains zero nonzero coefficients, as indicated by the top axis. From right to left then, the plot reveals the progression of lasso starting off with a least-squares solution (far right) to a full lasso (and trivial) “shrunk” solution in which all coefficients have been driven to 0 (far left, where  $|\beta|/\max|\beta|$  equals 0). At the far right of the plot, that ratio is equal to 1.0 since the  $\beta$  values are equivalent to their maximum OLS values.



We can obtain coefficients for each of the 20 measurements as follows:

```
> coef4.1 <- coef(object, s=4.1, mode="norm")
```

The code `mode = "norm"` designates the L1 norm (absolute value in this case; see the package for more options). We now print out the coefficients:

```
> coef4.1
[1] 0.2633462 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
[7] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
[13] 0.0000000 -0.1237479 0.0000000 0.0000000 0.0000000 0.0000000
[19] 0.0000000 0.0000000
```

All of the coefficients above are for the model for  $s = 4.1$  (i.e., just beyond the fourth value). If we would like coefficients at any particular value (L1 norm) for  $s$ , we can also use `predict.lars()`. For example, in the following we choose  $s = 3$  to see what the coefficients are at that particular value:

```
> coef3 <- predict(object, s=3, type="coef", mode="norm")
> coef3
$s
[1] 3

$coefficients
[1] 0.21054411 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[7] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[13] 0.00000000 -0.07358921 0.00000000 0.00000000 0.00000000 0.00000000
[19] 0.00000000 0.00000000
```

We can obtain a summary of the fit as follows:

```
> object
Call:
lars(x = x, y = y)
R-squared: 0.913
Sequence of LASSO moves:
Var 1 14 5 20 6 3 8 11 7 10 15 13 18 12 2 4 16 17 9 19
Step 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

We see that R-squared for the final model is equal to 0.913. The output also shows the steps where predictors were included in the model (variable 1 was included first, while variable 14 was included in the second step, and so on).

We will now proceed to fit the least-angle regression (Figure 6.9) by adding `type = "lar"`. Note that the only difference between the previous lasso and that of least-angle regression is the addition of the `lar` option:

```
> object2 <- lars(x, y, type="lar")
> object2
Call:
```

```

lars(x = x, y = y, type = "lar")
R-squared: 0.913
Sequence of LAR moves:
Var  1 14 5 20 6 3 8 11 7 10 15 13 18 12  2  4 16 17  9 19
Step 1 2 3  4 5 6 7  8 9 10 11 12 13 14 15 16 17 18 19 20

> plot(object2)

```

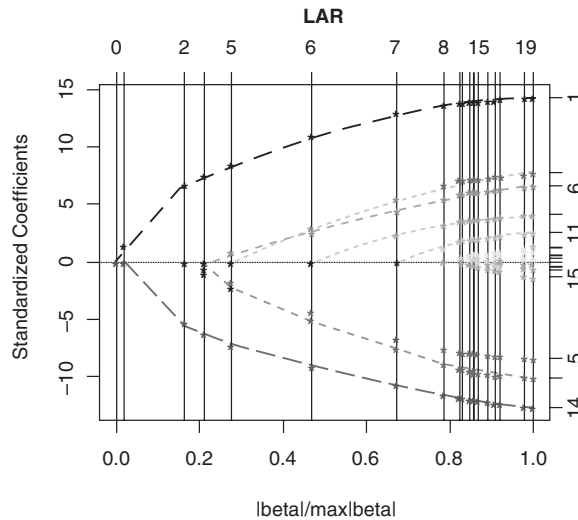


Figure 6.9 Least-angle regression (LAR) coefficients.

We note that the lasso and lar results agree perfectly. This is typical of lasso vs. lar results, where the coefficient profiles differ typically only when a coefficient curve crosses zero, which usually occurs under conditions of great correlation among predictors. For details and a discussion of this effect, see Hastie et al. (2009, pp. 75–76), who display a plot revealing identical lasso vs. lar profiles except for a single coefficient that crosses the  $y$ -axis = 0 (i.e., the coefficient path).

## 6.7 Additional Variable Selection Algorithms

Other selection algorithms similar in spirit to the ridge and lasso include the *Dantzig selector* as well as the *Garotte* (Breiman, 1995). But why use different penalties in the first place? That is, why not use the same selection algorithm in all situations regardless of the data? The answer to this question once more lies in the *No-Free-Lunch Theorem*, discussed earlier in the book, which, recall, holds that *no universal algorithm universally dominates all data sets*. That is, one data set may have specific “peculiarities” for which one algorithm capitalizes, while another data set may not (James et al., 2021). Hence, different regularization techniques may generate slightly different results depending on the peculiarities of the given data set and it is virtually impossible to be certain that one algorithm or method will dominate others on any particular data set. Just as no single treatment is guaranteed to “work” on any given individual, no single statistical or machine learning method is guaranteed to dominate all others on any single data set.

Still, the above admonition does not mean we cannot narrow things down to at least a “preferred” approach in “most” circumstances. According to Izenman (2008, p. 154), when comparing penalized regression approaches:

Extensive simulations comparing prediction accuracy under a wide variety of conditions and models ... show that the Lasso, which tends to choose too many variables, almost always contains the correct model; the nonnegative garotte tends to select smaller models than the Lasso, and all-possible subsets, while more likely to choose the correct model than the Lasso, selects models that contain the correct model far less often than the Lasso.

## 6.8 Chapter Review Exercises

1. Explain the concept of *regularization* with reference to simple linear regression and then multiple linear regression. Does regularizing a model always imply *simplifying* it? Why or why not?
2. At a technical level, how does ridge regression go about regularizing a model? What is the consequence of this adjustment?
3. Revisit the *Hald* data in R and conduct a ridge regression using only predictors  $x_1$  and  $x_2$ . How does your output compare to using all four predictors?
4. What is the difference between a *ridge estimator* and that of a *lasso estimator*?
5. Generate some random data in Python and conduct a lasso regression. Set up your model so that it has ten predictors and a single response. Then, obtain a prediction on the response for all values of predictors equal to 0.
6. Compare the penalties for *ridge regression*, *lasso*, and *elastic net*. Why is elastic net considered to be a “compromise” between ridge and lasso? Explain.
7. Under what circumstance is least-angle regression different from lasso regression? When will they generate the same output? Use the example featured in the chapter as a starting point to your discussion.
8. Generally, which model will be easier to interpret, that of a *ridge regression*, or that of *ordinary least-squares*? Why?
9. Suppose you conduct both a least-squares regression and a ridge regression. You are asked by a reviewer of your publication which model is “most correct.” How would you go about answering the reviewer?
10. Research the *garotte* algorithm online and briefly discuss how it is different from ridge and the lasso.

## 7 Nonlinear and Nonparametric Regression

The regression methods surveyed thus far in this book have all featured one common assumption: that of the relationship between response variables and explanatory variables being modeled is *linearly related*. Why is linearity the focus of so many statistical methods? Huge books, some well over a thousand pages, are written primarily on linear methods alone. Why the obsession with linearity? We see two primary reasons for this focus on linearity:

- 1) *Many phenomena in the sciences can be well approximated by linear methods.* The true function underlying the data may not be (and in fact, is *usually* not) linear, but linear models allow us to *approximate* those unknown functions rather well. In short, linearity “works” in many sciences because, though specifying the exact functional form underlying data is difficult to impossible, linearity is a useful *assumption* to make and makes for a nice *approximation* of scientific reality. The key words here are *assumption* and *approximation*. Very few data, if any, actually fall on a straight line.
- 2) A second reason why linearity is popular is because it is *simple* and *convenient*. In the absence of knowing the true functional form, linearity provides reasonable prediction and generality analogous in some sense to why we use means and averages to describe many phenomena, even though such can be grossly misleading at times. Even in its sister discipline, *linear algebra*, linearity is, not surprisingly given its name, assumed in most applications. The convenience of fitting simple, functional forms comes at a price, however. Rarely, if ever, do those functional forms represent empirical *reality*. Reality is usually much more complex, or at least in many cases, we *assume* it is more complex. However, assuming a linear relationship generally guards against overfitting and hence can be more generalizable than if we try too much to model the exact form of the relationship that may underlie the data.

*Though linear models dominate statistics and science, many, if not most phenomena in science are nonlinear. Seeking alternatives to linearity is often preferred to better capture the reality of scientific processes. Advancements in statistics and machine learning are often geared toward models and algorithms that help better unpack nonlinear features in data. Efforts to detect complex nonlinear patterns are at the heart of cutting-edge statistics and machine learning.*

The truth of the matter is that most if not all relationships are *nonlinear* and may be even more complex than we could even ever model. Having surveyed the linear ground thus far in this book, our goal in this chapter is to survey a few methods that model nonlinear relationships. There are

three general approaches that we will consider here that seek to capture nonlinearity in data: 1) *polynomial regression*, 2) *regression splines*, and 3) *kernel regression*, which we will first motivate by simple local-averaging.

## 7.1 Polynomial Regression

Recall the classic linear model,

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad (7.1)$$

The above model is actually a special case of the more general *polynomial regression model*,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \cdots + \beta_p x_i^p + \varepsilon_i \quad (7.2)$$

A polynomial of order  $p$  can have  $p - 1$  “bends.” For instance, a straight line of order “1” has  $p - 1 = 1 - 1 = 0$  bends, which is what makes it a straight line (i.e., there are no “bends” in a straight line, at least macroscopically). A parabola has a single bend (i.e.,  $2 - 1 = 1$ ), and so on. A constant-only model (e.g., the mean) implies  $p = 0$ .

Typically, polynomials of only degrees 1 through 4 are generally used in scientific practice, since the higher the order of the polynomial, the greater the possibility of *overfitting*. That is, while fitting a polynomial of say, degree 10 might provide an exceptional fit to your training data (you may in fact be connecting the dots if your sample size is rather small!), many of those higher degrees are likely simply capitalizing on chance factors or random noise in the training data and hence the fit will not generalize well when validated on test data. That is, a polynomial of excessive order is likely to overfit the data. Recall that when it comes to model validation, it is how the model fits on *test* data that matters most, not how it fits on *training* data.

*A polynomial of order (or “degree”)  $p$  can have  $p - 1$  “bends.” The complexity of a polynomial increases with the number of bends in it. Typically, polynomials of only up to degree 4 are used in scientific practice, as higher-order polynomials are likely to overfit the training data. However, higher-order polynomials may be useful depending on the nature of the scientific material and domain.*

When we are fitting the classic linear model, we are in actuality performing polynomial regression, only that the polynomial is a rather simple one, that of a *line*. That is, the linear equation is simply a special case of the wider polynomial equation where only powers of one are included:

$$\begin{aligned} y_i &= \{\beta_0 + \beta_1 x_i\} + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \cdots + \beta_p x_i^p + \varepsilon_i \\ &\quad \downarrow \\ y_i &= \{\beta_0 + \beta_1 x_i\} + \varepsilon_i \end{aligned}$$

Another way of thinking of the above is that the linear model is a polynomial model in which all higher powers beyond unity are constrained or set to zero.

### 7.1.1 Polynomial Regression in Python

We can fit a basic polynomial regression in Python quite easily using `numpy.polyfit()`:

```
import numpy as np
import matplotlib.pyplot as plt
```

We would like to fit a polynomial to the following data:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

We specify a third-degree polynomial (Figure 7.1):

```
mymodel = np.poly1d(np.polyfit(x, y, 3))
myline = np.linspace(1, 22, 100)

plt.scatter(x, y)
Out[209]: <matplotlib.collections.PathCollection at 0x1af74f20dd0>

plt.plot(myline, mymodel(myline))
Out[210]: [<matplotlib.lines.Line2D at 0x1af77d64cd0>]

plt.show()
```

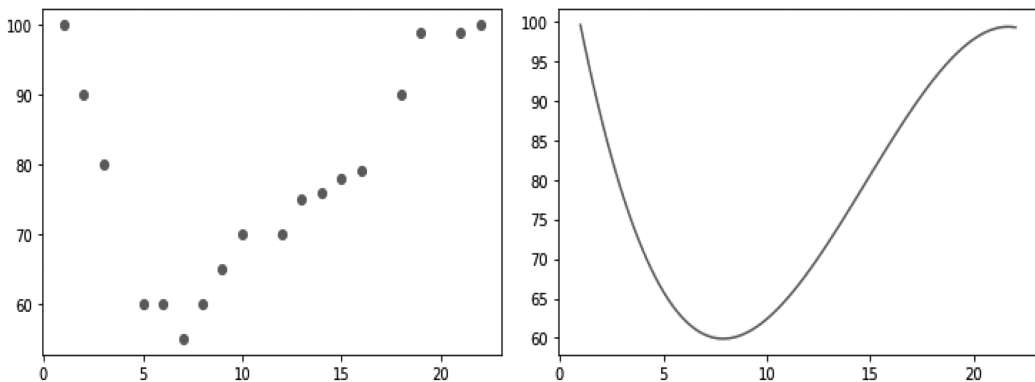


Figure 7.1 Third-degree polynomial fit to curvilinear data.

```
import numpy
from sklearn.metrics import r2_score
print(r2_score(y, mymodel(x)))
0.9432150416451026
```

We can see from the R-squared that the proportion of variance explained by the polynomial is approximately 94%. The cubic model (i.e., degree 3) fits better than had we tried a quadratic one:

```
mymodel = np.poly1d(np.polyfit(x, y, 2))
print(r2_score(y, mymodel(x)))
0.759777160189589
```

Had we fit a model of degree 4 (i.e., quartic), we would have obtained only a slightly larger R-squared than the cubic, which would have probably resulted in overfitting the data:

```
mymodel = np.poly1d(np.polyfit(x, y, 4))
print(r2_score(y, mymodel(x)))
0.9542030834699506
```

Polynomial regression can also be performed in Python using `statsmodels`.

### 7.1.2 Polynomial Regression in R

We illustrate polynomial regression on the Boston data set. This is data derived from information collected by the U.S. Census Service on housing characteristics and measures in the area of Boston, MA. We will use the *tidyverse* (Wickham et al., 2019) and *caret* (Kuhn, 2008) packages. A version of polynomial regression using this data can also be found in James et al. (2013, p. 115):

```
> library(tidyverse)
> library(caret)
> data("Boston", package = "MASS")
```

There are a total of 506 observations in the Boston data. We will now split the data 0.8 into train and 0.2 into test:

```
> set.seed(123)
> training.samples <- Boston$medv %>%
+   createDataPartition(p = 0.8, list = FALSE)
> train.data <- Boston[training.samples, ]
> test.data <- Boston[-training.samples, ]
```

We confirm that the data has been split correctly:

```
> nrow(train.data)
[1] 407
> nrow(test.data)
[1] 99
```

The number of rows in the train data is 407 while the number of rows in the test data is 99. Recall we had requested that the train data be 80% of the entire data. The total number of cases is equal to  $407 + 99 = 506$ , of which 407 is 80%.

We will now build the model, a polynomial model of degree 5 on the train data:

```
> model <- lm(medv ~ poly(lstat, 5, raw = TRUE),
+ data = train.data)
> summary(model)
```

```

Call:
lm(formula = medv ~ poly(lstat, 5, raw = TRUE), data = train.data)

Residuals:

Min          1Q      Median          3Q          Max
-13.1519   -3.1235   -0.5927    2.0962   27.1286

Coefficients:

                Estimate Std. Error t value Pr(>|t|)
(Intercept)      6.765e+01  4.273e+00  15.831 < 2e-16 ***
poly(lstat, 5, raw = TRUE)1 -1.177e+01  1.786e+00  -6.588 1.40e-10 ***
poly(lstat, 5, raw = TRUE)2  1.220e+00  2.580e-01   4.727 3.16e-06 ***
poly(lstat, 5, raw = TRUE)3 -6.385e-02  1.644e-02  -3.884 0.000120 ***
poly(lstat, 5, raw = TRUE)4  1.577e-03  4.714e-04   3.345 0.000901 ***
poly(lstat, 5, raw = TRUE)5 -1.459e-05  4.954e-06  -2.945 0.003421 **

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.205 on 401 degrees of freedom
Multiple R-squared:  0.6816, Adjusted R-squared:  0.6777
F-statistic: 171.7 on 5 and 401 DF, p-value: < 2.2e-16

```

We can see that all degrees of the polynomial are statistically significant, most importantly the highest degree in this case, which is the fifth degree ( $p = 0.003421$ ). The model explains approximately 68% of the variance in the response (i.e., Multiple R-squared). Adjusted R-squared is only slightly less at 0.6777.

We now obtain some predictions based on the test data:

```

> predictions <- model %>% predict(test.data)
> head(predictions)
      3      6      9     11     14     15
36.25964 31.52683 10.98130 14.57053 24.46163 22.15775

```

We can get an R-squared on the predictions as follows, which we can see is just a bit higher than the explained variance on the train data:

```

> R_squared = R2(predictions, test.data$medv)
> R_squared
[1] 0.6829474

```

We can also test smaller-degree models, such as simply a quadratic model. We can obtain coefficients for this model as follows:

```

> print(lm(medv ~ lstat + I(lstat^2), data = train.data))

Call:
lm(formula = medv ~ lstat + I(lstat^2), data = train.data)

```



```

Coefficients:
(Intercept)  lstat  I(lstat^2)
  42.5736  -2.2673    0.0412

```

The package *polyreg* (Matloff et al., 2022) can also be used to fit polynomial regression models.

### 7.1.3 Polynomial Regression as a Global Strategy

What is the problem with polynomial regression? There is no problem, really, if one wishes to model the entire range of  $x_i$  by assuming a common functional form. That is, both the classic linear model and polynomial regression fit the data in a very *global* manner, similar in some sense to how the arithmetic mean summarizes a set of data from one end of the distribution to the other. For example, consider the small data set

2, 5, 8, 10, 15, 16, 170, 171, 172

The mean of this data is 63.2. But how well does the mean represent the data? Not that well, since, for example, the data points 170, 171, and 172 are very much greater than the rest of the data and the mean will thus be pulled *upward* because of these more extreme scores:

2, 5, 8, 10, 15, 16, **170, 171, 172**

Computing the mean on the above set of data provides a very “global” summary of the data. We may not wish to model the data in this way, whether it is in computing the mean or conducting a regression, and may prefer a more *local* approach.

### 7.1.4 A More Local Alternative

For the above data, we may be better off calculating two averages, one for the data 2, 5, 8, 10, 15, 16, and one for the data 170, 171, 172. In doing so, we are essentially modeling the data at a more *local* level. Though this example is very simple, it demonstrates a strong but obvious point, and that is, *fitting global measures to data provides a global summary of that data*. If the data does not conform to a global representation, then fitting *subsets* of the data more locally may be preferable. Indeed, at its extreme, the most local we could go is to model each data point *individually*, but such would be foolish and detract from the purpose of modeling in the first place, except for extremely small data sets. Exact “curve-fitting” is essentially “connecting the dots,” which, for most purposes, defeats the goal of statistical modeling. In this case, we are not averaging anything; we are simply connecting observations on a descriptive level. The general goal of modeling is to estimate a general function over the data, even if local. The goal is not simply “connecting the dots” on the training data and achieving perfect fit. *Anything less than “connecting the dots,” in most cases, can be considered a very general universal form of the aforementioned and discussed regularization concept as it concerns generalizability and minimizing test error.*

### 7.1.5 Least-Squares Regression Line as a “Floating Mean”: Toward a Localized Approach

To further motivate localized regression approaches, recall that the classic least-squares regression line can be interpreted as a “floating mean” across the range of values of the predictor,  $x_i$ . That is, the slope  $\beta_1$  in  $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$ , analogous to the mean, is a kind of “summary statistic” because

the model assumes it is constant across the range of  $x_i$ . Even if the data suggested a different slope or functional form within a given range of  $x_i$ , the classic least-squares line would not account for this. Likewise for polynomial regression, in which the polynomial is fit across the range of  $x_i$ , essentially ignoring more “local behavior” within intervals of  $x_i$ . Both linear and polynomial regression *generalize* over  $x_i$  to provide a summary of the data in terms of model fit.

What if we wished, however, to consider *local variation* more carefully? Perhaps the easiest example is that of a scatterplot that you may have come across in introductory statistics, one featuring the presence of two “groups” within the plot, as in Figure 7.2.

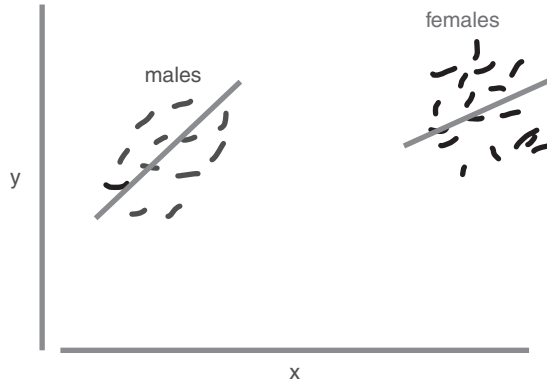


Figure 7.2 Basic scatterplot on two groups (males vs. females) suggesting a more localized approach to modeling, possibly by fitting an individual slope to each group.

Of course, for the data in Figure 7.2, a more localized approach to the regression could be as simple as modeling each group separately, with one slope for males and one slope for females. The locality of the data, in this case, is the *group*. But consider this example carefully and you will see that what we are actually proposing here, at least implicitly, is to recognize the potential for an *interaction* in the plot and not simply a main effect. That is, *spotting the potential of an interaction in this sense is an effort to provide a more localized representation of the data*.

*A localized representation of data can be as simple as computing different averages by identifying different sets in data, or by modeling interaction terms in a regression (or ANOVA) context.*

#### 7.1.5.1 Zooming in on Locality

What if we take locality to the limit, to the extreme? As mentioned, to do so would be to narrow in on each individual data point in our plot, or, for the example of averages, to simply consider each data point by itself. It doesn’t get much more “local” than considering the case study without producing an average at all!

In the regression context, we could produce interaction terms with increasingly greater degrees of “split” or factor levels in ANOVA in one or more variables. Hence, instead of dividing the

data into two groups, we could divide them into four or five, for instance. The general principle then in a localized approach to regression (or ANOVA) is that instead of assuming a general functional form across the range of  $x_i$ , we wish instead to “zoom in” and inspect *local variations* within given intervals of  $x_i$ . This is where localized methods of regression come into play. However, as we survey a few approaches to accomplish this, the reader should take away what we have discussed here: that zooming in locally on data can be as simple as splitting data in different ways or modeling interaction terms in regression or ANOVA. That is, local examination and modeling of data need not necessarily take the form of a sophisticated or advanced modeling strategy.

To understand and motivate local regression, it is imperative to first understand the nature of a *basis function* as it relates to regression and modeling in general. Basis functions are so general that they also open the door to understanding more advanced techniques such as neural networks as well, as there are intimate connections between basis functions and most other statistical and machine learning methods.

## 7.2 Basis Functions and Expansions

A *basis function*, or *linear basis expansion* (Hastie et al., 2009) is essentially the taking of a function of variable (or matrix, in this case)  $\mathbf{X}$ , which we can express quite generally,

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \beta_4 b_4(x_i) + \cdots + \beta_p b_p(x_i) + \varepsilon_i \quad (7.3)$$

Note that the above is not polynomial regression; none of the  $(x_i)$  are raised to exponents other than unity. The actual basis functions, which are functions of  $x_i$ , are  $b_1$  through  $b_p$ . For example,  $\beta_1 b_1(x_i)$  is the same as we would classically have in least-squares in that we are weighting  $x_i$  by the regression parameter  $\beta_1$ , but to be more precise, what is being weighted is not  $x_i$ ; we are instead weighting a *function* of  $x_i$ , and that function is  $b_1(x_i)$ . That is, we first weight  $x_i$  by  $b_1$ , then weight that product by  $\beta_1$ . The basis function here is  $b_1(x_i)$ .

To understand the above more clearly, consider the classic linear model as a special case of where basis functions occur. What basis function is inherent or “natural” in the classic linear model? To see this, we observe

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \beta_4 b_4(x_i) + \cdots + \beta_p b_p(x_i) + \varepsilon_i$$

$$y_i = \beta_0 + \beta_1 b_1(x_i)$$

$$y_i = \beta_0 + \beta_1 (1)(x_i)$$

$$y_i = \beta_0 + \beta_1(x_i)$$

and thus we end up with the classic one-predictor linear model. The basis function,  $b_1(x_i)$  is simply  $x_i$  with  $b_1$  implied or set to 1.

Let’s now consider a primitive though arbitrary basis function simply to extend on this simple definition. Instead of  $b_1(x_i) = (1)(x_i)$ , suppose the basis  $b_1$  were equal to 2, making  $b_1(x_i) = (2)(x_i)$ . Note that none of this is related to the slope  $\beta_1$ ; *the basis function here is simply an “action” directly on  $x_i$  before the partial regression slope is applied to it*. The question for us then, is what should the basis function comprise of? There are many possibilities, and the choice of a basis function will be motivated by the type of model we are considering. It doesn’t take long to realize that one could quite easily write an entire book titled *Basis Functions for Prediction*, which would encompass numerous possibilities and statistical models, which would include simple and multiple

linear regression, polynomial regression, step functions, regression splines, analysis of variance, and a whole gamut of others. The common thread in all of these models is that they are *identified by their bases*, that is, their transformations of  $x_i$  *before* weighting by least-squares or other technique; that is, before weighting by the parameter estimates in the given model.

### 7.2.1 *Basis Functions and Locality*

Among the chief benefits of basis functions is that they allow us to model the data at a more *local level*, much more local than polynomial regression ever could. The use of basis functions also allows us to model the relationship between  $y$  and  $x$  nonlinearly by adopting nonlinear basis functions. Because the basis function occurs *inside* of the expression “ $\beta_1 b_1(x_i)$ ”, we could, in theory, allow  $b_1$  to be a whole host of nonlinear functions (Bishop, 2006), which include *Gaussian basis functions* such as the following,

$$\phi_j(x) = \exp \left\{ \frac{(x - \mu_j)^2}{2s^2} \right\} \quad (7.4)$$

or a *sigmoidal basis function*,

$$\phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \quad (7.5)$$

among others, where  $s$  is the standard deviation in the aforementioned functions. The basis functions we have surveyed thus far have been all *fixed* beforehand; that is, they are not amenable or adaptable to the data. The curse of dimensionality will also limit the utility of fixed basis functions (Bishop, 2006). It will thus be advantageous to consider basis functions that are not fixed. We will encounter such basis functions, that is, those that are *adaptive* or *recursive*, when we survey neural networks later in the book and emphasize the connection between the regression methods of this chapter and neural network approaches.

*A basis function, or linear basis expansion, weights the predictors in a regression model. Depending on the basis function chosen, this may allow for a more “local” examination of the fit at that particular point or region in the range of predictor values, or, may be limited to a more global representation of the data. In regression, this weighting occurs before weighting by the relevant partial regression slope. The basis functions employed in this chapter are fixed and hence can be quite limiting. Neural network basis functions are generally more adaptive to the data.*

### 7.2.2 *Neural Networks as a Basis Expansion: Generalizing the Concept*

We mentioned above that we could title an entire book about basis functions or more precisely, *basis expansions*. We emphasized some of the more popular “special cases” of what we would include in such a book, namely simple and multiple regression, and so on. This was meant to

emphasize how general basis expansions actually are and how many statistical methods can be understood or conceived in terms of basis expansions. Indeed, neural networks can be considered as a basis expansion as well and yet another special case of this wider framework.

In a classic neural network, as we will see later in the book, we take *functions of functions*, otherwise known as *composite functions* from elementary mathematics. Taking composite functions allows us to implement a similar idea of a *basis expansion* as we did just earlier. The general concept we need is that of a *feature extractor* (Murray, 2022):

$$f(x; \theta) = \mathbf{W}\phi(x) + a \quad (7.6)$$

The parameters in the above are  $\theta = (\mathbf{W}, a)$  and if we were to write the classic functional equation, we would omit  $\phi$  altogether. The key to the above, however, is the inclusion of  $\phi$  which in our earlier regression example was  $\beta_1 b_1(x_i)$  where  $\phi = b_1$ . That is,

$$\begin{aligned} f(x; \theta) &= \mathbf{W}\phi(x) + a \\ &= \mathbf{W}b_1(x) + a \end{aligned}$$

where we now denote  $b_1$  as  $b_i$  to eliminate any confusion in notation. What is the benefit to writing  $f(x; \theta)$  this way? The answer is its *generality*. The more general form for  $f(x; \theta)$  allows  $\phi$  to be a more *arbitrary* and *general* function. More to the point, we are now permitting the feature extractor to have its own designated parameter,  $\theta_2$ :

$$f(x; \theta) = \mathbf{W}\phi(x; \theta_2) + a \quad (7.7)$$

As we will see when we study neural networks, this allowance of the *feature extractor* to have its own parameter vector will generalize our current idea of basis expansion to much wider possibilities. The general idea is to specify *functions of functions*, which is what neural networks are all about.

We can see then that basis functions are an entirely *general concept*, and the choice of a basis function is essentially theoretically limitless. In the earliest of courses in introductory statistics, basis functions are indeed present in *t*-tests, ANOVA, regression, and so on; however they are simply not revealed and are usually *implicit* within the given method or procedure. Interactions in ANOVA, for instance, are yet another example of a basis expansion. *Basis expansions essentially enrich the feature space*. Support vector machines are yet another example of a basis expansion, which we survey later in the book.

In what follows and in returning to our discussion of nonlinearity, we focus on a particularly common basis function or expansion that will prove useful for modeling nonlinear regression, that of *regression splines*.

### 7.2.3 Regression Splines and the Concept of a “Knot”

We have discussed so far the benefit of leaving global functions behind and fitting more local functions to data. The general idea is to fit a model locally, or “zoomed in” to different parts of the data. *Regression splines* are a common way to get at the local structure of the data in a regression context.

But how can we define local subsets of the data range? An easy and common way to do so is via *knots*. A *knot* is something that divides areas of the data on  $x_i$  into intervals, where the coefficients in the model *change* as to reflect a change in the data trend. In general, *the greater the number*

*knots, the greater the flexibility of the curve.* As the number of knots increases without bound to infinity, we eventually obtain perfect fit. If we use too many knots, however, we are at risk of overfitting. The selection of the right number is thus crucial and is typically made by cross-validation or possibly simply by visualization. *Smoothing splines*, on the other hand, often use a maximal or near-maximal number of knots instead of that number being arbitrarily selected by the user (Hastie et al., 2009).

Figure 7.3 depicts the concept of a knot in spline regression, where it is clear that the coefficient of the curve changes in a rather dramatic and somewhat exaggerated way to illustrate the concept.

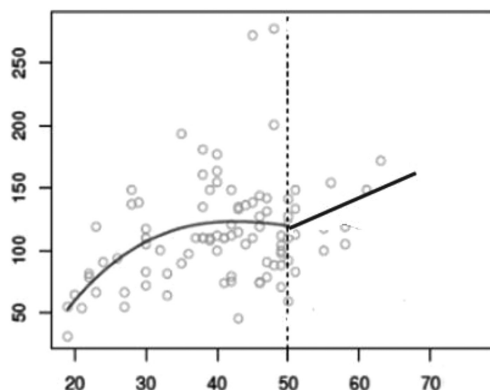


*Figure 7.3* Concept of a “knot” in spline regression. The location of the change in the coefficient of the curve is the location of the knot. Such is an exaggerated illustration, but clearly indicates that a knot would be useful since the trend in the data suggests a different function from 50 onward.

Recall that in the previous chapter we had chosen the tuning parameter in ridge, lasso, and elastic net by cross-validation to implement regularization. The choice of the number of knots can also be considered a form of regularization, and thus the number of knots can be thought of as a tuning parameter. That is, *by reducing the number of knots, thus reducing the chance of overfitting, we are essentially regularizing the model.* Indeed, this whole discussion of splines can actually be formally recast into the wider category of regularization (see Hastie et al., 2009, pp. 167–168 for a discussion). It is generally recommended that using 4 to 5 knots equally distributed, say at percentiles 20, 40, 60, etc., is a useful first strategy for their implementation in a typical regression problem (Fox, 2016).

*Knots in regression splines define local subsets of a data range where coefficients typically change. The choice of the number of knots in spline regression can yet again be considered an instance of regularizing a model, just as selecting a suitable tuning parameter in ridge and lasso regression was a form of regularization. Hence, the number of knots can be thought of as a tuning parameter.*

### 7.2.4 Conceptualizing Regression Splines

How should we conceptualize the idea of a regression spline? Regression splines are best represented as a special case of the more general idea of basis functions or basis expansions that we just surveyed. Recall the basis representation developed earlier:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \beta_4 b_4(x_i) + \cdots + \beta_p b_p(x_i) + \varepsilon_i$$

where  $b_1$  through  $b_p$  are the basis functions. That is, they are transformations of  $x_i$  *before* the relevant weighting by  $\beta_1$  through  $\beta_p$ . The placement of the knots can, in theory at least, be arbitrarily chosen based on shifts in the function. However, as mentioned, a uniform, equally spaced distribution of knots is usually the more common and initial strategy (James et al., 2013).

Cross-validation can be used to determine the best number of knots, that which produces the smallest residual sums of squares in the test error. *Cubic splines* are often preferred because these naturally ensure the function is continuous and has continuous first and second derivatives at each knot (Murphy, 2022). The number of knots divides the data into one or more *bins*. For example, 4 knots divide the data into  $4 + 1$  bins. If we increased the number of knots to their limit, we would have the number of knots equal to the number of values for  $x_i$  or data points. Using  $n$  knots to represent  $x_i$  *unique* values results in what is generally known as a *smoothing spline*. Since there is typically not a fixed number of knots determined *a priori*, smoothing splines are usually regarded as *nonparametric*, as the number of splines is a function of the number of data points or values of  $x_i$  in the data (Murphy, 2022, p. 395).

As noted by Fox (2016), spline regression models can also be considered a “bridge” between the *linear* and *generalized linear model*. In fact, the *generalized additive model* is one that extends the regression spline to the case of multidimensional inputs, assuming an additive form (i.e., no interactions). The generalized additive model can then be extended beyond regression and into classification, in which case one has the full-blown generalized linear model (Murphy, 2022).

### 7.2.5 Problem with Splines and Imposing Constraints

The great thing about splines is that we can be as flexible as we wish. However, too much flexibility entails a cost. For example, consider Figure 7.4. In this case, we are *too* flexible. That is, *we have not imposed enough constraints on the knots*. To remedy this, we begin imposing constraints, such as continuity from left to right to ensure there cannot be a jump (James et al., 2013). Recall what “imposing a constraint” means; it means we would like the curve to fit, but without, in this case of continuity, a *jump*. That is, we have taken the “jump” out of the possibility in the fitting or estimation of the curve; this is the *constraint* we are imposing.

This idea of imposing constraints on functions is not at all new. Indeed, virtually every statistical model has constraints either implicitly imposed that we cannot see or that the user imposes manually, such as in structural equation modeling where constraints are regularly imposed. As perhaps the simplest of examples, in linear regression the constraint of *linearity* is imposed, meaning that we will fit a functional form to data *with the constraint that it assumes only a linear form*. While this is an obvious example, it is nonetheless powerful and pedagogical; many researchers perform linear regression analysis without the awareness that they have implicitly constrained their predictive *potential* to that of lines at the expense of other curves such as a higher-order polynomial.



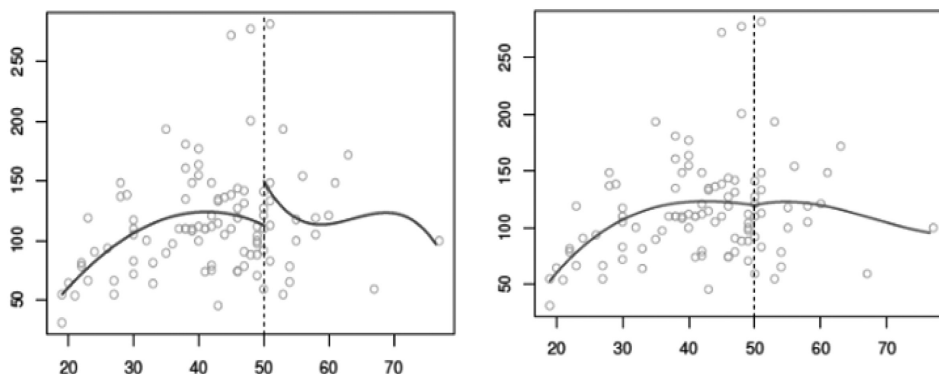


Figure 7.4 Imposing the constraint of continuity (right) on the cubic regression spline.

Constraining a spline regression to have no jumps, or other constraints, gives the model less flexibility as when such parameters need to be estimated, in addition to limiting the type of spline fit so that it is scientifically meaningful to the researcher. Hence, if it is unreasonable to assume a jump can occur in one's data, then constraining there to be no jumps (or other characteristic of the fitted curve) is perfectly reasonable. On the other hand, if such a jump is a reasonable possibility in the data, then imposing such a constraint would be ill-advised and the data should be modeled in some capacity that incorporates the discontinuity. In this regard, the *science* matters a great deal. If a discontinuity is a reasonable expectation for the given scientific data, it may make little sense to constrain it.

### 7.2.6 Polynomial Regression vs. Regression Splines

In our discussion above, we have presented regression splines as a special case of the more general idea of basis functions, and noted they are more local than methods such as polynomial regression. Which provides the better fit? As argued by James et al. (2013), regression splines often give superior results. Why? The reason that splines often give better results is that each polynomial regression and splines introduce flexibility in a different manner. *Splines typically do it better.* For polynomial regression, flexibility is introduced by increasing the degree of the function (e.g., from linear to quadratic to cubic and so on). Splines, on the other hand, introduce flexibility by keeping the degree fixed and instead increasing the number of knots, which can range from minimal to maximal. Hence, splines are simply more flexible than fitting a global polynomial, even of increasing degree of complexity, which usually means splines will perform better in cross-validation. What is more, as mentioned, regression splines can serve as a bridge to more generalized modeling such as the generalized linear model (Fox, 2016).

One-dimensional splines can also be generalized to multidimensional splines though these are beyond the scope of this book. Multidimensional splines are a straight generalization of the one-dimensional spline, however (Hastie et al., 2009). For an excellent overview of spline options across a variety of R packages, see Perperoglou et al. (2019). We can also use the *spline2* package (Wang, 2024) to fit, in addition to natural cubic splines, other splines as well, such as more general *B-splines*, *M-splines*, among others, which we will not detail here (see the package for details). Another popular package for fitting splines in R is that of *gss* (Gu, 2014). One can also use the *bspline* package (Sokol, 2024) for estimating spline regression. The *absorber* package (Savino, 2024) is a variable-selection method using b-splines in multivariate nonparametric regression models that can be useful in choosing the “best” variables in a spline regression problem.



### 7.3 Nonparametric Regression: Local and Kernel Regression

In our quest to model nonlinearity, we have thus far surveyed two possibilities: polynomial regression and regression splines. Polynomial regression provided a global fit to the data, while splines allowed us to model the data at a more local level. However, both approaches were *parametric* in nature, though splines, depending on the type and how we are using the terms parametric vs. nonparametric, are also often considered nonparametric (Fox, 2016, p. 511).

We now briefly survey two nonparametric approaches: 1) *simple local-averaging*, and 2) *kernel regression*. We will see that kernel regression is a generalization of simple local-averaging and the latter can be used to motivate the former. Kernel regression is our primary focus as a nonparametric technique.

To be more precise, kernel regression is a way to model a *non-hypothesized* form of the relationship, which will typically end up being some undesigned nonlinear form. It can be considered a *nonparametric* approach to regression (Fox, 2016). The nonparametric feature of kernel regression, as is true of kernel methods in general, allows the data to “speak” and tell us of its shape; we do not in advance assume a hypothetical linear or nonlinear form. That is, if the parametric linear model is

$$y = f(x, \theta) + \varepsilon \quad (7.8)$$

then the nonparametric linear model is

$$\begin{aligned} y &= f(x) + \varepsilon \\ &= f(x_1, x_2, \dots, x_p) + \varepsilon \end{aligned} \quad (7.9)$$

where now there is no parameter vector,  $\theta$  and instead the form of the relation is learned entirely by the data,  $x_1, x_2, \dots, x_p$ . *Simple local-averaging* on the other hand, is what is generally regarded as a “naïve” approach to regression where averages within predetermined subsets or bins along the  $x$ -axis are established in order to obtain a more *local flavor* in tracing the response variable. Kernel regression can be interpreted as a more general method of simple local-averaging.

#### 7.3.1 Motivating Kernel Regression via Local-Averaging

In classical least-squares regression, recall that we typically assume that the conditional distribution of  $y/x$  is Gaussian (i.e., normal). More formally, we model  $p(y, x|D)$  where  $y$  is the response,  $x$  is the predictor, and  $D$  is the data. That is, in typical classical regression, we are tracing the mean function:

$$\mu = E(y | x_1, \dots, x_p) = \alpha + \beta_1 x_1 + \dots + \beta_p x_p \quad (7.10)$$

Given the assumption that the conditional distribution is normal, consider if we backtracked for a moment and motivated ordinary regression via a naïve approach and dissected the explanatory variable into *bins*, that is, *subsets* of area or *intervals* along the  $x$ -axis. Not surprisingly, bias is minimized by making the bins as numerous and as narrow as possible (Fox, 2016). Sampling variance is minimized by using a small number of relatively wide bins, each with many observations. Hence, as usual, there is a trade-off between bias and variance here, encapsulating the classic *bias-variance trade-off*, as illustrated in Figure 7.5.

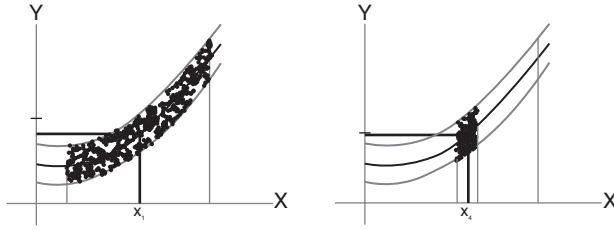


Figure 7.5 Local-averaging in regression. Bias is minimized by making the bins more numerous and narrow; wide bins (left), narrow bins (right).

We note in Figure 7.5 that as the bins get progressively narrower, we eventually “zero in” on the actual data points that make up the theoretical regression line as well as the actual points that happen to fall on the sample regression line. Hence, in this extreme version of nonparametric regression where each data point is a mean, we are simply *connecting the means* located within the bins. Clearly, such an approach would usually not be useful, and we are better off increasing the size of each bin. However, if we increase the bin size too much, we miss out on the *locality* of the data we initially were after; *this is the nature of the trade-off*.

What makes this approach *nonparametric* is that no assumptions about the population or form of the relationship are made. Sample size plays a crucial role here as well. As the sample size increases, the intervals become successively narrower, each containing more data, where in the limit as sample size gets larger, the naïve nonparametric regression coincides with the population regression (Fox, 2016). What happens with this approach when there are many explanatory variables? Not surprisingly, the previously discussed *curse of dimensionality* rears its ugly head. For simplicity, we focus on the case where we only have a single predictor.

A glaring weakness of this approach is that for a bin of a given size, assuming the bin is larger than each single data point, the estimate within each bin will be inexact for surrounding data points. That is, if the bin size contains 20 data points, the mean computed on these 20 data points will be a rather *coarse* representation of locality. Is there a way we can do better than the mean? Is there a way we can represent the data within each interval more “fairly” or precisely? The answer to this will be yes, via *kernel regression*.

### 7.3.2 Kernel Regression – “Locally Weighted Averaging”

Kernel regression seeks to improve on simple local-averaging by attempting to explicitly give more weight to observations that are close to the focal  $x_0$  and less weight to more remote values in the training data (Fox, 2016). What is a “focal  $x_0$ ”? This is the hypothetical value from which you wish to make a prediction of the response,  $y$ . This is an attractive idea, as doing so will provide a better representation of the data *within* each interval or “bin” of data. The distance  $x_i$  from  $x_0$  can be represented by (Fox, 2016, p. 529):

$$z_i = (x_i - x_0) / h \quad (7.11)$$

The scale factor here is  $h$  and is the *bandwidth* of the kernel estimator, controlling its degree of smoothness. As an example, suppose the distance  $(x_i - x_0)$  is equal to 5, with a bandwidth  $h$  equal to 1, yielding a  $z_i$  equal to 5. If we increase the bandwidth to 2, then  $z_i$  is equal to 2.5. The function of  $h$  then is to *scale* the distance  $(x_i - x_0)$ . For a constant distance, as  $h$  grows larger and larger, the interval on the  $x$ -axis becomes smaller relative to  $h$  and we obtain a *finer* fit. The

smoothness of the fitted function is thus largely influenced by the size of the bandwidth,  $h$ . In general, as  $h$  increases, the function becomes smoother. Too fine, or too smooth a fit could be detrimental in modeling scientific data, especially if the finer or smoother fit provides too “general” of a modeling approach across the span of the data where perhaps one strategy is more appropriate than another for particular ranges of the data space.

Scaling deviations  $x_i - x_0$  by the bandwidth  $h$  is fine, but we can do even better and apply a kernel function to the scaled deviation to obtain final weights,  $w_i$ :

$$w_i = K[(x_i - x_0) / h] \quad (7.12)$$

Using the above weights  $w_i$ , the fitted value at  $x_0$  is computed by weighted local-averaging of the  $y_i$  values:

$$\hat{f}(x_0) = \hat{y} | x_0 = \frac{\sum_{i=1}^n w_i y_i}{\sum_{i=1}^n w_i} \quad (7.13)$$

We thus see that kernel regression is effectively a form of *weighted regression*, where the weights feature the use of a kernel.

### 7.3.3 A Variety of Kernels

Note that in our description of kernel regression and the computation of weights,  $w_i$ , we have not yet described the nature of  $K$ , that is, the *kernel function* we are choosing to use for the given problem and by which we are applying to  $[(x_i - x_0) / h]$ . Not surprisingly, there are many choices for a kernel function. One choice, perhaps the default, is to use the *Gaussian* (or “normal”) kernel:

$$K(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \quad (7.14)$$

where the bandwidth in (7.14) is the standard deviation. As noted by Fox (2016, p. 530), distances greater than  $2h$  from the focal value of  $x_0$  receive virtually no weight due to the nature and shape of the normal density.

Following Fox (2016, p. 530), another possibility is to use the *tricube kernel*,

$$K_T(z) = \begin{cases} (1 - |z|^3)^3, & |z| < 1 \\ 0, & |z| \geq 1 \end{cases} \quad (7.15)$$

or the *rectangular kernel*,

$$K_R(z) = \begin{cases} 1, & |z| < 1 \\ 0, & |z| \geq 1 \end{cases} \quad (7.16)$$

There are other kernels we can choose from, however according to Fox (2016), the choice of kernel function is not that critical in carrying out the regression. There is nothing wrong with trying out a variety of kernels to study their impact on the given regression solution.

### 7.3.4 Kernel Regression Is Not Nonlinear; It Is Nonparametric

We reiterate that kernel regression should not be considered specifically a nonlinear approach to regression in the same vein as polynomial regression (in terms of modeling nonlinear trends, not the model being nonlinear). Rather, it is a *nonparametric approach*, which implies that it *learns from the data*. If that learned form is linear, then so be it, but the point is that kernel regression does not *a priori* assume a linear or nonlinear form. It is first and foremost nonparametric because it essentially makes no parametric assumptions regarding the shape of the true function. *Local-polynomial regression* extends kernel regression where polynomials are instead fit at the focal value of  $x_0$ ; however this is beyond the scope of this book. For an excellent discussion of kernel regression (and much of anything else to do with regression), see Fox (2016).

### 7.3.5 Kernel Regression in R

Kernel regression in R can be fit using the package *gplm* (Mueller, 2016). We adapt an example from the package's documentation below, where we first create data on 1000 observations for variable  $x$ :

```
> library(gplm)
> n <- 1000
> x <- rnorm(n)

> head(x)
[1] -0.71040656 0.25688371 -0.24669188 -0.34754260 -0.95161857 -0.04502772
```

We next create  $m$  as a function of  $x$  and then  $y$  as a function of  $m$ :

```
> m <- sin(x)
> head(m)
[1] -0.65214204 0.25406776 -0.24419733 -0.34058836 -0.81435593 -0.04501251

> y <- m + rnorm(n)
> head(y)
[1] 0.2638500 1.0546901 -1.1807664 -1.7413758 -0.6540784 -0.3189749

> plot(x,y,col="gray")
> lines(kreg(x,y),lwd=2)
```

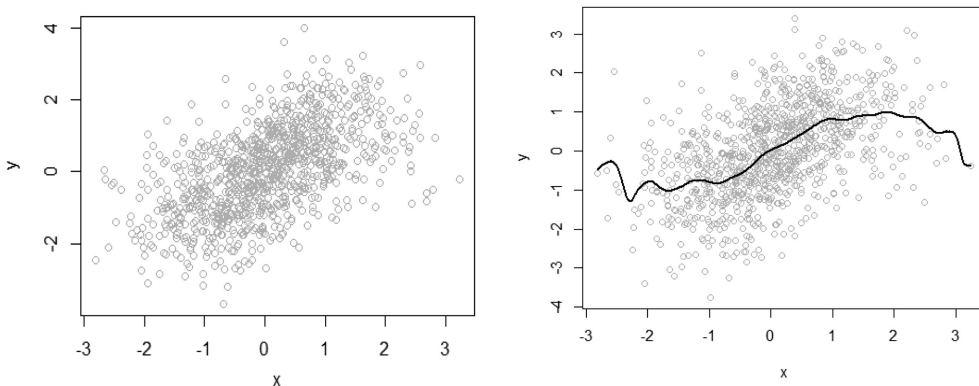


Figure 7.6 Plot of kernel regression line (right).

The squiggly line (Figure 7.6) is the kernel regression estimate, using the Nadaraya-Watson estimator (see Nadaraya, 1964). See the package for many more functions that can be applied to the kernel regression, including extracting estimated fitted values.

The *gplm* package (Mueller, 2016) also features rather sophisticated 3-D visualizations of kernel regression. As an example (see the package for the code), Figure 7.7 is an advanced graphic generated, where the original function is on the left and the estimated function is on the right.

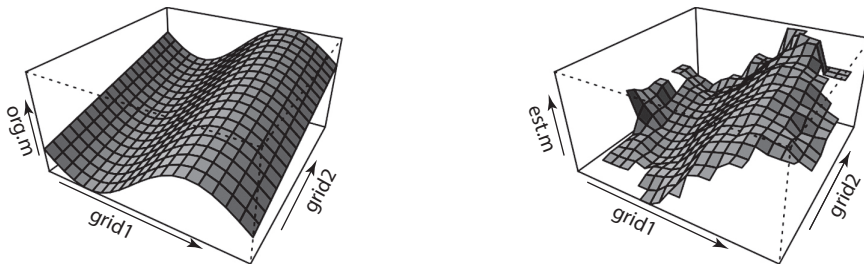


Figure 7.7 Graphic generated using the *gplm* package in R with both original (left) and estimated (right) functions.

Kernel regression can also be fit using the *np* package (Hayfield & Racine, 2008).

#### 7.4 Chapter Review Exercises

1. Explain how simple or multiple linear regression are special cases of the more general *polynomial regression model*.
2. Generate data in Python and fit a *quadratic polynomial*. Generate the data so that the fit is quite good.
3. Perform the polynomial regression in #2 in R.
4. Explain the concept of a *basis function* and discuss how basis functions are apparent even in univariate procedures such as *t*-tests and linear contrasts.
5. Compare *polynomial regression* to *regression splines*. How are they similar? Different? When might one be preferable over the other?
6. Explain how *kernel regression* goes about providing a more “local” means of estimating the true function behind data.
7. Generate a plot of the *rectangular kernel function* either manually or using software.
8. Distinguish kernel regression as being *nonlinear* vs. *nonparametric*. Why might some refer to kernel regression as nonlinear?
9. Perform a kernel regression in R using the *gplm* package, where sample size is equal to 10,000. Be creative in how you generate vectors for  $\mathbf{y}$  and  $\mathbf{x}$ .
10. Suppose a kernel regression gives vastly different results than a linear regression; does such imply that one model is more “correct” than the other? Why or why not? Brainstorm and explain.

## 8 Generalized Linear and Additive Models

### Logistic and Related Models

Up to now in our discussion of modeling, we have assumed the response variable is more or less *continuous*, or at least “pseudo-continuous,” since no empirical variables are ever truly continuous in the idealistic mathematical sense. Continuity itself is an *abstraction*, a philosophical idea that is expressed mathematically using such things as epsilon-delta definitions in real analysis. More pragmatically, we have assumed the response exhibits a rather wide range of “dense” values and the goal has been to predict a value of the response along this continuum based on one or more predictors. While many of these models have been linear, we have also surveyed a number of nonlinear models. However, in all of these models, response variables have all been assumed to be continuous and treated as such in our analyses. There are many instances in modeling where the response variable is not continuous but is instead *categorical*. Such responses contain mutually exclusive categories instead of a continuum of numerical possibilities. Some responses having this property are naturally occurring (e.g., pass vs. fail), while others can be operationalized by the user into specific categories (e.g., numerical score on a test classified into the binary outcome of pass vs. fail).

How can statistical models deal with binary or categorical responses? Thanks to the theory of the *generalized linear model* (McCullagh & Nelder, 1989), binary and polytomous responses can be situated in the same general framework that we can find classic linear models. The generalized linear model contains the entire family of these models and is the topic of this chapter.

Before surveying the generalized linear model, we should ask an important question from the outset: *Is “binning” a response a good idea?* What are the advantages and disadvantages of doing so? We begin this chapter with a brief survey of the benefits and drawbacks of operationalizing the response as a binary or polytomous outcome.

#### 8.1 How to Operationalize the Response

As mentioned, some response variables are naturally categorical. For instance, a salmon fish is either going “upstream” or “downstream,” and cannot be going both (though as we have all seen, they can just seemingly wade in the water without direction at times). When responses take on this quality, leaving the variable “as is” and treating it as binary is an easy decision since leaving it *as is* maintains it in its natural *measured* state. In other cases, however, the researcher may actually choose to bin the variable into mutually exclusive categories. In the machine learning literature, the pseudo-continuity of a variable before “binning” it is sometimes referred to as a *pre-classifier* (Kroese et al., 2020) because it is relatively continuous before forcing those values into categories.

### 8.1.1 Pros and Cons of Binning

Whether or not to bin a variable is an important decision. Indeed, different operationalizations of a single variable can lead the resulting statistical model to “discover” different effects. As a simple example, the variable *grade* on a class test operationalized as *pass* vs. *fail* may be predictive of final grade, but operationalized as a numerical score may alter the predictive ability. The variable is the same, but how we have operationalized it is now different. What is more, the choice of how to operationalize a variable in analysis of variance (ANOVA) can influence the resulting mean squares and ultimately the rejection of the null hypothesis. There is a whole literature on the influence of operationalizing effects in ANOVA (for example, see Prentice & Miller, 1992).

### 8.1.2 Detecting New Classes or Categories

While we have discussed the pros and cons of binning, it is worth mentioning as well that in machine learning, there are also techniques that actually allow the algorithm to detect new classes or categories based on the algorithm’s examination of the data. Such is referred to as *novelty detection* in machine learning circles, so that instead of classifying cases in one of  $C_l$  classes, a new class label  $C_{l+1}$  is derived not by the researcher’s judgment or decision of how to bin, but from the actual algorithm discovering the new category. This process sometimes goes by the general concept of *open set recognition* (Murray, 2022). In this book, we do not survey methods that detect new categories. We assume the researcher is actively deciding how to operationalize their variables.

## 8.2 The Generalized Linear Model

The motivation behind the generalized linear model (GLM) is to encompass models that have continuous responses such as in classic ordinary least-squares (OLS) regression, but also incorporate models with binary or categorical responses. The GLM is the “parent” to all of these models and essentially features a transformation of binary or polytomous responses into linear and continuous ones. In developing the GLM, it is first best to recall the classic linear model featured earlier in our discussion of regression:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon_i \quad (8.1)$$

Note that the above model is *linear in the parameters*, meaning that  $\beta_0, \beta_1, \dots, \beta_p$  are all raised to the exponent “1.” Even if  $x_1$ , for example, was raised to the second power,  $x_1^2$ , the model would still be regarded as a linear model so long as  $\beta_0, \beta_1, \dots, \beta_p$  remained raised to a coefficient of one; however, one could say it is no longer linear in the variables. *Linear models require only that coefficients be raised to no terms greater than one.* What basis transformations on the actual variables we perform is inconsequential as it pertains to the linearity or nonlinearity of the actual model in statistical parlance (Bishop, 2006). For example, the following model would rightly be considered *nonlinear* because it is nonlinear in the coefficients, notably the  $\beta_2^2$  term:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2^2 x_2 + \varepsilon_i \quad (8.2)$$

Note that the following model is also nonlinear because of  $\beta_2^2$  but not because of  $x_2^2$ ; it is considered both nonlinear in the *parameter* and *variable*:



$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2^2 + \varepsilon_i \quad (8.3)$$

### 8.2.1 *Intrinsically Linear Models*

A useful distinction to make when it comes to linear vs. nonlinear models is that between *intrinsic linearity* vs. *non-intrinsic linearity* (Neter et al., 1996). A model that is nonlinear, but that can be linearized through a transformation is often referred to as an *intrinsically linear model*. Perhaps the most classic example of this is the *exponential response*, where if we were to take the log of the exponential in such a model, we would obtain a linear model. That is, we can transform the response into one that is linear. It is intrinsically linear because its nonlinearity is a product only of a transformation. A *non-intrinsically linear model*, on the other hand, is one that cannot be linearized via transformation.

### 8.2.2 *General vs. Generalized Linear Models*

Many of the models featured thus far in this book can be considered special cases of the wider *general linear model*. Regression and analysis of variance models are indeed special cases of this wider framework. But what defines a general linear model exactly? When we refer to a specific model arising from this “general” framework, we usually mean that the distribution of the error term,  $\varepsilon_i$ , has an expected *normal distribution*. The classic linear regression model,  $y_i = \beta_0 + \beta_1 x_1 + \varepsilon_i$  is one such example where  $\varepsilon_i \sim N(0, \sigma_\varepsilon^2)$ , meaning that the error in the model is expected to be normal, having a mean of 0 and a finite and measurable variance,  $\sigma_\varepsilon^2$ . There are many situations, however, where we would not naturally expect the error term to behave this way. One classic and popular example is when the response variable is not continuous but instead *binary*. When a response variable is binary, the expectation of the error term is, not surprisingly, not normal. Nor could we reasonably expect errors to have constant variance (Fox, 2016).

The difficulty then on a theoretical level is how to deal with error distributions that have a non-normal expectation. They seem to be a different class of models altogether. However, thanks largely (but not exclusively) to the work of Nelder and Wedderburn (1972), a wider framework of statistical models was proposed that could encompass even these types of models with non-normal error distributions. Recall that understanding statistics is all about looking for *hierarchies*, of classifying “special cases” of models in the context of higher forms and generalities. This is precisely what these authors accomplished in proposing a framework for modeling that encompasses not only the classic linear model (regression, ANOVA) that features normally distributed error terms, but also models that contain non-normal error terms. This new class of models they called the *generalized linear model*, and hence a vast array of models can be situated in this general framework. That is, *the generalized linear model encompasses models that contain normally and non-normally distributed error terms*.

Essentially, the logic of the generalized linear model is to take an otherwise nonlinear model and make it mimic linearity through what is called a *link function*. Through the link function, the model becomes approximately linear and we can then treat the model more akin to how we would treat a classic linear model. This is the essential logic of the generalized linear model. An important distinction to make here is that it is the *decision surface* that is linearized, not truly the model entirely or necessarily (Bishop, 2006). Generalized linear models such as logistic regression engage in transformations that make them “linear-like” (Fox, 2016), such that otherwise nonlinear models can be “linearized” via the transformation (e.g., taking a link function such as the log of the odds in the case of logistic regression).



One very special case of the generalized linear model that has historically been very popular across virtually all scientific fields, especially medical fields, is the aforementioned *logistic regression model*, which we survey next.

### 8.3 The Logistic Regression Model

The logistic regression model is known as the *conditional Maxent model* (Mohri et al., 2018) in machine learning circles, which is simply the multinomial regression model of which binary regression is a special case. The logistic function is given by the following,

$$p(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \quad (8.4)$$

where we can work with the formulation in (8.4) and get

$$\frac{p(x)}{1 - p(x)} = e^{\beta_0 + \beta_1 x} \quad (8.5)$$

The left-hand side of (8.5),  $p(x)/[1 - p(x)]$  is what is referred to as the *odds*. In words, the odds is the probability of an event occurring,  $p(x)$ , relative to the probability that the event does not occur,  $1 - p(x)$ . If the probability of an event occurring is the same as the probability of it not occurring, then the odds are equal to 1.0. For example, suppose  $p(x) = 0.5$ , then it stands that,

$$\frac{p(x)}{1 - p(x)} = \frac{0.5}{1 - 0.5} = \frac{0.5}{0.5} = 1.0$$

Suppose we were to take the logarithm on both sides of

$$\frac{p(x)}{1 - p(x)} = e^{\beta_0 + \beta_1 x}$$

so that,

$$\log \left[ \frac{p(x)}{1 - p(x)} \right] = \log \left[ e^{\beta_0 + \beta_1 x} \right] \quad (8.6)$$

Recall that taking the log of an exponential function  $e$  has the effect of bringing the exponent down and so what we have from the above is

$$\begin{aligned} \log \left[ \frac{p(x)}{1 - p(x)} \right] &= \log \left[ e^{\beta_0 + \beta_1 x} \right] \\ &= \beta_0 + \beta_1 x \end{aligned} \quad (8.7)$$

That is, *the log of the odds is assumed to be a linear function*. The log of the odds goes by the name of the *logit*, and hence once we produce the logit, we are effectively now working with an approximate linear model similar in many respects to the classic linear model of regression. Of course, it is not *exactly* the same, since the units we are working in are no longer the natural units of the original variable, but instead are in units of the *logged odds*, or the *logit*, and hence the interpretation of parameter estimates will be in units of the logit.

For example, suppose that in the model of (8.7),  $\beta_1$  were equal to 0.8; what would be the interpretation? In classic least-squares regression, the interpretation, as we recall, would be *for a one-unit increase in  $X$ , we expect  $Y$  to increase by 0.8 units*, given the model under test. In logistic regression, the interpretation is the same, only that the “units” now are not natural units of the original variable, but instead are units of the *logit*, and hence, our interpretation would be *for a one-unit increase in  $X$ , we expect the logit of  $Y$  to increase by 0.8 logits*, given the model under test.

### 8.3.1 Odds and Odds Ratios

Recall that we said the odds were defined as the probability of an event occurring relative to the probability of the event not occurring, that is,

$$\frac{p(x)}{1 - p(x)}$$

The *odds ratio* is, as the name suggests, a *ratio of odds*, comparing the odds for one variable,  $x_1$ , or category of a response, to the odds of another variable,  $x_2$ , or complement category to  $x_1$ :

$$OR = \frac{\left( \frac{p(x_1)}{1 - p(x_1)} \right)}{\left( \frac{p(x_2)}{1 - p(x_2)} \right)} \quad (8.8)$$

For example, if the odds are the same for each variable or category, then the odds ratio is equal to one. An odds ratio greater than one implies that the odds for  $x_1$  are greater than the odds for  $x_2$ . Conversely, an odds ratio of less than one implies that the odds for  $x_1$  are less than the odds for  $x_2$ .

### 8.3.2 Logistic Regression in R

We can use the *blorr* package (Hebbali, 2020) for running logistic models. In what follows, we run a logistic regression on the *hsb2* data featuring two hundred observations that were randomly sampled from a survey conducted on high school seniors by the *National Center of Education Statistics*:

```
> library(blorr)
> head(hsb2)
```

|   | Id  | female | race | ses | schtyp | prog | read | write | math | science | socst | honcomp |
|---|-----|--------|------|-----|--------|------|------|-------|------|---------|-------|---------|
| 1 | 70  | 0      | 4    | 1   | 1      | 1    | 57   | 52    | 41   | 47      | 57    | 0       |
| 2 | 121 | 1      | 4    | 2   | 1      | 3    | 68   | 59    | 53   | 63      | 61    | 0       |
| 3 | 86  | 0      | 4    | 3   | 1      | 1    | 44   | 33    | 54   | 58      | 31    | 0       |
| 4 | 141 | 0      | 4    | 3   | 1      | 3    | 63   | 44    | 47   | 53      | 56    | 0       |
| 5 | 172 | 0      | 4    | 2   | 1      | 2    | 47   | 52    | 57   | 53      | 61    | 0       |
| 6 | 113 | 0      | 4    | 2   | 1      | 2    | 44   | 52    | 51   | 63      | 61    | 0       |

```
> model <- glm(honcomp ~ female + read + science, data = hsb2,
+ family = binomial(link = 'logit'))
```

```
> summary(model)

Call:
glm(formula = honcomp ~ female + read + science, family = binomial(link = "logit"),
    data = hsb2)

Coefficients:

                Estimate Std. Error z value Pr(>|z|)
(Intercept)  -12.77720     1.97555  -6.468 9.95e-11 ***
female1       1.48250     0.44736   3.314 0.00092 ***
read          0.10354     0.02576   4.019 5.86e-05 ***
science       0.09479     0.03045   3.113 0.00185 **

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 231.29 on 199 degrees of freedom
Residual deviance: 160.24 on 196 degrees of freedom
AIC: 168.24

Number of Fisher Scoring iterations: 5
```

We can see that all effects are statistically significant ( $p < 0.01$ ). We can see that the *residual deviance* is quite a bit smaller than the *null deviance*. The residual deviance here is the deviance on which the model is based, which is reflected by the lesser degrees of freedom (196). When we fit the model, we expect the *residual variance to be less than the null deviance*, akin to error sums of squares being smaller than total sums of squares when we fit an ANOVA model.

### 8.3.3 Logistic Regression in Python

We will use the `LogisticRegression()` function in Python to fit a simple logistic model:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
```

We print only a few cases of `X` and `y`:

```
X
Out [219]:
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3.0, 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5.0, 3.6, 1.4, 0.2],
```

```

y
Out[220]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

clf = LogisticRegression(random_state=0).fit(X, y)
clf.predict(X[:2, :])
Out[132]: array([0, 0])

```

In the above, `X[:2, :]` denotes the first two lines of `X` (had we wanted only the first case, we would have coded `X[:1, :]`):

```
X[:2, :]  
Out[32]:  
array([[5.1, 3.5, 1.4, 0.2],  
       [4.9, 3.0, 1.4, 0.2]])
```

The following are the probability estimates ordered by class of species:

```
clf.predict_proba(X[:2, :])
```

Out [133]:

```
array([[9.81804729e-01, 1.81952565e-02, 1.43620840e-08],  
       [9.71733109e-01, 2.82668608e-02, 3.00431075e-08]])
```

Recall that the first class is equal to 0 in our data, which is why when we coded `clf.predict(X[:2, :])` just above, it returned predictions of `[0, 0]` because the probabilities for species = 0 are highest above (9.81 . . -01 and 9.71 . . -01) when compared to species 1 and 2.

We can obtain a mean classification accuracy using the model:

```
clf.score(X, y)
Out[134]: 0.9733333333333334
```

The mean accuracy on the given data is equal to 0.97 which for most data sets is quite high. We can also use `statsmodels()` to fit logistic regression. See Denis (2020) for details.

## 8.4 Generalized Linear Models and Neural Networks

Recall that in our survey of linear and nonlinear models in previous chapters, we purposely drew a link to neural networks in a way to foreshadow them later in the book. Not surprisingly, we can likewise draw a link between the generalized linear model and that of neural networks. As we will see in a later chapter, neural networks are defined by an *activation function* which models a linear sum in a (typically) nonlinear fashion. Do generalized linear models have such a link function? Yes! The generalized linear model can be given by

$$\mathbb{E}[Y | X = x] = h(\mathbf{x}^T \boldsymbol{\beta}) \quad (8.9)$$

where the function  $h$  is the *activation function*. Note that the rest of the expression, that of  $\mathbf{x}^T \boldsymbol{\beta}$ , is simply the generalized linear model component we surveyed earlier. Reduced to a linear model,  $\mathbf{x}^T \boldsymbol{\beta}$  is simply the systematic portion of a linear model with activation function set at unity, such that  $h(\mathbf{x}^T \boldsymbol{\beta}) = 1(\mathbf{x}^T \boldsymbol{\beta}) = \mathbf{x}^T \boldsymbol{\beta}$ . In generalized linear models, this *concept* is referred to more generally as the *link function*. Hence, we see that while the generalized linear model is not as deep as neural networks typically are (as we will see later), both generalized linear models and neural networks can be said to feature the unifying notions of *activation* and *link functions*. This point is important, as the *concept* of activation functions is often regarded as “unique” to neural networks. However, they are much more general than they may at first appear. Indeed, prior familiarity with the generalized linear model allows one to better “see” the activation function featured in neural networks. The activation function, most generally, is simply the outer function that is being applied to the inner function, and is usually, *though need not necessarily be*, of nonlinear form. Both the generalized linear model and neural networks are unified by the concept of *composite functions*.

### 8.5 Multiple Logistic Regression

In our survey of logistic regression thus far, we have featured the one-predictor logistic model. As we did in generalizing simple linear regression to multiple, we can likewise generalize the logistic regression model to multiple logistic regression. Going from the one-predictor logistic model to the multiple logistic model is straightforward,

$$\begin{aligned} \log \left[ \frac{p(x)}{1 - p(x)} \right] &= \log \left[ e^{\beta_0 + \beta_1 x + \dots + \beta_p x_p} \right] \\ &= \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \end{aligned} \quad (8.10)$$

where, as before with the one-predictor model, we can write  $p(x)$  as follows:

$$p(x) = \frac{e^{\beta_0 + \beta_1 x + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x + \dots + \beta_p x_p}} \quad (8.11)$$

Interpreting coefficients in a multiple logistic setting is analogous to interpreting them in a multiple regression setting, meaning that coefficients must be interpreted *relative to the model* in which they appear. That is, each coefficient is interpreted in the *context* of other coefficients. For instance, the following interpretation for  $\beta_1$  would be *incorrect*:

*For a one-unit increase in  $x_1$ , we expect the logit to change by  $\beta_1$  amount.*

The correct interpretation is the following:

*For a one-unit increase in  $x_1$ , we expect the logit to change by  $\beta_1$  amount, **given the other predictors in the model**.*

As always, recall that when interpreting effects in a model, we are never, *ever*, interpreting them in a *vacuum*; they always occur or exist in the *context* of the other variables in the model. This is one of the key features to remember about virtually all statistical modeling as a whole, or any

narratives for that matter, that *there is always a context to the model you are interpreting, no matter how simple or complex that model may be*. The trailing phrase “given the other predictors in the model” or “given the context of the model” is not *optional* in interpretation; it is *essential* for a correct interpretation of modeling results. *Even in the case of a simple logistic regression with only a single predictor, the trailer “given the context of the model” is still required, even if that context is of no additional predictors. Interpreting univariate or bivariate effects in a multivariate or multivariable model without reference to context is equivalent to misinterpreting effects in the model. All models, as is true of all narratives, are context-dependent.*

### 8.5.1 Multiple Logistic Regression in R

The *glmnet* package (Friedman et al., 2010) for logistic regression in R can be used for multiple logistic regression. We adapt an example from the package’s documentation:

```
> library(glmnet)
> data(BinomialExample)
> x <- BinomialExample$x
> y <- BinomialExample$y
```

Above, we have created data on  $x$ , which are the predictors, and  $y$ , which is the binomial response. We proceed to fit the model on  $x$  and  $y$  and plot the result (Figure 8.1) to observe the behavior of the coefficients as a function of increasingly large L1 values:

```
> fit <- glmnet(x, y, family = "binomial")
> plot(fit)
```

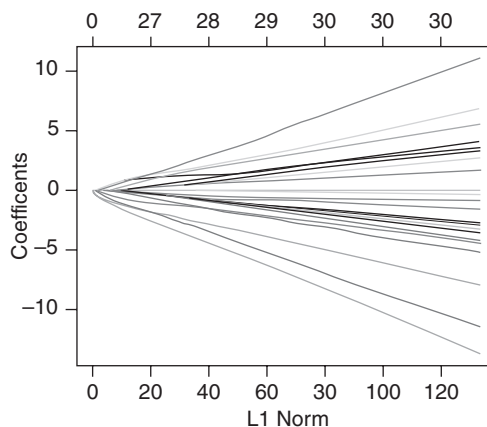


Figure 8.1 Coefficients as a function of L1 norm. As L1 decreases toward 0, the number of nonzero coefficients likewise goes to 0.

```
> predict(fit, newx = x[1:5,], type = "class", s = c(0.05, 0.01))
```

```
      s1  s2
[1,] "0" "0"
[2,] "1" "1"
[3,] "1" "1"
[4,] "0" "0"
[5,] "1" "1"
```

In the above,  $x[1:5,]$  indicates prediction for five new cases at values of  $\lambda$  equal to 0.05 and 0.01 [ $s = c(0.05, 0.01)$ ]. Had we changed, to say,  $x[1:10,]$ , we would be predicting for ten new cases. The specification `type = "class"` indicates the predicted class; `type = "coefficients"` yields the coefficients for each variable for each value of  $s$ :

```
> predict(fit, newx = x[1:10,], type = "coefficients", s = c(0.05, 0.01))
31 x 2 sparse Matrix of class "dgCMatrix"
```

```
      s1      s2
(Intercept) 0.20679667 0.208053336
V1           .      0.155759710
V2           0.27556586 0.817930902
V3          -0.16229700 -0.654411766
V4          -0.68944280 -1.301248267
V5          -0.07571269 -0.252512178
```

The following performs cross-validation (ten-fold) using misclassification error as the objective function to be minimized (Hastie et al., 2023, p. 17):

```
> cvfit <- cv.glmnet(x, y, family = "binomial", type.measure = "class")
> plot(cvfit)
```

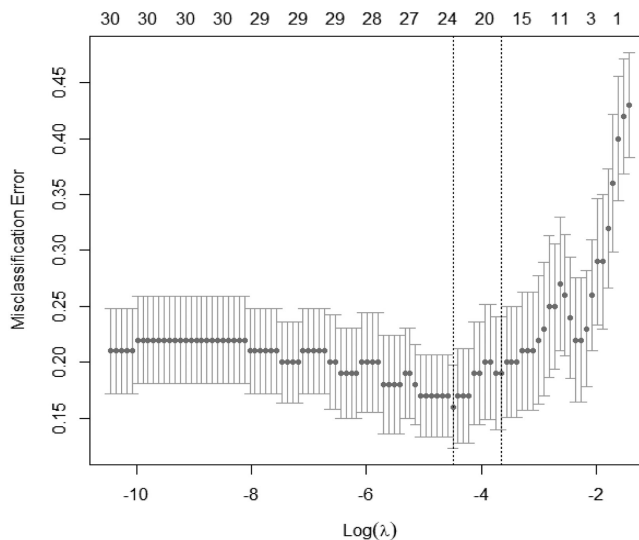


Figure 8.2 Misclassification error as a function of  $\lambda$ .

```
> cvfit
Call: cv.glmnet(x = x, y = y, type.measure = "class", family = "binomial")
Measure: Misclassification Error
```

|     | Lambda  | Index | Measure | SE      | Nonzero |
|-----|---------|-------|---------|---------|---------|
| min | 0.01116 | 34    | 0.16    | 0.03712 | 24      |
| 1se | 0.02578 | 25    | 0.19    | 0.05044 | 17      |

We see from Figure 8.2 that the value for lambda that produces the minimal misclassification is 0.01116 (`cvfit$lambda.min`), of which when we take the natural log ( $\ln$ ), we obtain  $-4.495$ , which agrees with what we see in the plot (i.e., left vertical line in the plot). The line to the right corresponds to `cvfit$lambda.1se` (i.e., within one standard error).

## 8.6 Poisson Regression

While logistic regression assumes the response is binary, Poisson regression assumes the response has a Poisson distribution, which may or may not be skewed depending on the value for  $\mu$ . A Poisson regression is often used when the response variable is a *count*. The distribution of a Poisson variable is the following:

$$p(y) = \frac{\mu^y e^{-\mu}}{y!} \quad (8.12)$$

where  $\mu$  is the mean of  $y$  and  $y!$  is equal to  $y(y-1)(y-2)\cdots(1)$ . The Poisson distribution is useful for modeling  $y$  when such is the number of events occurring during a fixed time interval of a given length. The expectation of  $\mathbf{Y}$  conditioned on  $\mathbf{x}$ , that is,  $\mathbf{Y} / \mathbf{x}$ , where  $\mathbf{x}$  is a vector of predictor variables, is given by the following where the model coefficients  $\boldsymbol{\theta}$  appear in the numerator, that is, they are in the exponent to the  $e$  base:

$$E(\mathbf{Y} / \mathbf{x}) = e^{\boldsymbol{\theta}'\mathbf{x}} \quad (8.13)$$

Murray (2022) and Fox (2016) provide good treatments of Poisson regression.

### 8.6.1 Poisson Regression in R

Just as was true for logistic regression, Poisson regression can also be fit using the *glmnet* package (Friedman et al., 2010). Below we adapt an easy example from that package:

```
> library(glmnet)
> data(PoissonExample)
> x <- PoissonExample$x
> y <- PoissonExample$y
```

The above provides the data we require for the model, which we now fit, being sure to specify `family = "poisson"` whereas in logistic regression recall that the family was that of `binomial`:



```
> fit <- glmnet(x, y, family = "poisson")
> plot(fit)
```

In Figure 8.3 the size of coefficients as a function of the L1 norm is plotted. As the penalty increases, the L1 norm *decreases* until the number of nonzero coefficients (top axis) is equal to 0 for an L1 norm equal to 0.

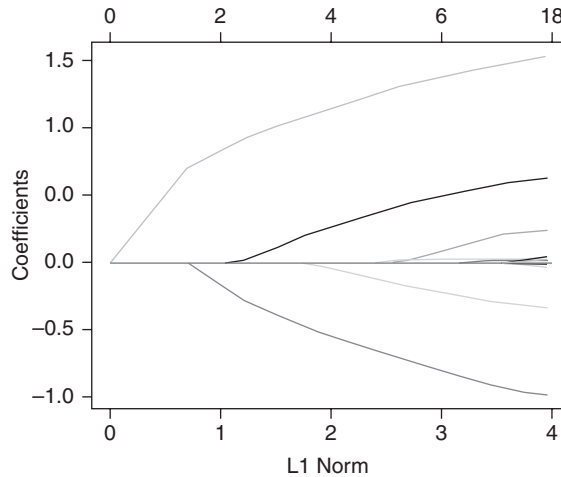


Figure 8.3 Coefficients as a function of L1 norm. A decrease in the L1 norm (i.e., an increase in the penalty) is reflected by having fewer nonzero (top-axis) coefficients.

We can apply the same functions as previously using *glmnet* for lasso (Hastie et al., 2023, p. 22):

```
> coef(fit, s = 1)
```

The above will provide us with the coefficients for penalty  $s = 1$ , of which we print the first few only:

```
21 x 1 sparse Matrix of class "dgCMatrix"
```

```

              s1
(Intercept)  0.61123371
V1           0.45819758
V2          -0.77060709
V3           1.34015128
V4           0.04350500
V5          -0.20325967
V6           .
V7           .
V8           .
```

We can see coefficients for V6, V7, and V8 have been driven to 0. We now obtain some predicted values for two values of  $s$  (0.1 and 1), that is,  $s = c(0.1, 1)$ :

```
> predict(fit, newx = x[1:5,], type = "response", s = c(0.1, 1))
```

```

           s1          s2
[1,]  2.4944232  4.4263365
[2,] 10.3513120 11.0586174
[3,]  0.1179704  0.1781626
[4,]  0.9713412  1.6828778
[5,]  1.1133472  1.9934537
```

We can perform cross-validation on the object:

```
> cvfit <- cv.glmnet(x, y, family = "poisson")
> cvfit
```

Call: `cv.glmnet(x = x, y = y, family = "poisson")`

Measure: Poisson Deviance

|     | Lambda  | Index | Measure | SE      | Nonzero |
|-----|---------|-------|---------|---------|---------|
| min | 0.05545 | 59    | 0.9748  | 0.05687 | 16      |
| 1se | 0.22383 | 44    | 1.0189  | 0.06113 | 7       |

We see the minimum value of deviance occurs at a lambda value of 0.0554 (Figure 8.4). We can visualize this in a plot, where the leftmost line corresponds to the value of  $\lambda$ , that is,  $\ln(0.05545) = -2.89$ . Not surprisingly, we see that deviance, recall which is loosely “error,” increases with increasingly large values of the tuning parameter:

```
> plot(cvfit)
```

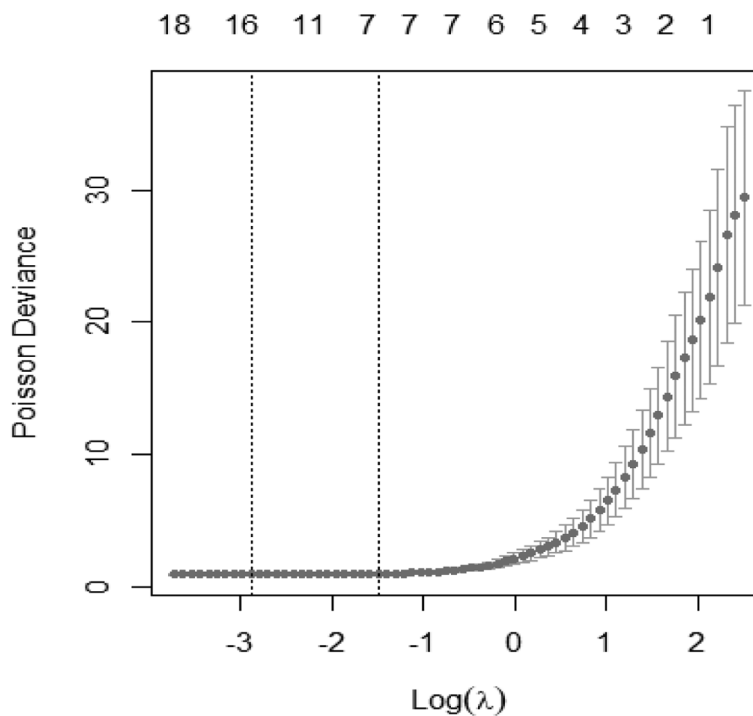


Figure 8.4 Poisson deviance as a function of lambda.

### 8.6.2 Poisson Regression in Python

In Python, we can use *sklearn* to fit a Poisson model. We create some easy data on  $x$  and  $y$ , and fit a basic Poisson model in what follows:

```
from sklearn import linear_model
clf = linear_model.PoissonRegressor()
X = [[1, 2], [2, 3], [3, 4], [4, 3]]
y = [12, 17, 22, 21]
```

Note from the above that there are two columns of  $X$  and a single column of  $y$ . We now proceed with the fit:

```
clf.fit(X, y)
Out [110]: PoissonRegressor()
```

The estimated regression coefficients for the model are the following:

```
clf.coef_
Out [112]: array([0.12109212, 0.15836976])
```

The intercept of the model is given by:

```
clf.intercept_
Out [113]: 2.0885914156053205
```

We now obtain some predicted values for  $y$  based on  $X$ :

```
clf.predict([[1, 1], [3, 4]])
Out [114]: array([10.67658784, 21.87505182])
```

Poisson regression can also be performed using `statsmodels()` by specifying a Poisson link.

## 8.7 Generalized Additive Models: A Flexible Nonparametric Alternative

The generalized linear model, with special cases such as logistic and Poisson regression, features “linear-like” or other *parametric* forms. That is, we approach the data with an *a priori* parametric idea or form in mind to fit to the data that particular form or preconceived structure. As emphasized throughout this book, however, there are many times where we do not want to approach the data with a *prior hypothesized structure* in mind. Instead, we would like to “discover” structure or form in the data without preconceived hypothesized ideas about that structure.

We have seen that the general linear model is a special case of the generalized linear model. However, the generalized linear model can be conceived as a special case of an even more general and wider model, that of the *generalized additive model* (Hastie and Tibshirani, 1986), or “GAM” for short, where in this formulation, a precise functional form is not specified *a priori*. In GAM,

we no longer assume linear forms such as  $\sum_{j=1}^J \beta_j x_j$ , even after transformation, but instead generalize this notion to  $\sum_{j=1}^J s_j(x_j)$ , where instead of linear regression weights  $\beta_j$ ,  $s_j$  are unspecified functions that are estimated using a nonparametric *scatterplot smoother* (or some other nonparametric form, such as a kernel estimate or spline) and an iterative procedure known as the *local scoring algorithm* (Hastie & Tibshirani, 1986) or *iterative weighted least-squares* (Fox, 2016, p. 575). The reader may recall from our earlier discussion in the book that smoothers are useful in providing a close fit to the data without generally making parametric assumptions *a priori*.

### 8.7.1 Why Use a Smoother Instead of Linear Weights?

The question naturally arises as to why use a smoother ( $s_j$ ) instead of linear weights ( $\beta_j$ )? What is the motivation for this approach? Like the motivation for the generalized linear model, the goal of the generalized additive model is to *capture nonlinear effects*, but by estimating them *nonparametrically*. Just as in kernel regression surveyed earlier, we wish to not make any parametric assumptions about the form of the relationship and instead want to have the data essentially speak for itself. This is what nonparametric estimation is all about. The idea of a smoother is a nonparametric idea in which we are allowing the data “to speak” in this sense and fit a function that approximates the data as well as possible, but without a prior hypothesized functional form such as we would have in the classic linear model estimated by least-squares, for instance. This is a hallmark of machine learning such that many analyses are *data-driven* (i.e., loosely, *nonparametric*) instead of *hypothesis-driven* (i.e., loosely, *parametric*). Generalized additive models, even in their own right, are pedagogically useful if for nothing else than understanding and appreciating how various models can be contextualized as special cases of others.

It can also be shown that GAMs can be likened to a neural network with a logistic activation function (Izenman, 2008). Further, generalized additive models can also be interpreted as extensions to both the class of additive models, and as we have already seen, the class of generalized linear models.

*The big idea of the generalized additive model is to use a nonparametric smoother (or some other nonparametric structure) to explore nonlinearity in the relationship between the response and covariates. Whereas the generalized linear model assumes some functional form (i.e., a linear form), the generalized additive model uses the data entirely to determine fit and hence makes no a priori parametric assumptions of what class of parametric functions will fit best. The generalized linear model can thus be considered a special case of the wider nonparametric generalized additive model.*

### 8.7.2 Deriving the Generalized Additive Model

To understand and appreciate how the generalized additive model arises, we begin once more with what is most familiar, that of the standard linear model, where the expectation of  $y$  is conditional on predictors  $x_1, x_2, \dots, x_p$ :

$$E(y | x_1, x_2, \dots, x_p) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \quad (8.14)$$

The generalized additive model makes the classic linear model a special case of

$$E(y | x_1, x_2, \dots, x_p) = s_0 + \sum_{j=1}^p s_j(x_j) \quad (8.15)$$

The distinction between models is that  $s_j(x_j)$  replaces  $\beta_j(x_j)$  and are now *smooth functions*. These functions  $s_j(x_j)$  are also typically standardized (Hastie & Tibshirani, 1986) such that  $E(s_j(x_j)) = 0$ . The functions  $s_j(x_j)$  are estimated, as mentioned, via scatterplot smoother or some other nonparametric device. The word “smooth” here generally refers to the idea of “nonparametric” because it implies the local area of the data being “smoothed” by a flexible curve instead of a rigid *a priori* (typically *parametric*) structure (e.g., a line or other curve). Cubic smoothing splines or the kernel smoothers that we saw earlier in the book are special cases. GAMs, however, place no restriction on what *can* be used, and hence the family of potential nonparametric functions is much wider. The algorithm used for estimating parameters in a generalized additive model is then tasked with estimating all the  $p$  functions in the data, whatever we have chosen those functions to be.

### 8.7.3 GAM as a Smooth Extension to GLLM

As we have seen, the generalized additive model is entirely general in the sense that it can subsume a variety of forms. Parametric structures are hence a special case of the wider GAM framework. The general linear model is easily cast as a GAM and so is the generalized linear model also a special case of the wider generalized additive model. Since in this chapter we are focused on the generalized linear model, we now detail such a bit more deeply.

Recall that in a generalized linear model, the linear predictor, now in the language of the generalized additive model, is given by

$$\eta = s_0 + \sum_{j=1}^p s_j(x_j) \quad (8.16)$$

where in the above,  $s_1(\cdot), \dots, s_p(\cdot)$  are now, in the GAM, *smooth functions*, which are not in parametric form but will instead be estimated nonparametrically. The intercept,  $s_0$  is analogous from the generalized linear model to the GAM. Where the models differ is that in the classic “non-smooth” generalized linear model,  $s_j(x_j)$  was simply  $\beta_j(x_j)$  where  $\beta_j$  were the linear weights as in regression. That is, the GAM applies a flexible smooth function to predictors  $x_j$  instead of setting them as linear as in the classic generalized linear model. As Hastie et al. (2009, pp. 296–297) note,  $s_1(\cdot), \dots, s_p(\cdot)$  do not need to consist exclusively of nonlinear terms but may also contain linear and other possibilities. The possibilities are virtually endless in the GAM. For instance, Hastie et al. (2009, p. 297) give the case of the following model,

$$g(\mu) = x^T \beta + \alpha_k + f(z) \quad (8.17)$$

which is considered *semiparametric*, where  $x$  are variables modeled linearly, and  $z$  *nonparametrically*. The GAM easily accommodates such a model, whereas the general linear model or the generalized linear model would have difficulty doing so without alteration.

Other possibilities may consist of including qualitative variables or interaction terms as other examples. There is no real limit to the possibilities of combining parametric and nonparametric style inputs in the GAM and such decisions will be made in part based on the prior experience of the researcher but also on what the actual data have to say. While the generalized additive model is nonparametric, we note that deciding on what nonparametric “objects” to include in the model is still a decision that must be made by the researcher. Whether to include a kernel or spline or whether to mix linear and nonlinear terms is still up to the researcher. Hence, *nonparametric models learn from the data almost exclusively, but what tool does the learning is still a decision the researcher must make.*

There are numerous algorithms that can be used to fit GAMs. Some of the more popular include the *backfitting algorithm* and the *Newton-Raphson procedure* (Hastie et al., 2009, p. 299). Iterated weighted least-squares can also be adapted for GAMs and combined with the backfitting algorithm, the so-called IWLS-backfitting procedure (Fox, 2016, p. 576). Likelihood-ratio chi-square tests along with deviance can be used to assess model fit. Generalized additive models can also be extended to handle multinomial responses. For details, see Hastie et al. (2009) and Fox (2016).

#### 8.7.4 Generalized Additive Models and Neural Networks

Once again, we can draw a link between models, this time generalized additive models and neural networks. We can relate the generalized additive model to neural networks in the following way. Suppose we take our earlier function,

$$\eta = s_0 + \sum_{n=1}^p s_j(x_j) \quad (8.18)$$

and now take a *function of this function*,

$$\eta = g\left(s_0 + \sum_{n=1}^p s_j(x_j)\right) \quad (8.19)$$

The formulation in (8.19) for  $\eta$  is essentially that of  $g(f(x))$ ; that is, we are taking, in a general sense, a *composite function*, which, as we will see in our discussion of neural networks later, allows us to use backpropagation via the chain rule in calculus. Once more, as we do in several other places in the book, in appreciating the nature of neural networks it is helpful to see what other methods can be conceived as related or even special cases of that more global framework or approach. We encourage the reader to return to this and other sections in the book where we relate models to neural networks once we discuss and develop them more thoroughly later.

#### 8.7.5 Linking the Logit to the Additive Logistic Model

We have seen that generalized additive models are a way to deal with nonlinearities in our data via nonparametric estimation. Perhaps not surprisingly at this point, there is a natural connection between the logit link function and the additive logistic regression model (Hastie et al., 2009). Consider the logit once more, that is, the log of the odds,

$$\log\left(\frac{\mu(x)}{1-\mu(x)}\right) = \alpha + \beta_1 x_1 + \cdots \beta_p x_p \quad (8.20)$$

where now  $\mu(x)$  is the probability of the event of interest (e.g., survival) and  $1 - \mu(x)$  the probability of its complement (e.g., death). The *additive logistic regression model* exchanges each linear term by a more general and to-be-defined functional form  $f_1$  through  $f_p$ , generating the following function:

$$\log\left(\frac{\mu(x)}{1-\mu(x)}\right) = \alpha + f_1 x_1 + \cdots f_p x_p \quad (8.21)$$

As before, not all of the  $f_1 x_1 + \cdots f_p x_p$  need to be nonlinear. As was the case where the generalized additive model replaced terms in the general linear model with nonparametric estimates, the generalized additive model likewise replaces the parametric terms in the logistic generalized linear model with smooth nonparametric estimates.

The package *mgcv* (Wood, 2017) can be used in R for fitting generalized additive models.

## 8.8 Overview and Recap of Nonlinear Approaches for Nonlinear Responses

The purpose of this chapter thus far has been to introduce and survey ways of modeling and accounting for nonlinearity or unknown structure in data where the response variable is something other than a continuous variable and instead binary or polytomous. As mentioned from the outset, while many or even most statistical models are linear in nature, many if not most scientific *phenomena* are *nonlinear* or at minimum, unknown, such that we may be better off allowing the data to “speak for itself.” A powerful way to understand the range of nonlinear and nonparametric methods is to consider how far we wish to depart from linearity to capture nonlinear effects. In this section, we provide a global review of the strategies we have surveyed so far. To set things up for this review, consider the basic scatterplot (Figure 8.5) with a single response variable and two predictors.

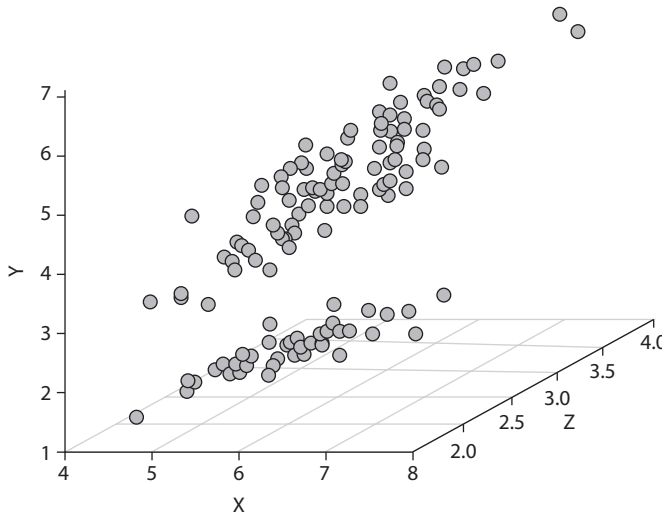


Figure 8.5 Generic scatterplot of  $y$  as a function of  $x$  and  $z$ .

The central question we wish to ask is:

*What are the options for modeling this data where  $y$  is hypothesized as a function of  $x$  and  $z$ ?*

We now review and summarize the ways of modeling such data discussed in this and the previous chapter, which include *linear*, *nonlinear*, as well as *nonparametric* approaches. For each of the following, we encourage the reader to keep the plot in Figure 8.5 in mind to help conceptualize how each of the following approaches would address this data:

1. *Linear Regression* – This classic approach will ignore any nonlinearity in the data and simply model a linear function to the data points. This can be a useful strategy if we are not that concerned about any nonlinear trend that may or may not be present in the data. Linear regression can be interpreted as a special case of the *general linear model*, which in turn can be considered a special case of the more *general nonlinear model* framework.
2. *Principal Components Regression* – In this approach, we generate principal components on the predictor side and use those component scores to predict the response variable. Recall that one limitation to this approach is that the components generated on the predictor side are not those necessarily maximally related to the variance in the response. That is, *there is no guarantee that the components generated vary in the directions of the response*. Still, this approach can be useful for reducing the data on the predictor side. However, the approach is still linear and will inadequately address any nonlinearities in the data. For the data depicted in Figure 8.5, the approach would not be fruitful since we only have two predictors. However, given a much larger number of predictors, dimension reduction on the predictor side before predicting the response may be an effective strategy.
3. *Partial Least-Squares Regression* – As was the case for principal components regression, we seek to reduce the set of predictors to a smaller set that can be used for prediction of the response. In doing so, PLS, unlike PCR (principal components regression), attempts to incorporate both the response and predictor sides of the equation.
4. *Multivariate Reduced-Rank Regression* – Many models can be conceived as a special case of the wider reduced-rank regression problem. This context is more pedagogical than applied, as most models we will fit will not go by this name.
5. *Regularization (ridge, lasso, elastic net)* – This approach seeks to regularize the model by the concept of *shrinkage*, where one or more predictors have weights driven toward (or *to*) 0. This serves to simplify the model so that predictors that are not essential are minimized in importance. While the ridge estimator will drive weights *toward* 0, lasso will drive them exactly *to* 0, making lasso a kind of variable-selection procedure. Elastic net can be interpreted as a kind of compromise between the ridge and the lasso.
6. *Least-Angle Regression* – This takes a linear and stepwise approach, but instead of including variables at each step, estimated parameters are increased in a direction *equiangular* (in part, motivating the name *least-angle*) to each one's correlations with the residual. That is, least-angle regression only includes that “part” of a predictor that is “next up” in terms of importance (at least, *statistical* importance) at each step.
7. *Canonical Correlation Analysis* – This relates each side of the equation just as in bivariate linear correlation, where each side is a set of variables. A Pearson Product-Moment correlation is a special case of the canonical correlation where each side of the equation consists of very simple linear combinations, those of only a single variable. That is, the “variate” in this case is a single variable on each side.



8. *Polynomial Regression* – This seeks to model a polynomial that is more general than a degree 1 polynomial (line). This is a *global approach* that seeks to capture the *global nonlinearity* or trend of the data. Linear models can be considered a special case of the wider and more general polynomial regression problem. Linear regression can generally be considered a special case of polynomial regression.
9. *Regression Splines* – In this approach, we choose splines as a *basis expansion* for our predictors. Regression splines are not *equivalent* to a basis expansion, but are merely a special case. Splines, as was the case for polynomial regression, are meant to capture nonlinearities in the data, but do so at a more *local level*. The locality of splines is what makes them so attractive and useful. Splines feature the use of “knots” that are employed to capture the local and non-linear segments of the data trend. It is generally regarded as a nonparametric technique as it uses the data (not the model) to determine the fit; however, elements of regression splines can also make them partially parametric.
10. *Kernel Regression* – This approach also seeks to model the data more locally using a kind of weighted regression and a kernel to draw focus to focal values in the regression while at the same time incorporating values of nearby or “neighboring” points. Kernel regression is a kind of weighted-average regression that extends notions of naïve nonparametric regression.
11. *Generalized Linear Models* – These employ a link function to “linearize” an otherwise nonlinear response such as one that is binary or polytomous. The response may also be categorical or count data, for which a Poisson link is an option. Many other link functions are available. The general linear model can be considered a special case of the generalized linear model where the link is the identity function.
12. *Generalized Additive Models* – These models feature nonparametric “smoothers” that generalize many other models, including the generalized linear model and polynomial models. The choice of what nonparametric smoother to use is left up to the researcher; however, common ones are smoothing splines, kernels, or many others. Pedagogically, the generalized additive model is very useful for conceptualizing a host of other models as special cases and in turn can also be considered a simplified version of neural networks.

## 8.9 Discriminant Analysis

Recall that we briefly introduced discriminant analysis toward the end of our discussion of the multivariate analysis of variance (MANOVA) model earlier in the book. There, we said that we would recap and extend on our discussion of the model later in the book when we introduced the generalized linear model. We discuss discriminant analysis here in this chapter on generalized linear models because it belongs to the types of models used for classification such as binary logistic regression and can be conceptualized as a special case of the generalized linear model with a binary response. Recall that the reason we originally introduced discriminant analysis in the context of MANOVA is that it shares many technical similarities with the MANOVA model and is often used as a follow-up analysis to the MANOVA procedure. Whereas in MANOVA we sought mean vector differences on a linear combination of response variables, in discriminant analysis we turned the problem around and sought to classify observations into groups based on a linear combination of predictors.

In the machine learning and statistical learning literatures, the presentations of discriminant analysis are rarely, if ever, related to multivariate analysis of variance. At least authors usually make no mention of the connection. Discriminant analysis is seen purely as a classification technique, one of the many *classifiers* featured in a wider discussion of multivariate and machine learning algorithms. We find the omission of multivariate analysis of variance in these contexts

unfortunate and a clear missed pedagogical opportunity. Because this is a book on both multivariate statistics *and* machine learning, we have chosen to take both approaches, where earlier in the book we discussed discriminant analysis in the context of MANOVA and where here we focus on the machine learning paradigm of discriminant analysis as purely a classifier without any mention of MANOVA.

We begin by discussing the most “optimal” strategy for classification, which is that of using Bayes. As we will see, *Bayes is best* when it comes to classification. For reasons that we will discuss, using Bayes for classification is often not possible, and hence our second choice will be to implement the discriminant analysis model to *approximate* the Bayes boundary.

As we will also see, linear discriminant analysis (LDA) is yet another example of a statistical methodology in which *generative assumptions* (such as linearity) greatly simplify the learning process and can afford greater predictability. This is the reason why developing the discriminant analysis model is worthwhile; otherwise we would be content with using such procedures as *k-nearest neighbors* which, as we will see, essentially makes no prior parametric assumptions about the data. The principle here is that, once more, *making generative assumptions simplifies the problem and in cases where those assumptions are satisfied, doing so yields great benefit*. In cases where assumptions are not satisfied, nonparametric approaches are fair competitors to parametric ones.

*Generative assumptions can greatly simplify the problem of modeling. If those assumptions are not met, however, nonparametric approaches can be useful, and in many cases are more appropriate competitors.*

### 8.9.1 *Bayes Is Best for Classification*

Recall the nature of Bayes’ theorem discussed earlier in the book,

$$p(y = k \mid x = x_i) = \frac{\pi_k f_k(x_i)}{\sum_{l=1}^K \pi_l f_l(x_i)} \quad (8.22)$$

where  $p(y = k \mid x = x_i)$  is the *posterior probability* of  $y$  equaling a particular class  $k$  given some particular value of  $x = x_i$ . The numerator in Bayes’ theorem,  $\pi_k f_k(x_i)$  contains the *prior probability*,  $\pi_k$ , multiplied by the density function of  $x_i$ ,  $f_k(x_i) = p(x = x_i \mid y = k)$ . Note that the final result of Bayes essentially “inverts” this conditional probability, that is, from  $p(x = x_i \mid y = k)$  to  $p(y = k \mid x = x_i)$ . In a nutshell, this is the goal of Bayesian analysis in a simple problem such as this, to “flip” or invert conditional probabilities.

Recall also that Bayes is not simply an “equation” or “rule” (i.e., “Bayes rule”) as above and how it can be “casually” represented in elementary statistics, but is more generally a *philosophy* of how theory or belief should be revised in light of data or experience more generally. As we did earlier in the book, a more general way of writing Bayes’ theorem is the following:

$$p(T \mid D) = \frac{p(T) \cdot p(D \mid T)}{p(T) \cdot p(D \mid T) + p(\bar{T}) \cdot p(D \mid \bar{T})} \quad (8.23)$$

where now  $p(T|D)$  is the probability of the theory  $T$  given data  $D$ . That is, given the numerator  $p(T) \cdot p(D|T)$ , the prior probability multiplied by the likelihood, we seek to “flip”  $p(D|T)$  to  $p(T|D)$ .

Why is Bayes so important? Why do we continuously hear about Bayes in statistics and science? Why are there entire *books* written on Bayesian statistics and Bayesian philosophy? Aside from the fact that Bayes’ theorem has a very long history and is the cornerstone of a branch of statistics appropriately called *Bayesian statistics*, in our current context of classification it is especially important and meaningful because the Bayes’ boundary, or more generally, the Bayes *classifier* has the lowest error rate out of all classifiers (Cover and Hart, 1967; James et al., 2013), *under the assumption that all terms in Bayes’ theorem can actually be defined and measured*. Under such conditions, as summarized by Cover and Hart (1967, p. 23), “The Bayes risk serves as merely a reference – the limit of excellence beyond which it is not possible to go.”

Long story short, Bayes’ theorem is an optimal sought-after result when it comes to classifying outcomes. It is the theoretical *ideal*. It is what we would wish for all the time if we had the choice. For the 0–1 loss (binary) problem, the Bayes classifier has us classifying to the most probable class using the conditional (discrete) distribution. The error rate generated is called the *Bayes rate* and is the “holy grail” of error rates. Many other methods, both in statistics and machine learning, such as the *k-nearest-neighbor approach*, are meant as direct *approximations* to the Bayesian ideal. They are “second choices” to using a Bayes boundary. *Bayes is always first choice and is always best*. However, nearest-neighbor approaches can have tremendous value in approximating Bayes. Indeed, it has been shown that the nearest-neighbor probability of error,  $R$ , for a classification problem with  $M$  categories, is bounded below by the Bayes probability of error,  $R'$ , and above by  $R'(2 - MR'/M - 1)$ . As Cover and Hart have found, “any other decision rule based on the infinite data set cut the probability of error by at most one half. In this sense, half of the available information in an infinite collection of classified samples is contained in the nearest neighbor” (p. 27).

### 8.9.2 Why Not Always Bayes?

If Bayes is so wonderful, why not just use it all the time? Ideally, we would use Bayes for virtually all classifier problems. The problem, however, is that the terms needed for Bayes are usually not known. For instance, we do not know the term  $f_k(x_i)$  in Bayes, and hence, we need a second approach to *estimate* it. The linear discriminant analysis model, and other approaches, are used as *approximations* to the Bayes classifier. However, only when observations are multivariate Gaussian with equal covariance matrix between classes is linear discriminant analysis the estimated Bayes classifier (Hastie et al., 2009). Hence, our discussion of discriminant analysis should be “housed” in the context of trying to approximate Bayes, but all the while remembering that Bayes is regarded as the optimally desired approach if the terms in Bayes are available. In fictitious toy problems where terms may be available, *all else constant*, one would virtually always choose Bayes over discriminant analysis or any other classifier because it will provide the lowest of error rates. In reality, however, techniques such as discriminant analysis or competitors will usually be employed. Hence, we may say that Bayes is best; however, pragmatic concerns, including variations on sample size availability, will usually suggest a competing methodology such as the linear discriminant analysis model.

### 8.9.3 Linear Discriminant Analysis Model

We have said that we do not typically know  $f_k(x_i)$  in the Bayesian formulation. We would like to estimate it. To estimate it, however, we need to make some model assumptions about it. We begin by assuming that  $f_k(x_i)$  has a normal density. That is, we assume:

$$f_k(x_i) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x_i - \mu_k)^2\right) \quad (8.24)$$

where the right side of (8.24) is the normal density and  $k$  is the class or group number. Each class has a mean  $\mu_k$  and variance  $\sigma_k^2$ . In the current setting, there are only two groups on the response variable since it is binary classification, and hence  $k = 2$ . The density for the first group is thus equal to:

$$f_1(x_i) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left(-\frac{1}{2\sigma_1^2}(x_i - \mu_1)^2\right)$$

and for the second (notice the change in subscripts from 1 to 2),

$$f_2(x_i) = \frac{1}{\sqrt{2\pi}\sigma_2} \exp\left(-\frac{1}{2\sigma_2^2}(x_i - \mu_2)^2\right)$$

An important assumption made in the classic discriminant analysis model, one which we will relax later when we consider quadratic discriminant analysis, is that the standard deviation or variance is *constant* across both groups or levels on the response variable. That is, we assume

$$\sigma_1^2 = \dots = \sigma_k^2 \quad (8.25)$$

We will also assume equality of covariances between responses, which, along with the assumption of equality of variances, yields the *equality of covariance matrices assumption*. This is essentially the same assumption we encountered earlier in our original discussion of MANOVA and discriminant analysis earlier in the book, which as we saw we can evaluate using the *Box M test*.

### 8.9.4 How LDA Approximates Bayes

Here is the trick to understanding how the normal density can be used to approximate Bayes. We desire Bayes in the end, so we substitute in the previous normal densities for the Bayes densities we wish to estimate, those of  $f_k(x_i)$ , where the numerator is the density for the specific class we are modeling and the denominator is the sum of the classes, mimicking the structure of Bayes,

$$\begin{aligned}
p(y = k \mid x = x_i) &= \frac{\pi_k f_k(x_i)}{\sum_{l=1}^K \pi_l f_l(x_i)} \\
&= \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x_i - \mu_k)^2\right)}{\sum_{l=1}^K \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu_l)^2\right)}
\end{aligned} \tag{8.26}$$

where  $\mu_k$  and  $\sigma_k^2$  is the mean and variance for class  $k$ , the class we are modeling, where again, we assume  $\sigma_k^2$  is constant across classes. When we now take the log of equation (8.26), it can be shown that it is equivalent to assigning the observation to the class for which the following gives the largest quantity:

$$\delta_k(x_i) = x_i \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k) \tag{8.27}$$

But what is (8.27)? What is this function? This function is called the *linear discriminant function*. At a technical level, the linear discriminant function approximates the Bayes classifier by estimating  $\pi_k$ ,  $\mu_k$ , and  $\sigma^2$ , and substituting these into (8.27). Once we have those values estimated, the approximation to the Bayes classifier is complete.

#### 8.9.5 Estimating the Prior Probability and Parameters of the Discriminant Function

The estimate for the prior probability  $\pi_k$  can come from several sources, including simply the opinion or educated guess of the investigator. Recall that in Bayesian theory, the prior is essentially a *belief* espoused by the investigator. This isn't to suggest it is wise to have *irrational* beliefs about the prior, only that ultimately the prior can be set by the preference of the researcher. Most commonly, the prior is derived from prior knowledge or previous research of the problem under study, that is, from substantive knowledge of the particular problem. In the absence of such information, and in reality probably the most common way of estimating the prior in practice, we can obtain an estimate of  $\pi_k$  via

$$\hat{\pi}_k = n_k / n \tag{8.28}$$

where  $n_k$  is the number of cases in class or group  $k$  and  $n$  is the total number of cases. This can be seen as a “default” method or starting point to estimating the prior, but it should be emphasized that such priors can be “tweaked” in line with, again, what the researcher knows about the specific problem under consideration. Recall that by their nature, priors are *subjective probabilities* and hence there are no strict rules about them. Of course, if you advance an irrational prior, then few will take it seriously. The benefit of specifying the prior on an individual/personal basis is that any biases or even *irrationality* in the estimate is open for others to see. This is a strength of Bayesian approaches in general such that *they reveal your degree of belief, even if your degree of belief is irrational*.

In the discriminant function of (8.27), recall we also required values for  $\mu_k$  and  $\sigma^2$ , that is, the mean for each class and the variance of the classes. These can be estimated by

$$\hat{\mu}_k = \frac{1}{n_k} \sum_i x_i \quad (8.29)$$

and

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^K \sum_i (x_i - \hat{\mu}_k)^2 \quad (8.30)$$

where  $n_k$  is the number of cases in class  $k$  and  $n-K$  is the total sample size minus the total number of classes. We can see that  $\hat{\sigma}^2$  is essentially a variance estimate across classes, pooling across classes  $k=1$  to  $K$ .

All of the aforementioned principles apply equally to the case of multiple predictors. That is, in the case of more than a single predictor (*as different from multiple response categories*), instead of the univariate Gaussian distribution, we require the *multivariate Gaussian*, which recall is given by

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \quad (8.31)$$

where the only difference now is that we require a mean vector  $\boldsymbol{\mu}$  and covariance matrix  $\Sigma$  instead of only a mean  $\mu$  and variance  $\sigma^2$  as we required in the univariate case. This is so because now we have several predictor variables, and hence in each classification category there is a mean *vector* instead of simply a mean, and a covariance *matrix* instead of simply a variance (try sketching the multivariate LDA cells on paper).

The classification will assign  $x = x_i$  to the class for which the following multivariate discriminant function is the largest:

$$\delta_k(\mathbf{x}) = \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \log \pi_k \quad (8.32)$$

A comparison of the univariate discriminant function in (8.27) to that of the multivariate in (8.32) will reveal their similarities, especially if we keep in mind that  $\Sigma^{-1}$  is serving in place of  $\sigma^2$ .

## 8.10 Multiclass Discriminant Analysis

In multiclass discriminant analysis where we have more than two categories on the response, we can use a *Mahalanobis distance* metric to best separate classes from each other and provide a linear projection of dimensionality up to  $k-1$ , where again  $k$  is the number of groups or levels on the response variable. Why is the dimensionality limited to  $k-1$  dimensions? Why not exactly  $k$ ? The reason for this can be easily understood through the following simple example. Suppose you are asked to guess which hand behind my back holds the marble. You require only a single “discriminator” to make the selection. If not in the left hand, then it must be in the right, or vice versa. When we generalize to more than two groups, there is always one less the number of groups for which a “rule” can be established for a similar reason (*a weighted sum of predictors, the reverse of the MANOVA*

problem; see Eq. (8.35)). The “discriminator” is a rule that separates classes, and hence having a number of discriminators equal to the number of classes would make little sense. *The purpose of a discriminator is to differentiate between at least two objects. Discriminating between a single object makes no sense. Hence the reason why for  $k$  groups, only  $k - 1$  discriminant functions are required.*

### 8.11 Discriminant Analysis and Dimensionality in a Simple Scatterplot

The dimensionality aspect of discriminant analysis is perhaps best understood using an example from introductory statistics. Consider the bivariate scatterplot of data in Figure 8.6.

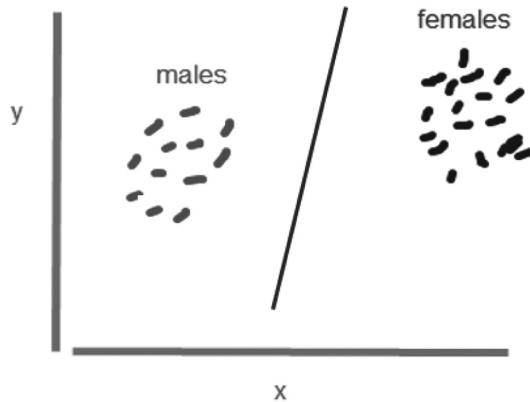


Figure 8.6 Scatterplot of data where  $x$ , the sole predictor and hence “linear combination” composed of only a single variable, discriminates well between males and females.

We can see in the plot that there are two groups: one group is male while the other is female. In this simple plot, we have the *concept* of a discriminating *dimension*; in this case, that dimension is made up of only a single variable. That is, the function, made up of a single variable, typically with an intercept, *separates well*. We see that it separates males from females along a *rule* (weighted sum of predictors) defined by the dimensionality of  $x$ . The dimension is simple in this case, but it’s still a dimension. To reiterate, *the dimension in this case is made up of only a single variable*. The extension to a more complicated and traditional LDA is that each dimension is made up of more than a single variable; it is composed of a *linear combination* or *composite* of variables.

If you keep this simple example in mind, you will always appreciate discriminant analysis as not only a “classifier,” but also as a means of *reducing the dimensionality* of data, similar in some respects, but by no means *equivalent*, to principal components analysis. The simple scatterplot featured in introductory statistics with two groups appearing in the plot could have been the starting point to discussing discriminant analysis because once we move on to considering  $t$ -tests and such in elementary statistics, we are in fact engaged at some level in at least the *concept* of discriminant analysis as the two techniques will share technical similarities, especially as we venture into multivariate  $t$ , which is Hotelling’s  $T^2$  (Rencher & Christensen, 2012). That “simple”  $t$ -test in “stats 101” was, in the end analysis, a first conceptual “opportunity” and preview of discriminant analysis.

As in traditional analysis of variance, we seek to minimize within-scatter and maximize between-scatter of the classes. The discriminant function guarantees that the separation is *maximal*.

By “maximal” we mean that the function separates better on the given data than any other function we could have used in the parametric family (e.g., linear models). It accomplishes this by finding the projection that maximizes the ratio of the following, called the *Rayleigh coefficient* (sometimes called *Rayleigh quotient*):

$$\max_a = \frac{\mathbf{a}^T \mathbf{H} \mathbf{a}}{\mathbf{a}^T \mathbf{E} \mathbf{a}} \quad (8.33)$$

where  $\mathbf{H}$  and  $\mathbf{E}$  are hypothesis and error matrices respectively, and  $\mathbf{a}$  are coefficients or *coordinates* (Hastie et al., 2009, p. 116) of the estimated linear combination, or more precisely, linear discriminant function.

We can perform some algebra on (8.33) and obtain

$$\mathbf{a}'(\mathbf{H}\mathbf{a} - \lambda \mathbf{E}\mathbf{a}) = 0 \quad (8.34)$$

We seek to know the values of  $\mathbf{a}$  (other than 0) that maximize values for  $\lambda$ . The solutions are found by

$$(\mathbf{E}^{-1}\mathbf{H} - \lambda \mathbf{I})\mathbf{a} = 0 \quad (8.35)$$

The solutions for  $\lambda$  and  $\mathbf{a}$  are the corresponding eigenvalue and eigenvectors of the discriminant function(s). Large eigenvalues are associated with discriminant functions that are “important” (*statistically*, not necessarily *substantively*) and essentially doing a lot of “work” in separating classes. *They are not variances in the context of LDA (in PCA, eigenvalues will be variances)* and hence are best considered measures of “importance” of the given discriminant function. Smaller eigenvalues are associated with less important functions. Such measures of importance can help us in deciding how many functions to keep from our analysis. If an analytical solution for discriminant analysis is unavailable or a gradient descent solution is preferred, gradient descent can be performed where reduction of error occurs by taking partial derivatives of the error  $E$  with respect to coefficient weights  $a_1, a_2, \dots, a_p$ ,

$$\nabla_a E = \left[ \frac{\partial E}{\partial a_1}, \frac{\partial E}{\partial a_2}, \dots, \frac{\partial E}{\partial a_p} \right]^T \quad (8.36)$$

where at each step, weights  $a_i$  are updated,

$$\Delta a_i = -\eta \frac{\partial E}{\partial a_i}$$

where  $\eta$  is the learning rate, and hence new weights are computed by  $a_i = a_i + \Delta a_i$  (Alpaydin, 2020).



## 8.12 Quadratic Discriminant Analysis

In the linear discriminant analysis above, recall that we assumed Gaussian distributions as well as equal covariance matrices among populations. More formally, we assumed that any given observation comes from a given class  $k$  such that

$$x_i \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

That is, we assume the covariance matrices  $\boldsymbol{\Sigma}_k$  are the *same* from population to population. What if, instead, we did not make this assumption, but allowed *flexibility* such that covariance matrices between populations were permitted to be *unequal* instead of *constrained* to be equal? Note our use of the word “flexibility” here, which means that instead of imposing the constraint/condition that the covariance matrices are equal, we will allow for unequal covariance matrices between populations and to incorporate these differences into our modeling. Allowing for this inequality has the effect of modeling a nonlinear or irregular (i.e., quadratic, in this case) decision boundary, and hence whereas classical LDA uses only class centroids to determine membership, QDA will use more than a single “prototype” (i.e., class centroids as well as covariance matrices). Indeed, LDA is known as a *prototype classifier* (Hastie et al., 2009).

When we allow for differing covariance matrices, we will then assign  $x = x_i$  to the class for which the following function is the largest, where one can readily see the relevant substitutions,

$$\begin{aligned} \delta_k(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \log |\boldsymbol{\Sigma}_k| + \log \pi_k \\ &= -\frac{1}{2} \mathbf{x}^T \boldsymbol{\Sigma}_k^{-1} \mathbf{x} + \mathbf{x}^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \log |\boldsymbol{\Sigma}_k| + \log \pi_k \end{aligned} \quad (8.37)$$

For a brief but excellent discussion of why the assumption of equal covariance matrices matters and the essential difference between linear vs. quadratic discriminant analysis, consult James et al. (2021). Simplified, the argument essentially goes like this: for example, with  $p = 3$  predictors, the covariance matrix looks as follows:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma^2 & \text{cov} & \text{cov} \\ \text{cov} & \sigma^2 & \text{cov} \\ \text{cov} & \text{cov} & \sigma^2 \end{bmatrix} \quad (8.38)$$

Notice that for  $p = 3$  predictors, there are six parameters to estimate, or more generally,  $p(p+1)/2$  parameters. That is,  $p(p+1)/2 = 3(3+1)/2 = 6$ . Even if we have  $k = 3$  groups, we still have only six parameters to estimate since we are assuming  $\boldsymbol{\Sigma}_k$  is constant across groups. However, what if we designated a *separate* covariance matrix per group? This would imply that we have  $p(p+1)/2$  parameters *per group*, which yields  $kp(p+1)/2$ . Thus, for the  $k = 3$  group problem, we will now have  $3(6) = 18$  parameters to estimate, which is a lot more than in the linear case. The assumption of equal covariance matrices induces linearity, which means the LDA model is *less flexible* since a great number of parameters are *a priori fixed*. That is, because of the equality condition, once we know one covariance matrix, we know them all. In the quadratic case, we are freeing this parameter to be different for each covariance matrix. Freeing and fixing parameters is a commonality in virtually all modeling. Nothing would stop us, for instance, from fixing or equating only a *few* parameters in the wider discriminant analysis framework, thereby building a more “customized”

model. Presumably, such action would be based on theory or prior information. In the absence of such information, LDA or QDA are usually sufficient as “off the shelf” methods.

QDA then, is the more *general model*, where LDA can be conceptualized and shown to be a special case of QDA where the assumption of equal covariance matrices is imposed. The linear case is essentially a special case of the wider quadratic case in which quadratic terms are canceled due to shared covariance (Bishop, 2006). Instead of  $kp(p+1)/2$  parameters to estimate in QDA, the linearity in  $x$  implies there to be only  $kp$  coefficients to estimate since we only have to count a single covariance matrix in our estimation, not a number equal to however many groups we have on the response (in our earlier example, that number was 3). While QDA is more flexible, it has been found that classification rules for QDA usually require larger sample sizes than for its LDA counterpart and may also be more sensitive to parametric assumptions overall. For a discussion, see Wald and Kronmal (1977). Indeed, it should be emphasized as well that maximal separation is not necessarily equivalent to *maximal classification*, as classification incorporates additional considerations (e.g., see Rencher & Christensen, 2012).

### 8.13 Regularized Discriminant Analysis

As we have seen, *linear* discriminant analysis is so popular, on a technical level, in part due to the assumption that covariance matrices are equal and linear separation is a good way to simplify the problem. It is a very easy and facilitating assumption to make about parametric structure. We have seen that by assuming equality of covariance matrices, it lessens the number of parameters that need to be estimated. Hence, even if the assumption is violated and covariance matrices are different to some extent, one convenient way to *regularize* in this setting is to simply and forcefully set them as equivalent, or to *pool* them. As Friedman (1989, p. 167) notes:

Even if the population class covariance matrices are substantially different, the decrease in variance accomplished by using the pooled covariance estimate can sometimes lead to superior performance, especially in small-sample settings. This is a large part of the reason for the success and popularity of linear discriminant analysis.

By *pooled covariance estimate* above is simply meant an average of covariance matrices and hence reducing the number of parameters that need to be effectively estimated. Recall that in general, when you simplify a model, you essentially regularize it as it is usually less prone to overfitting. However, doing so also risks the introduction of potential bias. Indeed, “regularizing” a linear regression equation to simply consist of an intercept is very unlikely to provide a good estimate of a bivariate relationship. In other words, over-regularizing usually *underfits* the data.

For details on how regularization can be done much more thoroughly in discriminant analysis beyond this simple trick, see Friedman (1989). Hastie et al. (2009, Chapter 18) also discuss discriminant analysis for the case in which the number of cases is greater than the number of variables, the so-called *diagonal LDA classifier*, where within-class covariance matrices are diagonal, thus implying independent predictors. Mixture models in discriminant analysis are also a possibility,

which combine several densities (typically Gaussian) and can be estimated via the EM algorithm to obtain maximum likelihood estimates.

## 8.14 Discriminant Analysis in R

We will fit a basic discriminant analysis on the iris data:

```
> library(MASS)
> data(iris)
> head(iris)
```

|   | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 2 | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |
| 3 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |
| 4 | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| 5 | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| 6 | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |

```
> linear <- lda(Species ~., iris)
> linear
Call:
lda(Species ~ ., data = iris)
```

```
Prior probabilities of groups:
      Setosa versicolor virginica
0.3333333  0.3333333  0.3333333
```

```
Group means:
```

|            | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|------------|--------------|-------------|--------------|-------------|
| setosa     | 5.006        | 3.428       | 1.462        | 0.246       |
| versicolor | 5.936        | 2.770       | 4.260        | 1.326       |
| virginica  | 6.588        | 2.974       | 5.552        | 2.026       |

```
Coefficients of linear discriminants:
```

|              | LD1        | LD2         |
|--------------|------------|-------------|
| Sepal.Length | 0.8293776  | -0.02410215 |
| Sepal.Width  | 1.5344731  | -2.16452123 |
| Petal.Length | -2.2012117 | 0.93192121  |
| Petal.Width  | -2.8104603 | -2.83918785 |

```
Proportion of trace:
```

|  | LD1    | LD2    |
|--|--------|--------|
|  | 0.9912 | 0.0088 |

The *coefficients of linear discriminants* are the estimated raw centered (not *standardized*) coefficients for each function and are to be read vertically for LD1 and LD2. These are the optimized weights that maximize separation between groups analogous in concept to regression weights that are estimated to minimize sum of squared error in a classic OLS regression problem. We can see

from the above that the first discriminant function is the most important, having a proportion of the trace equal to 0.9912, while the second is of minimal importance having a value of 0.0088, which implies the eigenvalue for the first function is much larger than that of the second. Clearly, the first discriminant function is much more “important” than the second when it comes to group separation. That is, the first discriminant function is, essentially, “doing all the work.”

Among a host of output from `lda()`, the following posterior probabilities of group membership are provided by obtaining predicted values:

```
> p <- predict(linear, iris)
$posterior
```

|    | setosa       | versicolor   | virginica    |
|----|--------------|--------------|--------------|
| 1  | 1.000000e+00 | 3.896358e-22 | 2.611168e-42 |
| 2  | 1.000000e+00 | 7.217970e-18 | 5.042143e-37 |
| 3  | 1.000000e+00 | 1.463849e-19 | 4.675932e-39 |
| 4  | 1.000000e+00 | 1.268536e-16 | 3.566610e-35 |
| 5  | 1.000000e+00 | 1.637387e-22 | 1.082605e-42 |
| 6  | 1.000000e+00 | 3.883282e-21 | 4.566540e-40 |
| 7  | 1.000000e+00 | 1.113469e-18 | 2.302608e-37 |
| 8  | 1.000000e+00 | 3.877586e-20 | 1.074496e-39 |
| 9  | 1.000000e+00 | 1.902813e-15 | 9.482936e-34 |
| 10 | 1.000000e+00 | 1.111803e-18 | 2.724060e-38 |

Not surprisingly, the probabilities for *setosa* dominate the first few observations. We can plot the separation on the two dimensions (Figure 8.7):

```
> plot(linear)
```

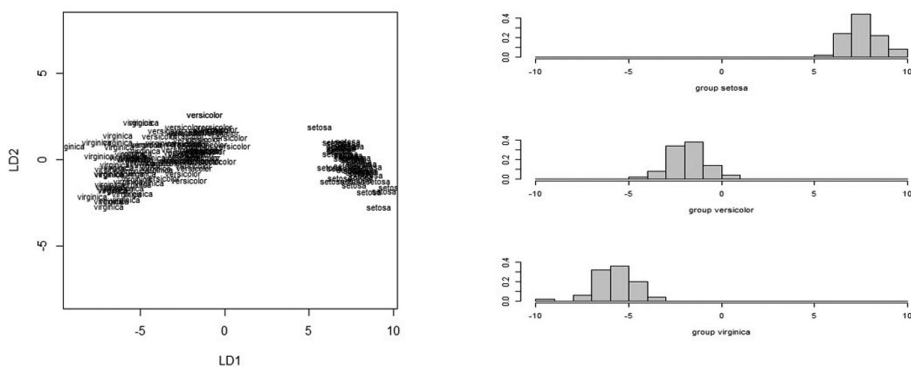


Figure 8.7 LD1 and LD2 separation on iris data.

```
> ldahist(data = p$x[, 1], g = iris$Species)
```

We see from Figure 8.7 that the linear discriminant analysis provides relatively good separation between classes. While there is substantial overlap between *versicolor* and *virginica*, species *setosa* is quite separated from these two.

### 8.15 Discriminant Analysis in Python

We can conduct a simple LDA in Python using *sklearn*, where in the following we contrive some data and make classifications on groups 1 vs. 2:

```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

We produce some data on X and y:

```
X = np.array([[-1, -1], [-5, -1], [-3, -6], [1, 1], [3, 1], [3, 2]])
y = np.array([1, 1, 1, 2, 2, 2])
```

X

```
Out [87]:
array([[ -1,  -1],
       [ -5,  -1],
       [ -3,  -6],
       [  1,   1],
       [  3,   1],
       [  3,   2]])
```

y

```
Out [88]: array([1, 1, 1, 2, 2, 2])
```

We now fit the discriminant analysis:

```
clf = LinearDiscriminantAnalysis()
clf.fit(X, y)
LinearDiscriminantAnalysis()
```

Let's predict group membership for values -0.9 and -2 on X:

```
print(clf.predict([[-0.9, -2]]))
[1]
```

We see from the above that for values on X = -0.9, -2, the predicted classification is in group 1. For `(clf.predict([5, 10]))`, we find the predicted class is 2.

### 8.16 Naïve Bayes: Approximating the Bayes Classifier by Assuming (Conditional) Independence

Recall that the purpose of discriminant analysis, as is true of many other classifiers, was to approximate the Bayes classifier. More pragmatically, its purpose was to classify populations based on a set of explanatory variables, but ideally, one would usually, pragmatics aside, use the Bayes classifier over the discriminant analysis model since it has the lowest theoretical error rate among all classifiers. That is, discriminant analysis is, at least theoretically, a second choice to the Bayesian ideal. However, we also said that LDA and QDA are not the only approaches available to approximate Bayes.

*Naïve Bayes*, as the name would suggest, is yet another method that attempts to approximate the Bayes classifier. As we will see, it accomplishes this approximation by making the simple assumption of *conditional independence*, one that is unlikely to ever be met in practice.

### 8.16.1 What Makes Naïve Bayes “Naïve?”

To understand and appreciate what makes Bayes “naïve,” we must first revisit the fact that when we conduct methods such as linear discriminant analysis, we assume a great deal when implementing the model. For instance, we assume that the density function is multivariate normal; we assume equal covariances in the case of linear discriminant analysis (not so in QDA). All of these assumptions make the task of fitting a model relatively “simple,” in the sense that we can cross many things off our “checklist” and assume them to be *constant* rather than *variable*. That is, by making assumptions in models, we can focus on the change we want to focus on and consider the other elements “fixed” and not fluctuating, analogous to how an experimentalist holds empirical conditions constant except for that which is under investigation, the so-called *experimental manipulation*.

As another example, linear regression is a great model under the assumption that the relationship is linear. How great are parametric assumptions in facilitating the task! They “prep the ground,” so to speak, so that we can take the next step and estimate the functional form. We are starting off with the bed half-made. Much of statistical modeling is, in actuality, as is true with most theories and narratives, *the specifying of assumptions a priori* (Shao, 2023).

This is where Naïve Bayes differs from methods such as linear discriminant analysis. With Naïve Bayes, we typically do not make assumptions about the parametric form, nor about parametric densities such as imposing a multivariate normal distribution. This is not to say we do not make *any* assumptions, but the assumptions we do make are rather limited. Here is what we usually assume with Naïve Bayes:

*Within the  $k^{\text{th}}$  class, the  $p$  predictors are independent.*

But what is the above, really? Recall the independence assumption from basic probability theory for events  $A$  and  $B$ ,

$$p(A \cap B) = p(A)p(B)$$

which reads that the probability of the joint event  $A$  and  $B$  is equal to the product of their unconditional probabilities. Recall that if the two events were not independent, then the joint event would involve a *conditional* probability:

$$p(A \cap B) = p(A)p(B / A)$$

Under independence of  $A$  and  $B$ , it stands that  $p(B) = p(B / A)$  and so we obtain,

$$\begin{aligned} p(A \cap B) &= p(A)p(B / A) \\ &= p(A)p(B) \end{aligned} \tag{8.39}$$

Equality (8.39) is also generally known as the *product rule for independent events* in probability theory. We will generalize this assumption to *densities* instead of variables, and state

$$f_k(x) = f_{k1}(x_1) \cdot f_{k2}(x_2) \cdots f_{kp}(x_p)$$

where the different densities are now indexed by  $k$  for  $p$  variables. Naïve Bayes assumes that for a given class, group or category, *predictors are independent* (Hastie et al., 2009). If we choose to incorporate the group or category directly in our statement of independence, then the statement of independence becomes that of *conditional independence*:

$$p(A, B / C) = p(A / C)p(B / C) \quad (8.40)$$

where  $C$  is now the class membership stated *explicitly* in the probability statement instead of needing to “attach” the membership statement in our earlier definition of conditional independence. That is, conditional independence simply generalizes the concept of independence to conditional probabilities. Though the Naïve Bayes assumption is technically about conditional independence, it makes more sense, as many other authors do, to speak about the assumption of *independence* rather than continually state *conditional independence*. That is, we make it implicit in this discussion that we are referring to independence *within a particular class*.

Now, stop and think for a moment – *Are two events  $A$  and  $B$  ever truly independent?* That is, can we realistically imagine any two *real* events to be completely independent? Probably not, no more than we can think of two events that are totally uncorrelated. All variables are correlated to *some* degree at least, just as pretty much all events are dependent upon one another to some degree, at least *probabilistically*. That your colleague found a correlation between his or her variables is *expected*. The *unexpected* result is finding a correlation of precisely 0. Likewise for probabilistic dependence, the probability of winning the lottery is likely *not* independent of the probability of a mouse living in your house. We often *assume* they are independent (and for good reason, pragmatically they *are*), but probabilistically they are (probably) not so, even if the ensuing probability is so remote to be trivial. *Correlations and probabilities are not necessarily reflective of empirical processes*, especially given that probabilities are typically measured on a continuous scale.

Even given the reality of empirical independence virtually never being satisfied, the *assumption* of independence will be a useful, pragmatic, and simplifying one to make, just as the assumptions we employed for things like LDA were useful (and simplifying) as well. Hence, Naïve Bayes does not totally jettison the making of assumptions, but one might say the assumptions it does make are rather *simple* and *easy*, such as that for conditional independence. Nevertheless, it is important to emphasize the presence of *some* assumption here, even if quite simple and basic. As has been emphasized throughout this book, very few procedures are devoid of *any* assumptions, even if they are advertised as making few. Even in nonparametric techniques we will usually find hidden assumptions, though they are typically much less in importance or influence than in their competing assumption-laden parametric counterparts.

The assumption of conditional independence in Naïve Bayes is made for the purpose of *utility*. We know in advance it will likely not be true. It is made for *convenience* and *pragmatism*, similar to the idea of assuming Gaussian distributions in LDA. Will Naïve Bayes estimate correctly? Probably not, and hence, it is *biased*, but with added bias comes a reduction in variance, similar to how using ridge regression allows for an increase in bias but with a reduction in variance. Recall once more the fundamental bias-variance trade-off:

$$E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + \left[ \text{Bias}(\hat{f}(x_0)) \right]^2 + \text{Var}(\varepsilon)$$

In the case of Naïve Bayes,  $\text{Var}(\hat{f}(x_0))$  typically decreases at the expense of an increase in bias,  $\left[ \text{Bias}(\hat{f}(x_0)) \right]^2$ .

Naïve Bayes is simple to understand because it is more easily understood as an extension of Bayes theorem, which once more recall is the following:

$$p(y = k \mid x = x_i) = \frac{\pi_k f_k(x_i)}{\sum_{l=1}^K \pi_l f_l(x_i)} \quad (8.41)$$

Since  $f_k(x_i) = f_{k1}(x_1) \cdot f_{k2}(x_2) \cdots f_{kp}(x_p)$  as a result of conditional independence, once we substitute this into (8.41) for Bayes, we obtain

$$p(y = k \mid x = x_i) = \frac{\pi_k f_{k1}(x_1) \cdot f_{k2}(x_2) \cdots f_{kp}(x_p)}{\sum_{l=1}^K \pi_l f_{l1}(x_1) \cdot f_{l2}(x_2) \cdots f_{lp}(x_p)} \quad (8.42)$$

where the product  $f_{k1}(x_1) \cdot f_{k2}(x_2) \cdots f_{kp}(x_p)$  denotes the independence (i.e., product of densities). Hence, we see that Naïve Bayes is simply a generalization of the classic Bayes formula, where within each class,  $k$  predictors are *assumed* independent and can, depending on the given research context, also in many cases conveniently be assumed to be drawn from a normal distribution with mean  $\mu_{jk}$  and variance  $\sigma_{jk}^2$ . As James et al. (2021, p. 155) remark,

While this may sound a bit like QDA, there is one key difference, in that here we are assuming that the predictors are *conditionally independent* [emphasis added]; this amounts to QDA with an additional assumption that the class-specific covariance matrix is diagonal.

A diagonal matrix is one with zeroes everywhere except the main diagonal, which in this case, implies the predictors will have a variance (along the main diagonal), but that covariances along the off-diagonal will equal 0. Recalling our covariance from our statistical review earlier for a  $3 \times 3$  matrix, the diagonal matrix would look like this:

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}$$

Hence, this is a key difference between QDA and Naïve Bayes. In QDA, nowhere did we assume predictors were uncorrelated or conditionally independent. In Naïve Bayes, we assume they are. Again, it's an *unrealistic assumption*, but one that, if made, allows us to process and fit this type of model.



### 8.16.2 Naïve Bayes in Python

*Scikit-learn* offers many kinds of Bayes classifiers, among them `GaussianNB`, `BernoulliNB`, and `MultinomialNB`. The following example is on the iris data featuring Gaussian Naïve Bayes (`GaussianNB`) adapted from *scikit-learn.org*:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X, y = load_iris(return_X_y=True)

X
Out[143]:
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],

y
Out[144]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

We now split the data into train and test sets. For now, we will keep the size of the test set at 0.5:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_
size=0.5, random_state=0)
```

We now run the Gaussian Naïve Bayes on the train data and evaluate how well the model does on the test data:

```
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points: %d"
% (X_test.shape[0], (y_test!= y_pred).sum()))
```

```
Number of mislabeled points out of a total 75 points: 4
```

We can see that out of 75 points, Naïve Bayes only gets four of them wrong, and hence the accuracy of prediction is approximately 94–95%. Depending on the research area or the task at hand, that number may be pretty good! We encourage the reader to explore with the above function by modifying the test size of the data as well as working with the many additional options available for Naïve Bayes featured on *scikit-learn.org*.

Naïve Bayes can be fit in R using the package *naivebayes* (Majka, 2024) as well as the *e1071* package (Meyer et al., 2023).

### 8.17 LDA, QDA, or Naïve Bayes: Which Is Best?

We conclude our discussion of classification methods surveyed in this chapter with a very brief comment comparing approaches. Recall that a major purpose of this book is to encourage the reader to situate statistical and machine learning models relative to one another, to “see” the *grand structure* so that one does not think that different methods necessarily imply wholly different technical structures. To this end, we wish the reader to be able to appreciate that many of the methods featured in this book are *special cases* of others, or at minimum, *relatable* to other techniques. We refer the reader to James et al. (2013, pp. 151–152) who provide a much more thorough discussion.

First, as we have already discussed, LDA can be considered a restricted version, or “special case” of QDA, one for which the equality of covariance matrices condition is imposed,

$$\Sigma_1 = \cdots = \Sigma_K = \Sigma$$

Second, we can conclude that many classifiers with a linear boundary can be considered, in a sense, as a special case of *Naïve Bayes*, which means that LDA can be considered a special case of Naïve Bayes. However, because QDA does not impose linearity, this correspondence does not hold for QDA. Naïve Bayes is additive only, where QDA and LDA can also include interaction terms, which is an important deciphering characteristic between these models.

So, which of these models is better than the other? Not surprisingly by now, it turns out that *none are universally better than the others*. As summarized by James et al. (2021, p. 160):

None of these methods uniformly dominate the others; in any setting, the choice of method will depend on the true distribution of the predictors in each of the  $K$  classes, as well as other considerations, such as the values of  $n$  and  $p$ . The latter ties into the bias-variance trade-off ... we expect LDA to outperform logistic regression when the normality assumption (approximately) holds, and we expect logistic regression to perform better when it does not.

### 8.18 Nonparametric K-Nearest Neighbors

*K-nearest neighbors* (KNN) takes a completely different approach from the parametric classifiers in this chapter (such as LDA) and is a completely nonparametric approach. That is, virtually no assumptions are made about the shape of the decision boundary whatsoever. KNN will likely dominate LDA and logistic in most circumstances when the decision boundary is relatively non-linear. Unlike logistic regression, KNN will not tell us which predictors are *important* in any sense as we will not get a table of coefficients as part of the output in KNN. We discuss KNN in more detail now.

#### 8.18.1 K-Nearest Neighbors (KNN): Only Looking at Nearby Points

The  $k$ -nearest-neighbor approach is often used for classification into one of two categories, though multiclass KNN is a possibility as well for cases in which there are more than two classes. What makes K-nearest neighbors nonparametric? KNN relies exclusively on the data to make its classification. It derives its name from the fact that it looks at *neighboring data* to determine which group the current observation should be classified to. KNN stores the training data and simply makes a

decision based on points “close” (in terms of distance) to the observation under consideration. For this reason, KNN is sometimes referred to as a *lazy learner*; “lazy,” because it does not bother to incorporate that much information in making its classification decision; it only looks at the nearby points and that is enough.

KNN is the prototype method of *local learning*, since it approximates the target function using only nearby data points. This also makes the technique an example of what is referred to as *instance-based learning* as it is not imposing a model as in *model-based learning* (e.g., linear regression). That is, it does not attempt to look at the “big picture” or to data points that are not in the *neighborhood*. Because KNN focuses only on *local* information, it can give a very misleading conclusion if outside of the neighborhood the more “global” structure is quite different than nearby. It would be akin to predicting the home price of a particular house based solely on a few of the nearby homes instead of the homes a bit further down the block.

As we have seen in this book, however, the above is not only true for KNN; it is true for virtually all methods that prioritize *local structure* over *global structure*. Looking more locally has advantages, but the cost is potentially ignoring the bigger picture. This can be good or bad, depending on how you look at it. To use our analogy of home prices, perhaps the house you are buying is worth the asking price because its “mini neighborhood” consists of mansions, even if its “extended neighborhood” features many less expensive houses. In the end, it’s a judgment call based on what is most important to the investigator.

KNN is also considered a *memory-based learner* because it focuses only on the training data and nearby points. What is remarkable about nearest-neighbor decision rules is that half of the classification information in an infinite sample set is contained in the nearest neighbor (Cover & Hart, 1967). That is, “this simple rule has a probability of error which is less than twice the Bayes probability of error, and hence is less than twice the probability of error of any other decision rule, nonparametric or otherwise, based on the infinite sample set.” (1967, pp. 21–22) For a theoretical justification for how nearest-neighbor methods converge on risk, see Cover and Hart (1967), who justify that half of the available information in an infinite collection of classified samples is contained in the nearest neighbor (p. 27).

As mentioned, such instance-based learning algorithms are sometimes referred to as “lazy” learning methods because they delay processing until a new instance must be classified. One advantage of this kind of delay is that instead of estimating the target function one time for the entire instance space, such methods can estimate it *locally* and differently for each new instance that needs to be classified. Recall that locally weighted regression is one type of method that accomplishes something similar to KNN. Not all learners that are instance-based are also lazy. For instance, support vector machines are considered instance-based yet are not considered a part of the class of lazy learners.

KNN can also be operationalized from a Bayesian perspective which incorporates class priors in the classification of new points based on a sphere of data denoted as the nearest neighbors. For details, see Hastie et al. (2009). The history of nearest-neighbor rules seems to go back to at least Fix and Hodges (1951), with further developments by Cover and Hart (1967).

### 8.18.2 Example of KNN

As a simple example of KNN, consider the illustration in Figure 8.8.

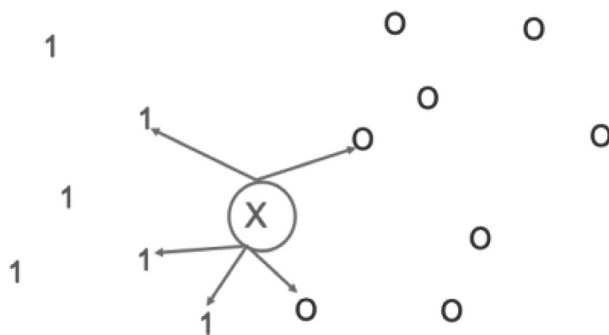


Figure 8.8 KNN will assign the “x” data point to group “1” since group 1 observations are its nearest neighbors.

The data point  $x$  will be classified to group “1” because its nearest neighbors are mostly values of 1. That is, there are 3 *ones* and 2 *zeroes*. KNN thus focuses on *local behavior* of nearby data points to determine classification. In running a KNN, we are required to select a value for  $k$ , which in this example is equal to 5, designating the nearby points to consider when making the classification. Of those 5 nearest neighbors, 3 of them have values of 1, and hence  $x$  is classified to group “1.” For cases that are equally as close to other points, random selection can be used to determine which is the “nearest neighbor” (Kroese et al., 2020).

If we set  $k = 3$ , for instance, we would have Figure 8.9.

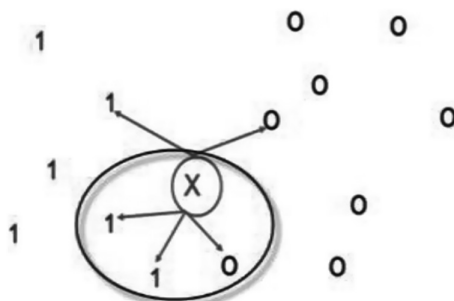


Figure 8.9 Example of KNN with  $k$  set to 3. The observation  $x$  will be classified to group 1.

With  $k$  set to 3, we can see that 2 out of 3 of the nearby objects have values of 1, and hence the new case  $x$  is classified once more to the group “1.”

Whereas least-squares assumes the underlying function  $f(x)$  is well-approximated by a global linear function, the  $k$ -nearest neighbors approach assumes  $f(x)$  is well-approximated by a *locally constant function* (Hastie et al., 2009). This fact is operationalized in KNN as the nearby points. While the KNN approach may appear more flexible, the sheer size of the dimensionality problem (i.e., the *curse of dimensionality*) makes assumption-laden approaches such as OLS regression often more attractive.

When  $k = 1$ , then each data point has a single nearest neighbor, and hence the classifier will fit very well; however, this will likely result in *overfitting*. Specifying  $k = 1$  results in what is known as a *Voronoi tessellation* of points (Figure 8.10), where each point has its own neighborhood of a single data point (think of the skin on a giraffe).

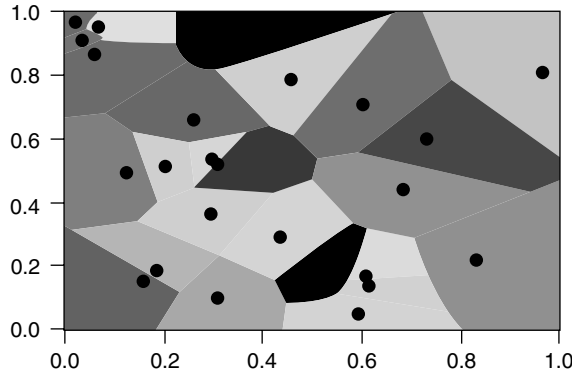


Figure 8.10 Voronoi tessellation of points. Each point has its own neighborhood based on its single nearest neighbor.

Clearly, setting  $k = 1$  is not very useful and will usually overfit. We seek instead to set  $k$  to a larger value for most problems. Incidentally, when  $k = 1$ , KNN is sometimes simply referred to as NN (“nearest neighbor”). What distinguishes KNN from NN is the option of specifying values for  $k$  greater than 1. Of course, if we set  $k$  equal to  $n$ , the total number of data points, then KNN will simply select the most common class of all data points.

The other requirement of KNN in addition to setting  $k$  is specifying a *distance metric*. Concluding that a data point is in the “neighborhood” is fine, but what denotes a “neighborhood?” New York City is in the neighborhood of Washington DC if we consider eastern cities only, but is not in the neighborhood if we adopt a different metric or different context. As is true of most statistical and machine learning methods, findings can be heavily influenced by how we choose to model things, and those decisions often come down to decisions such as the nature of the *metric* we choose to use.

The distance between two observations,  $d(x, x')$ , as we have seen throughout this book, can be measured in several ways. One very common, popular, and perhaps “ideal” way is that of the Mahalanobis distance:

$$d_M(x, \mu) = \sqrt{(x - \mu)^T \mathbf{M}(x - \mu)} \quad (8.43)$$

where  $\mathbf{M}$  is a positive definite matrix, typically the covariance matrix  $\mathbf{S}$ . The distance reduces to Euclidean if  $\mathbf{M} = \mathbf{I}$  and we obtain, simply,

$$\begin{aligned} d_E(x, \mu) &= \sqrt{(x - \mu)^T \mathbf{I}(x - \mu)} \\ &= \sqrt{(x - \mu)^T (x - \mu)} \end{aligned} \quad (8.44)$$

The default distance metric for KNN methods is that of Euclidean; however, KNN is actually compatible with a number of distance metrics. If using a more complex metric such as Mahalanobis, the computational expense increases significantly. It should be emphasized that like in multidimensional scaling, as we will later see, there is no “correct” choice of distance measure in KNN; sometimes a variety of measures need to be tried and sometimes the decision is more problem-dependent. Many of the distance measures used for KNN are those used in clustering algorithms and in addition to the classic Euclidean distance also include the *Manhattan* and *Minkowski* distances. The *Hamming distance* can also be used, which is simply the Manhattan distance applied to binary features.

To speed up KNN algorithms, a variety of so-called “pruning” techniques can also be used, such as removing outliers so the algorithm does not have to incorporate such extreme observations in determining its classification. This is sometimes referred to as *KNN editing*.

### 8.18.3 *Disadvantages of KNN*

At first glance, KNN can seem like an ideal and excellent classifier. It is simple, intuitive, and in actuality classifies relatively close to the ideal and most perfect classifier, which is Bayes. Remember, Bayes is best! Or once more, as Cover and Hart (1967) put it, “The Bayes risk serves merely as a reference – the limit of excellence beyond which it is not possible to go” (p. 23). That is, we cannot do “better” than the Bayes risk, and KNN does a reasonably good job at approximating the accuracy of this “ideal” classifier.

So why not use KNN all the time? The problem is that discussed at the outset of this book, which is the *curse of dimensionality*. Briefly, as the number of dimensions in the data increases, the “nearest neighbor” ends up being a point that is quite *distant*, and hence using such points to determine the classification of the current point will likely be deficient. In other words, that “distant” point is not *really* a “nearest neighbor” other than simply by *mathematical* definition. It would be like if your neighbor lived on Mars; they may still be your *nearest neighbor*, but they are a planet away! More technically, *the volume of the space we are working in grows exponentially with dimensionality* (Hastie et al., 2009), so unless we have the sample size to fill that volume of widening space, dimensionality will be a problem. This drawback then is one motivating factor for introducing a parametric model, that is, a model that comes “pre-packaged” with assumptions. The parametric model will make assumptions underlying the data so that the curse of dimensionality is less problematic (Murray, 2022).

*KNN, at first glance and due to its simplicity, can appear to be an excellent classifier. However, due to the curse of dimensionality, the “nearest neighbor” will often end up being a point that is quite distant. The solution to this problem, aside from possibly greatly increasing sample size, is often to introduce a parametric model, that is, one that comes with it prior assumptions about the form of the relationship.*

Another possibility and solution to the problem of dimensionality is to simply locate a smaller set of the observations in higher dimensions and make classification decisions more *locally*. We can easily see as well that the nearest-neighbor approach is quite sensitive to random perturbations in the data. That is, it has *high variance*. This can easily be appreciated by inspecting the aforementioned *Voronoi tessellation* of points (Figure 8.10). Note that by moving just a single point significantly, we can drastically change the classification status for that point, since its new nearest neighbor now is a different neighbor. The KNN method often has more variance associated with it than it has bias in the classic *bias-variance trade-off* paradigm.

How can we make data points a bit less sensitive to nearby data points? We can do so by regularizing the approach, which in this case means simply introducing more neighbors, that is, increasing  $k$ . For example, increasing  $k = 2$  to  $k = 5$  means that instead of looking at the closest 2 nearest neighbors, we will instead consider the closest 5. It is plausible that this will reduce variance but at an increase in bias. In this case, we are essentially *smoothing* our classification process.

Computationally, KNN also needs to store all of the data so that it can draw on such data to make each classification. As Murray (2022) remarks, this is not a very efficient use of memory, and hence the KNN approach for large sample size can be relatively slow. Consequently, KNN is not best used when you have a lot of features (Müller & Guido, 2017). An alternative approach, which is still nonparametric, but which overcomes this memory issue, is that of the *sparse kernel machine*, which only considers a subset of the data in making its classification decisions. This technique also capitalizes on the kernel trick, which is a nonparametric approach.

KNN can be performed in Python using `KNeighborsClassifier`. The packages *neighbr* (Bolotov, 2020) and *caret* (Kuhn et al., 2024) can be used to perform  $k$ -nearest neighbors. The *caret* package includes cross-validation techniques for selecting the best number for  $k$  to maximize accuracy.

### 8.19 Chapter Review Exercises

1. From a *scientific* (rather than *statistical*) perspective, how might operationalizing the response variable differently potentially lead to variable substantive conclusions? Explain.
2. What is a *generalized linear model*, and how does it differ from a *general linear model*? Which is the more inclusive model?
3. Explain, using an example, the nature of *odds*. If the odds are 1 to 1, what does that mean?
4. Discuss the difference between an *odds* and an *odds ratio*.
5. Suppose the coefficient for a logistic regression is equal to 0.20. Give a precise interpretation of the coefficient.
6. Compare a *generalized additive model* to that of a *generalized linear model*. How are they similar? Different?
7. Drawing on the discussion in the chapter, review the many ways a relationship between a response and predictor can be modeled, starting out with linear regression and ending with the generalized additive model. Your complete discussion should feature twelve different models.
8. Distinguish between *linear discriminant analysis*, *quadratic discriminant analysis*, and *regularized discriminant analysis*. How are they similar? Different?
9. Conduct a quadratic discriminant analysis on the iris data using all variables (or “features”) to classify species *setosa* from *versicolor*.
10. Perform a linear discriminant analysis in Python on the same data featured in exercise 9.
11. Is linear discriminant analysis a special case of *Naïve Bayes*? Why or why not? Explain.
12. Using the package *naïvebayes* in R, perform an analysis predicting classes *setosa* and *virginica* based on all features.

## 9 Support Vector Machines

In the previous chapter, we surveyed the method of linear discriminant analysis as a popular method for classifying a linear combination of input features into two or more mutually exclusive classes or groups. The *linear discriminant function* is only one of many methods that generates a hyperplane with the goal of classification. “Hyperplane” here, in two dimensions, is synonymous with *linear*.

In the current chapter, we study another classifier, but capable of more generality than the classic discriminant function. The *support vector machine* allows one to not only classify linear boundaries but also *nonlinear* ones by essentially constructing a linear boundary in a transformed version of the feature space (Hastie et al., 2009). Modern support vector machines were developed by Cortes and Vapnik in the 1990s and are quite sophisticated in the way they go about discriminating. They are a special case of what are generally known as *kernel machines*. Kernel machines, as the name suggests, make heavy use of the *kernel operator*, usually in the form of what is known as the *kernel trick* in statistics and machine learning.

Entire books and numerous papers are written on support vector machines and kernel methods more generally. In this chapter, we highlight their main features and emphasize what differentiates them from more classical techniques. The full support vector machine is a generalization of what is called the *maximum margin classifier*, and hence before studying the SVM (*support vector machine*), we need to discuss the nature of a maximal margin classifier and briefly survey its properties. This will open the door to SVM, which will allow for the classification of nonlinear boundaries.

### 9.1 Maximal Margin Classifier

We begin with the definition of a hyperplane. A *hyperplane* is a flat affine subspace of dimension  $p - 1$ , given by the following:

$$f(x) = x^T \beta + \beta_0 = 0 \tag{9.1}$$

which in the general case allows for as many dimensions as we wish. We typically set the length of  $\beta$  to unity, so that  $\|\beta\| = 1$ . For instance, if we let  $\beta = 1$  and  $\beta_0 = 0$ , we will obtain the line in Figure 9.1 that passes through the origin.



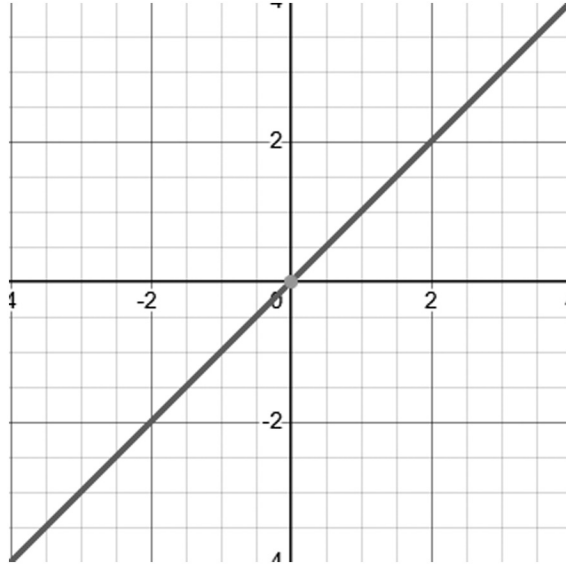


Figure 9.1 Line that passes through the origin having  $\beta = 1$  and  $\beta_0 = 0$ .

In the case where we have two dimensions, (9.1) becomes

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0 \quad (9.2)$$

Any  $x = (x_1, x_2)^T$  for which  $f(x)$  is true is a point on the hyperplane. In  $p$  dimensions, we can extend  $f(x)$  to be

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0 \quad (9.3)$$

Again, consider that we can specify any  $x = (x_1, x_2, \dots, x_p)^T$  for which

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0 \quad (9.4)$$

holds. All of these points, for  $x_1, x_2, \dots, x_p$  fall on the hyperplane, which is why the sum is equal to 0. Any points for  $x_1, x_2, \dots, x_p$  not falling on the hyperplane will fall on either side of it.

### 9.1.1 When Sum Does Not Equal Zero

Consider the case now where  $\beta_0 + \beta_1 x_1 + \beta_2 x_2, \dots, \beta_p x_p \neq 0$ . What are some possibilities? The sum could be *larger* than 0, or *smaller*, as we note below for the multidimensional case:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p > 0$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p < 0$$

The fact that a point can fall on either side of the hyperplane implies we have two classes, which we can enumerate as  $\{-1, 1\}$ . We can now define a separating hyperplane for any given observation,  $x_i$ , with the following specific membership categories, now including  $y_i = 1$  for one class and  $y_i = -1$  for the other,

$$\begin{aligned}\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} &> 0, y_i = 1 \\ \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} &< 0, y_i = -1\end{aligned}$$

The above naturally implies the following property for the hyperplane,

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0 \quad (9.5)$$

Hence, what we have established is a basic rule by which to classify new observations,  $x_i$ . Further, the *size* of the deviation greater than or less than 0 can be used to assess the magnitude of the discrepancy away from the hyperplane. For example, though the result

$$\begin{aligned}\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} &> 0, y_i = 1 \\ 10 &> 0, y_i = 1\end{aligned}$$

yields the same classification as

$$\begin{aligned}\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} &> 0, y_i = 1 \\ 2 &> 0, y_i = 1\end{aligned}$$

the *magnitudes* of the size of discrepancy, of course, differ greatly. We have more *confidence*, in a sense, in the first classification than in the second due to the magnitude of the deviation (i.e., 10 vs. 2).

#### 9.1.1.1 *So, What's the Problem?*

At first glance, the above seems to be a simple matter of how to classify with observations,  $x_i$ , based on the establishment of a hyperplane. The problem, however, is that estimating the hyperplane poses a challenge since if our data can be well-separated by a hyperplane, then there is an infinite number of such hyperplanes, as shown in Figure 9.2.

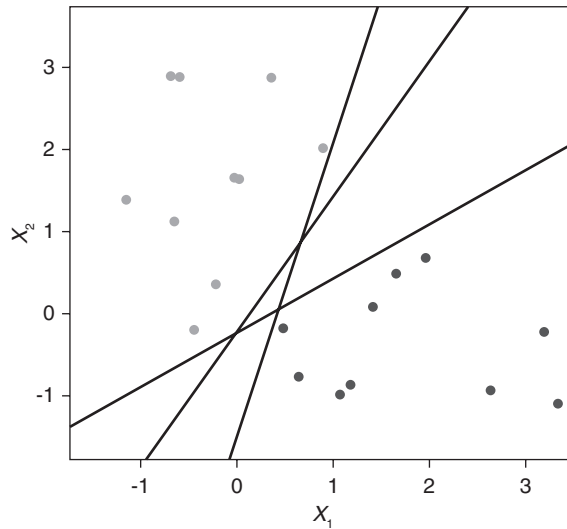


Figure 9.2 Data that are linearly separable by an infinite number of hyperplanes.

Notice that all the drawn hyperplanes successfully separate the binary categories. The problem of the *maximal margin classifier* boils down to the following question:

*Which of the infinite hyperplanes should we choose?*

Does this problem sound familiar? Of course it does! A similar problem posed itself when we considered earlier “classical” procedures in this book, such as *linear regression*, *discriminant analysis*, and even *principal components analysis*. What an opportune time to present a unifying theme! In all of these methods, the goal was to utilize a hyperplane based on some optimization criteria, such as minimizing or maximizing some function. Recall that in regression analysis, ordinary-least-squares required us to minimize the sum of squared errors. In discriminant analysis, the job was to maximize group differences. In principal components, the goal (as we will see later) is to maximize perpendicular distances. Now we come to yet another situation where we need to define the optimization criteria for a new approach. The question thus becomes:

*What is the best way to choose a hyperplane such that it maximizes classification?*

Is the above question novel in any way? Absolutely not. What we are about to survey with the maximum margin classifier and support vector machine *is* new, but the question these techniques seek to answer is definitely not. That is, the *concept* of hyperplanes that are optimized by some criteria is the same question we asked in least-squares regression, discriminant analysis, and other techniques. In this chapter, we simply survey a novel way of classification based on what we may more generally call *the hyperplane problem*.

The answer to this question is that we will choose the *maximal margin hyperplane*, also known as the *optimal separating hyperplane*, which, in the sense we will define shortly, is the hyperplane *farthest* from the training observations. That is, *it is the hyperplane that maximizes the margin of the data to the hyperplane*. Of course, this definition is vague and we need to define its criteria more precisely. With a single point on either side of a hyperplane, maximizing the margin would simply equate to maximizing the distance between the hyperplane and the points, as shown in Figure 9.3.

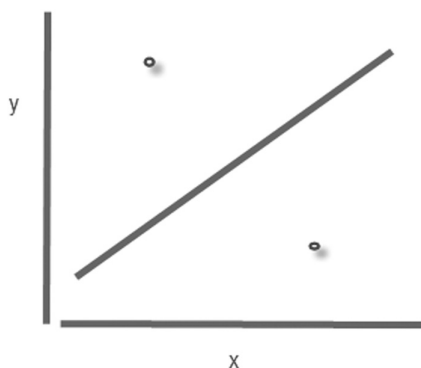


Figure 9.3 A hyperplane maximizing the distance between two observations.

The difficulty, of course, occurs when our data consist of more than two points, such as when we have many observations as in Figure 9.4.

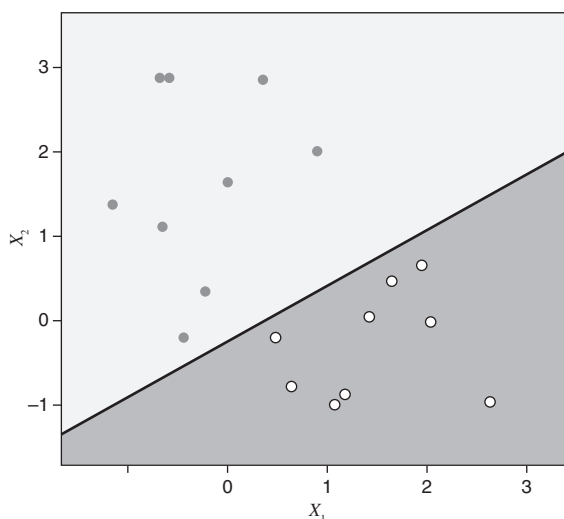


Figure 9.4 Hyperplane separating two groups consisting of several data points, introducing the problem of deciding which points should have the most influence in deciding the placement of the hyperplane.

The question becomes how to best determine a “maximal margin” since there are so many competing points and we cannot simply compute a Euclidean distance in best separating two points. We will operationalize in a moment what is described best by James et al. (2021, p. 341):

We compute the perpendicular distance from each training observation to a given separating hyperplane; the smallest such distance is the minimal distance from the observations to the hyperplane, and is known as the margin. The maximal margin hyperplane is the separating hyperplane for which the margin is largest, that is, it is the hyperplane that has the farthest minimum distance to the training observations ... In a sense, the maximal margin hyperplane represents the mid-line of the widest “slab” that we can insert between the two classes.

Figure 9.5 is a depiction of the “widest slab” generated by the maximal margin classifier.

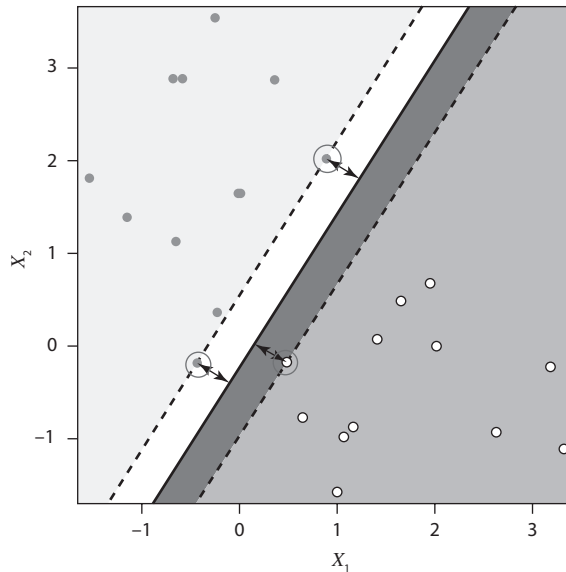


Figure 9.5 Depiction of the “widest slab” concept in determining support vectors.

Notice that the placement of the hyperplane generates the widest margin. The points, or vectors that determine this widest margin are called *support vectors* since they determine the position of the hyperplane; that is, they “support” the placement of the maximal margin hyperplane. *Movement of the support vectors will have the effect of moving the hyperplane.* Hence, the support vectors themselves define the placement of the hyperplane and are thus extremely important in determining its location. *No other points in the plot are actually determining the placement of the hyperplane.*

The fact that only the support vectors determine the location of the hyperplane, and no other points, is of great consequence. Consider for a moment how this differs from ordinary least squares (OLS) regression or discriminant analysis. In both of those methods, *all points in the training data contributed in determining the placement of the hyperplane.* The concept of support vectors, however, relies on only *a few points or vectors.* We can already tentatively conclude then that hyperplanes determined by support vectors are more *resistant* to outlying data points than hyperplanes that “average” the data in some sense. To illustrate the point, consider the plot in Figure 9.6, specifically the circled points.

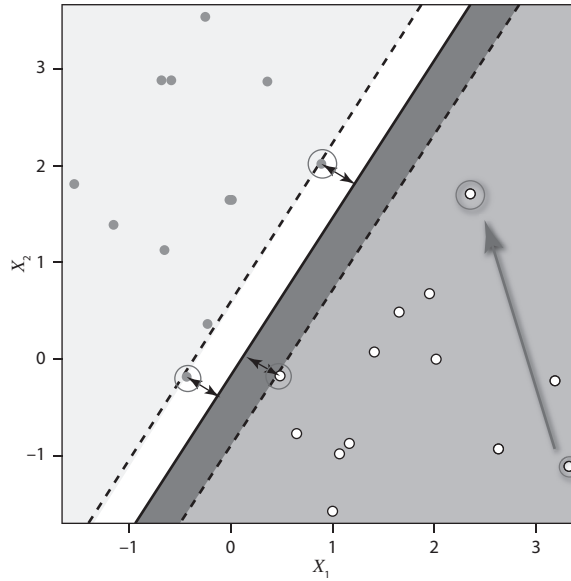


Figure 9.6 Movement of a single data point (represented by the arrow) does not influence the placement of the hyperplane unless that data point is a support vector.

Notice that the movement of the point circled from the lower right to a higher location has had zero effect on the hyperplane. If the hyperplane were defined more in the sense of OLS regression or discriminant analysis, the hyperplane should necessarily *shift* due to the shifting point. The maximum margin classifier, however, is insensitive or *robust* to any points not going by the name *support vectors*. This is not to suggest outliers cannot have a significant influence on the *placement* of the maximal margin classifier; they surely can. However, once the support vectors are determined, a change in position of any other points will not move the placement of the maximum margin classifier *unless such has an impact on the placement of the support vectors*.

### 9.1.2 Building the Maximal Margin Classifier

To reiterate, when we built the least-squares regression line or the discriminant function, both respective hyperplanes in their own right, we *maximized* or *minimized* some function of the data in a basic problem of optimization. In building the maximum margin classifier, we must likewise estimate parameters, once more for  $\beta_0$  and  $\beta_1$  through  $\beta_p$  just as we did in regression and discriminant analysis. However, in doing so, we will impose some constraints. We will wish to maximize the margin  $M$  subject to the following conditions or constraints,

$$\sum_{j=1}^p \beta_j^2 = 1 \quad (9.6)$$

and,

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \quad (9.7)$$

where  $M = 1/\|\beta\|$ . The solution can be obtained via *convex optimization* (Hastie et al., 2009). As the reader may recall from earlier in the book, convex optimization is simply the process of optimizing a function that has a *bowl-like* shape, which implies that finding a global minimum is achievable without the algorithm getting “stuck” in local minima. It is worth asking, however, what the above constraints guarantee. The two constraints assure us that each observation  $y_i$  will be on the correct side of the hyperplane, so long as the margin,  $M$ , is positive (James et al., 2021). The maximal margin classifier is a useful and efficient way to perform classification if a separating hyperplane exists, that is, if we can place a hyperplane such that both classes are completely separated. If we cannot do this, such that the best hyperplane we could draw would not separate points completely, then we need to build on the maximum margin idea but allow for some flexibility. We consider this possibility next.

## 9.2 The Case of No Separating Hyperplane

There are two situations to consider here. First, suppose a separating hyperplane does exist. Even if it does, it can be tremendously sensitive to movement of one or more of the support vectors, as depicted in Figure 9.7.

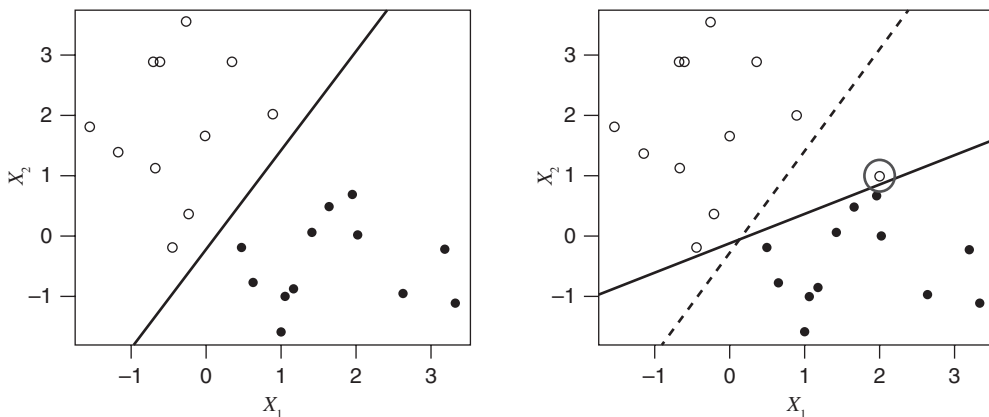


Figure 9.7 Movement of a single point (circled) can dramatically shift the location of the maximum margin classifier.

Observe the movement of the single circled point and notice how it dramatically shifts the maximum margin classifier. Can we tolerate such a shift of the hyperplane? Yes, we can, but do we really want a single point or vector to determine the location of the hyperplane to this extent? Probably not. We may be better off with a hyperplane that is not so much influenced by a point or two in determining its location. But to effectuate this compromise, we will need to require our separating hyperplane to not classify *perfectly*. That is, we may be better off with a hyperplane that classifies *most* of the observations correctly instead of *all* the observations, while still allowing some error for points or vectors that are not on the correct side of the hyperplane. As we will see, this is one motivating factor for developing the *support vector classifier* and preferring it over the maximum margin classifier.

There is a second reason for wanting to do better than the maximum margin classifier. Earlier we noted that even if a vector or two dramatically shifted the hyperplane, these did not by themselves *negate* the separating hyperplane. The hyperplane was affected by the movement of the

vectors, but it could still be *defined* on the data. However, *what if no separating hyperplane exists?* The solution for a problem like this is to extend the concept of a separating hyperplane to one that *almost* separates the classes. The *support vector classifier* generalizes the maximal margin classifier to the non-separable case. Hence, note carefully that we started with the maximal margin classifier for the linearly separable boundary; in the linearly non-separable case, the support vector classifier is what we wish to construct for data that look similar to that in Figure 9.8, as one possibility.

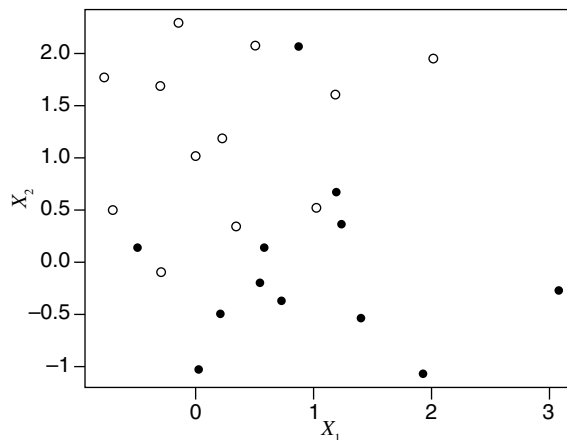


*Figure 9.8* Data on which there is no clear separating hyperplane. That is, we cannot fit a hyperplane that will completely separate the classes. Used with permission of Springer Nature BV, from *An Introduction to Statistical Learning* by Gareth James et al, © 2013; permission conveyed through Copyright Clearance Center, Inc.

It is easy to see in Figure 9.8 that a simple hyperplane will not be sufficient in separating the classes since there is clearly no separating hyperplane that can exist with this data. The solution to this problem will be to construct the *support vector classifier*. The support vector classifier will seek to classify most of the observations correctly but will allow *imperfect classification* such that it is not as *rigid* as the maximum margin classifier. As a comparison, recall that linear discriminant analysis (LDA) featured a relatively “lax” hyperplane as well, but in a different sense. In LDA, we sought to maximize between-group variance relative to within-group variance, which, unless the data were “perfect,” implied that some of our classifications would also be in error, which we could easily observe by computing a confusion matrix, for instance.

### 9.3 Support Vector Classifier for the Non-Separable Case

The non-separable case operates by *almost* separating the two classes via what is known as a *soft margin*. This classifier will attempt to classify *most* of the training data. That is, it allows for some observations to be on the wrong side under the condition that it is still classifying most of the observations correctly. This is indeed why it is called a *soft margin* since, though it establishes a margin, it is not “too” strict about it, analogous to a parent who establishes ground rules for the house but allows a bit of deviation away from those rules from time to time. Soft margins have benefits over hard margins, however, as most of the time they will still get things right; that is, they will still, in general, classify correctly.



To appreciate how the support vector classifier is operationalized, we return once more to the maximum margin classifier discussed earlier. Recall its optimization criterion,

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M$$

The optimization criterion of the support vector classifier is similar,

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \varepsilon_i) \quad (9.8)$$

What has changed? We note the addition of  $(1 - \varepsilon_i)$  on the right-hand side, so that instead of the criterion being  $\geq M$  it is now  $M(1 - \varepsilon_i)$ , where  $\varepsilon_i$  is referred to as a *slack variable*. The slack variable is what gives the soft margin its flexibility and non-rigidity when compared to the hard margin. For  $\varepsilon_i = 0$ , there is no slack, and the  $i^{\text{th}}$  observation is on the correct side of the margin, and the support vector classifier reduces to the maximum margin classifier featured earlier as can easily be seen by substituting  $\varepsilon_i = 0$ :

$$\begin{aligned} y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) &\geq M(1 - \varepsilon_i) \\ y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) &\geq M(1 - 0) \\ y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) &\geq M \end{aligned}$$

Of course,  $\varepsilon_i = 0$  is not the reason why the support vector classifier was established. It was motivated for cases in which  $\varepsilon_i > 0$ , since such implies the observation is on the wrong side of the margin, and if  $\varepsilon_i > 1$ , then on the wrong side of the hyperplane (Bishop, 2006; Izenman, 2008). The value of  $\varepsilon_i$  here is thus very important, since it alone determines the degree of *flexibility* we would like to establish in our classifier both for the case in which  $\varepsilon_i = 0$  (in which the case is simplified to the maximum margin classifier) and  $\varepsilon_i \neq 0$  for which we need to “regulate” the degree of slack that we will allow. Hence, this becomes a *regularization* problem, and we introduce the tuning parameter  $C$  to regulate the extent to which we will allow  $\varepsilon_i$  to deviate from 0. We will require  $\varepsilon_i \geq 0$  such that the sum of all  $\varepsilon_i$  is less than some constant,  $C$ ,

$$\sum_{i=1}^n \varepsilon_i \leq C \quad (9.9)$$

Similar to how we conceptualized the tuning parameter in methods such as ridge and lasso regression, the number  $C$  above provides a kind of “budget” for how much of a violation we will tolerate as it relates to our soft margin (Izenman, 2008). It is thus a *tunable hyperparameter*. We could have also required a different constraint, that instead of  $M(1 - \varepsilon_i)$  on the right-hand side, we could, at least in theory, choose  $(M - \varepsilon_i)$  such that our criterion now becomes

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M - \varepsilon_i \quad (9.10)$$

However, as Hastie et al. (2009, p. 419) note, in addition to the two choices leading to disparate conclusions, requiring  $M(1 - \varepsilon_i)$  results in a *convex* optimization problem, whereas  $(M - \varepsilon_i)$  is *nonconvex*, and so  $M(1 - \varepsilon_i)$  is chosen over  $M - \varepsilon_i$  primarily based on this criterion. Recall that optimization of a convex loss function is unaffected by local minima, which is what in general makes a convex solution desirable over a nonconvex one. This is distinct, for instance, from neural

networks in which loss functions are typically nonconvex and hence renown for getting stuck in local minima. This gives support vector machines a strong competitive advantage over neural network methods as well as decision trees (Izenman, 2008).

How should we choose  $C$ ? In practice we could take a reasonable guess at it, but it would probably be wiser to choose it via *cross-validation*. Note as well that as  $C$  increases, the margin becomes increasingly wider which translates to many observations violating the margin. Such a classifier would have relatively low variance but likely high bias. On the opposite extreme, if  $C$  is small, this would imply the classifier would have low bias but relatively high variance. The selection of  $C$  should be a compromise between these two extremes. Further details on computations involved in the support vector classifier can be found in Hastie et al. (2009, p. 420) which involves re-expression of the above identities and constraints for computational efficiency and convenience.

#### 9.4 Support Vector Machines: Introducing the Kernel for Nonlinearity

Up to now, we have discussed the maximum margin classifier and the support vector classifier. The maximum margin classifier was useful when there was a clear separating hyperplane that clearly divided two groups. The support vector classifier was a bit more flexible in that it allowed observations to be on the other side of the hyperplane; in essence, it built in a bit of “slack” (via *slack variables*) so that even if perfect separation did not exist, the support vector classifier attempted to fit a hyperplane anyway that would separate *most* observations, even if it missed a few. The support vector classifier also featured a tuning parameter  $C$  that could be chosen possibly via cross-validation to determine the amount of “slack” one wished to incorporate into the classifier. *The commonality between the maximum margin classifier and the support vector classifier is that the decision boundary is linear.* This point is essential, because nowhere in our discussion thus far have we entertained the idea of a *nonlinear* decision boundary. Even if our data up to now suggested the boundary may be nonlinear, we still, at minimum, imposed a linear boundary, possibly with allowing some slack in the form of  $\varepsilon_i$ .

Now we consider the case where the decision boundary is definitely *not linear*, which will require us to “evolve” our previous classifiers to something new, the full *support vector machine*, of which both the maximum margin classifier and support vector classifiers can be regarded as special cases. Hence, to summarize, our progression has been as follows, from left to right:

$\text{maximum margin classifier} \rightarrow \text{support vector classifier} \rightarrow \text{support vector machine}$   
 $(\text{linear}) \qquad \qquad \qquad (\text{linear}) \qquad \qquad \qquad (\text{nonlinear})$

The question of interest then is how to produce a classifier with a nonlinear decision boundary. What does a nonlinear boundary look like? Consider the plots in Figure 9.9.

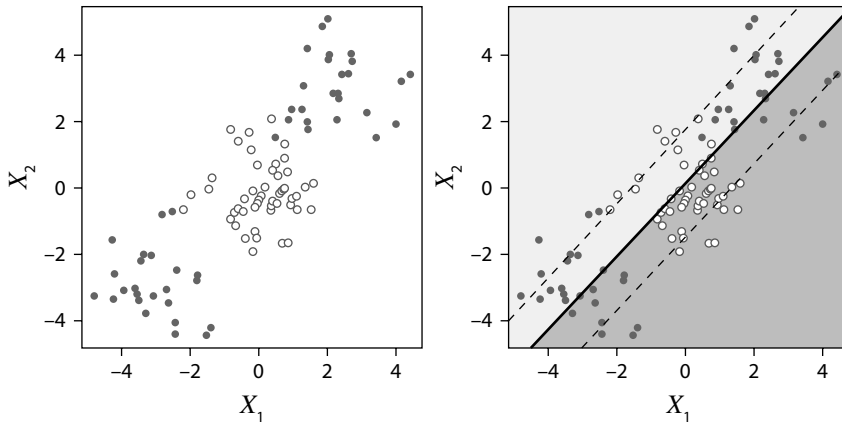


*Figure 9.9* Data featuring a nonlinear decision boundary. The maximum margin classifier and the support vector classifier will be inadequate in generating a decision boundary because the decision boundary is clearly nonlinear. Used with permission of Springer Nature BV, from *An Introduction to Statistical Learning* by Gareth James et al, © 2013; permission conveyed through Copyright Clearance Center, Inc.

We see in Figure 9.9 that, clearly, the decision boundary is not anywhere close to linear. Both the maximum margin and support vector classifiers will be inadequate in this situation. Discriminant analysis will not work either since it also imposes a hyperplane on the data, which is still linear. In fact, if we were to sketch the ideal decision boundary simply by hand, it would probably look something like that in Figure 9.10.

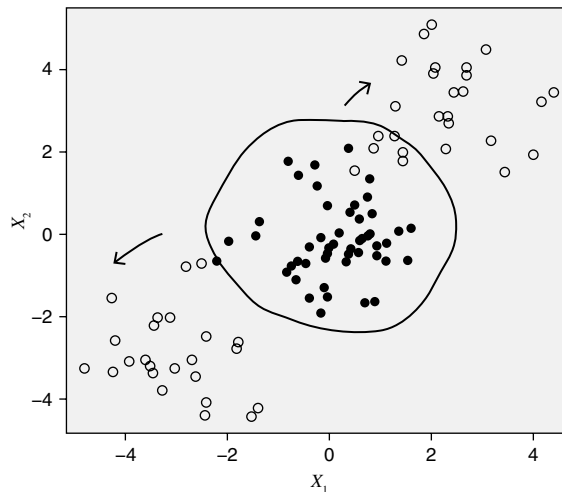


*Figure 9.10* “Ideal” crude nonlinear decision boundary for data that are not linearly separable. Used with permission of Springer Nature BV, from *An Introduction to Statistical Learning* by Gareth James et al, © 2013; permission conveyed through Copyright Clearance Center, Inc.

We drew the above by hand as an approximation only, but the message is clear; for such data, we are going to require a classifier that builds a *nonlinear boundary*. Allowing “slack” alone will clearly not solve the problem because there is no way to even impose a linear boundary. *No hyper-plane will get the job done*. The solution to creating a nonlinear boundary, as we discuss now, is what really put support vectors on the map historically as a machine learning method and that is as a general methodology for dealing with nonlinear boundaries. The solution is to enlarge the feature space by *kernelizing* the problem.

#### 9.4.1 *Enlarging the Feature Space with Kernels*

Cortes and Vapnik (1995, p. 271) describe their contribution of the support vector machine as follows:

The support-vector network implements the following idea: it maps the input vectors into some high dimensional feature space  $Z$  through some non-linear mapping chosen a priori. In this space a linear decision surface is constructed with special properties that ensure high generalization ability of the network.

The key to the above is that of *enlarging the feature space*. But what does that mean? Essentially, it means introducing more complex *basis functions*. As a reasonable attempt, we might try to take higher powers and terms of polynomials, such as the following,

$$x_1, x_1^2, x_2, x_2^2, \dots, x_p, x_p^2$$

or even more complex interaction terms, etc. *When we enlarge the feature space, in this case taking new basis functions such as powers and polynomials, the decision boundary often becomes separable by a hyperplane.*

We might ask then whether there is a *general approach*, aside from trying different polynomials, to enlarging the feature space that will also lead to efficient computations. It turns out there is, one for which we are already familiar, and involves computing *kernels* in the data or variable space instead of directly choosing among a variety of basis functions such as polynomials. What is more, *knowledge of inner products of observations is often just as useful as knowing the observations themselves*. These results are significant, if not rather extraordinary and extremely convenient, because they suggest we can use kernels in the original data or variable space to account for *potentially nonlinear feature spaces* instead of attempting to take polynomials and higher powers in the original space.

As we have seen earlier with regard to kernels, when we *kernelize* data in the input space, such says something about the similarity of observations in a higher-dimensional space by way of inner products in that higher-dimensional feature space. That is, *taking the kernel  $K(x_i, x_{i'})$  of observations  $x_i, x_{i'}$  in the input space essentially makes the nonlinear problem in data space now potentially linear in the higher-dimensional feature space*. Said differently, *a kernel quantifies the similarity between two observations and, in generalizing from an inner product to a kernel function, essentially “hides away” the nonlinear feature map* (Deisenroth et al., 2020).

As already discussed elsewhere in this book, there are many kinds of kernels, but they all essentially seek to account for *similarity* in some sense. The most obvious case for taking the kernel is to use a *linear kernel*:

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j} \quad (9.11)$$

Notice that the linear kernel is the same as simply taking the inner product on its own, and it can be shown that the above plays a key role in the Pearson correlation coefficient between observations. This isn't surprising since recall that the numerator of Pearson, in part, features

$$\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

and will be easily recognized as an inner product of deviations from respective means.

Of course, computing the linear kernel is not very climactic. The generality, however, of taking kernels is the option of being able to try out a variety of kernels (e.g., *polynomial kernel*, *radial basis function kernel*, *sigmoid kernel*, etc.), not only the linear kernel (Cortes & Vapnik, 1995).

Kernels are not in any way new in linear algebra. Much of the theoretical work on kernels in mathematics took place in the early 1900s. Kernel functions as inner products were first introduced into machine learning by Aizerman et al. (1964) and were first applied to support vector machines by Cortes and Vapnik (1995). However, it should be emphasized that the kernel property itself is in no way *unique* to support vector machines, as Hastie et al. (2009, p. 424) emphasize:

In the early literature on support vectors, there were claims that the kernel property of the support vector machine is unique to it and allows one to finesse the curse of dimensionality. Neither of these claims is true.

Hence, as we have noted elsewhere in the book, kernels can be applied quite widely and generally to different methods. Support vector machines do not have a “monopoly” on their utility or application. Many statistical and machine learning methods can be *kernelized* and kernels are undoubtedly one of the hallmarks of modern machine learning in general.

#### 9.4.2 Support Vector Machines in Python

As we have discussed in this chapter, a support vector machine can be used for a linear boundary or one that is nonlinear. The following is sample code from *sklearn* for a simple support vector machine (*classifier*) in Python:

```
from sklearn import svm
```

We will next define some data on X:

```
X = [[0, 0], [1, 1]]
X
Out[41]: [[0, 0], [1, 1]]
```

We next define the categories on the response variable:

```
y = [0, 1]
```

Note that we could have used any values for y, it was not necessary to use 0, 1. Next, we fit the SVM using a *radial basis function* (`kernel = "rbf"`) with a regularization penalty of `C = 100`:

```
clf = svm.SVC(kernel = "rbf", C=100)
clf.fit(X, y)
Out[44]: SVC()
```

Now, for values on  $X$ , we ask the support vector classifier (SVC) to make predictions on the response class:

```
clf.predict([[2., 2.]])
Out[45]: array([1])

clf.predict([[2., 0.]])
Out[46]: array([1])

clf.predict([[0., 0.]])
Out[47]: array([0])
```

We can see the predicted group membership for 2, 2, is 1, for 2, 0 is 1, and for 0, 0 is 0. We can identify the actual support vectors through `clf.support_vectors_`:

```
clf.support_vectors_
Out[160]:
array([[0., 0.],
       [1., 1.]])
```

For boundaries that are linear, one can also use `LinearSVC`.

### 9.4.3 *Support Vector Machines in R*

We can fit a support vector machine in R using the *e1071* package (Meyer et al., 2023). We adopt and explain the code from the package on the iris data to demonstrate the application of a simple support vector machine:

```
> library(e1071)
> data(iris)
> attach(iris)
> x <- subset(iris, select = -Species)
> y <- Species
```

Note that we have designated species to have its own  $y$  vector and removed species from the iris data (`select = -Species`) calling the new vector  $x$ . We now run the model (`scale = FALSE` if you do not wish to standardize):

```
> model <- svm(Species ~., data = iris, scale = TRUE)
> model
```

```
Call:
svm(formula = Species ~ ., data = iris)
```

```
Parameters:
  SVM-Type   : C-classification
  SVM-Kernel : radial
  cost       : 1
```

Number of Support Vectors: 51

We see from the above that the kernel used, by default, is the *radial kernel*, and that the number of support vectors is equal to 51. To specify the kernel directly,

```
> model <- svm(Species ~., data = iris, kernel = "radial")
```

We can try different kernels. Here are two more possibilities (output not shown):

```
> model <- svm(Species ~., data = iris, kernel = "sigmoid")
> model <- svm(Species ~., data = iris, kernel = "polynomial")
```

We now obtain a more complete summary of the model (for `kernel = "radial"`), which confirms that the number of classes we are trying to predict is equal to 3, and just above it is the distribution of the support vectors (8 for *setosa*, 22 for *versicolor*, and 21 for *virginica*):

```
> summary(model)
Call:
svm(formula = Species ~ ., data = iris)

Parameters:
  SVM-Type : C-classification
  SVM-Kernel : radial
    cost : 1

Number of Support Vectors: 51
( 8 22 21 )

Number of Classes: 3

Levels:
  setosa versicolor virginica
```

We can obtain predicted values for the classes (i.e., class labels, *setosa*, *versicolor*, *virginica*) as follows, where we also generate a confusion matrix to better visualize the classifications:

```
> pred <- predict(model, x)
> table(pred, y)
      y
pred   setosa versicolor virginica
setosa    50         0         0
versicolor 0         48         2
virginica  0         2        48
```

We see that there are 4 errors in prediction (the highlighted “2” values). We now obtain predicted values and request the actual continuous decision values that led to the classification results just observed. These continuous values are requested by `decision.values = TRUE`:

```
> pred <- predict(model, x, decision.values = TRUE)
> attr(pred, "decision.values")[1:4,]
```

|   | setosa/versicolor | setosa/virginica | versicolor/virginica |
|---|-------------------|------------------|----------------------|
| 1 | 1.196152          | 1.091757         | 0.6708810            |
| 2 | 1.064621          | 1.056185         | 0.8483518            |
| 3 | 1.180842          | 1.074542         | 0.6439798            |
| 4 | 1.110699          | 1.053012         | 0.6782041            |

The following will visualize the classes by color, where the support vectors are indicated by crosses (code in Meyer et al., 2023):

```
> plot(cmdscale(dist(iris[, -5])),
+ col = as.integer(iris[, 5]),
+ pch = c("o", "+")[1:150 %in% model$index + 1])
```

The function `cmdscale()` is *classic multidimensional scaling* performed on the iris data with Euclidean distances, `dist()` computed on all variables except the fifth variable (`[-5]`), which is species. The plotting character (`pch`) is for cases 1 through 150 for the model indexed by case number (+1 is the first column representing case number in the iris data). The plot appears in Figure 9.11.

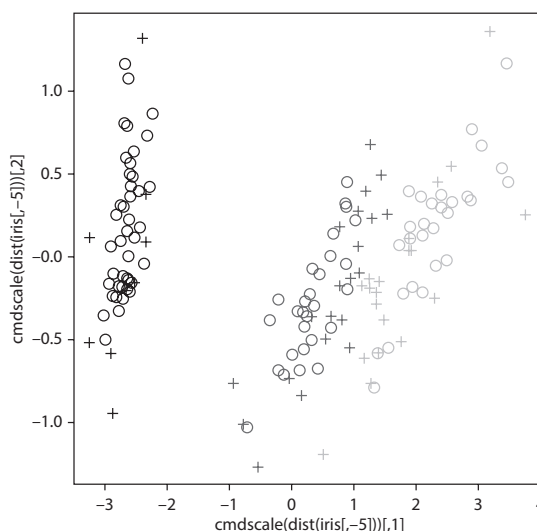


Figure 9.11 Discrimination plot resulting from support vector machine on iris data.

The `e1071` package contains several more options (including cross-validation techniques) for running support vector machines as well as a host of other statistical and machine learning methods (e.g., clustering and Naïve Bayes). The `caret` package (Kuhn, 2024) can be used as with other methods, for cross-validation and, for `svm` specifically, selecting the value for  $C$  (tuning parameter) that maximizes accuracy.

The `MASS` package (Venables & Ripley, 2002) also has excellent `svm` visualization capabilities. For example, the following plots the classification boundaries for a support vector machine on the *cats* data (Figure 9.12), where the objective is to predict *sex* membership (*male* or *female*) based on bodyweight (`Bwt`) in kilograms and heart weight (`Hwt`) in grams:



```
> data(cats, package = "MASS")
> head(cats)
  Sex Bwt Hwt
1  F  2.0  7.0
2  F  2.0  7.4
3  F  2.0  9.5
4  F  2.1  7.2
5  F  2.1  7.3
6  F  2.1  7.6

> m <- svm(Sex~., data = cats)
> plot(m, cats)
```

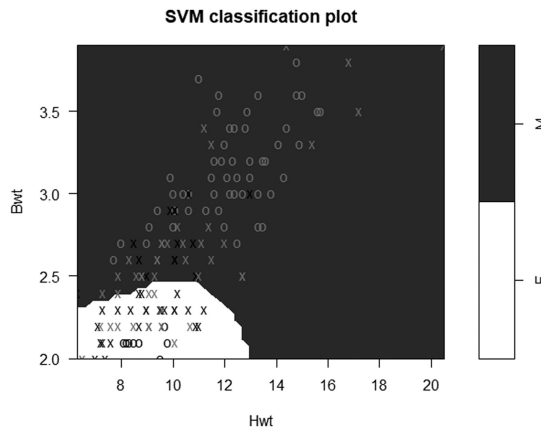


Figure 9.12 Nonlinear decision boundary on the cat data.

We can see from the plot the nonlinear boundary separating male cats from females on the variable of sex based on Bwt and Hwt.

## 9.5 Chapter Review Exercises

1. What is a *maximal margin classifier* and why is the length of  $\beta$  typically set to 1.0?
2. How is a maximal margin classifier different from an ordinary least-squares regression line? Are they not both *linear*?
3. Interpret the following inequality:  $y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) > 0$ .
4. How does a maximal margin classifier go about classification? Explain.
5. What is the significance of the maximum margin classifier's solution being obtainable by *convex optimization*? What if the solution was nonconvex?
6. Why is the case of no separating hyperplane challenging to deal with? Aside from support vector machines, what are some other possible solutions to dealing with this problem?
7. What is the influence of a *slack variable* in the *support vector classifier*? How does this help define a *soft margin*?
8. What role does the kernel function play in addressing potential nonlinearity in the classification problem? Explain the *kernel trick* in relation to support vector machines.
9. Explain how the *linear kernel* reduces to simply taking the *inner product* of two vectors.
10. Perform a support vector machine on the iris data using a *sigmoid kernel*. Compare results to using a *radial kernel*. Are results similar or different?

## 10 Decision Trees, Bagging, Random Forests and Committee Machines

Thus far in this book we have surveyed a number of *learners*. A *strong learner*, as can be said of an intelligent human being, adapts relatively quickly to new data (or *experience* more generally) and adjusts how it does things as the incoming data call for it. One touch of the hot stove psychologically conditions the individual to stay away from hot stoves – perhaps forever. A “close call” and near traffic accident on a winter road conditions you to forever drive with more care in winter, or even perhaps in all seasons. If we had the choice, *we would usually prefer strong learners*.

A *weak learner* learns much more slowly. This is the individual who has five DUI arrests yet still risks the lives of others and “never learns” (and probably belongs forever in jail). More statistically, a weak (also called a *base*) classifier is one that correctly classifies just over 50% of the time, that is, a bit better than chance (Izenman, 2008). Even though we would like all learners to be strong, the truth is that many (perhaps most) times, we are confronted with weak learners. Given this is the case, the question then becomes the following:

*How can we take a series of weak learners and combine them to produce a stronger one?*

This is what this chapter is all about. It is about how to take a number of weak learners, combine them in *some capacity*, and generate a much stronger learner. Perhaps the most amazing thing here is that by combining weak learners, we generate a *stronger learner*. This in itself is quite a remarkable idea. There are even times when choosing a set of weak learners dominates a single strong learner. Hence, though we in general prefer strong learners, sometimes combining weaker learners is actually more efficient. Five “close calls” with the hot stove (set of weak learners) may teach you to avoid hot stoves better than a single experience (strong learner) in which you undergo an actual burn.

*Given the choice, we would generally prefer all learners be strong learners. However, by combining weak learners, it becomes possible to generate a stronger learner. Combining a set of weak learners can actually be preferable in some cases over a single stronger learner.*

### 10.1 Why Combining Weak Learners Works: Concept of Variance Reduction Using Averages

It is well known in statistics that *averaging reduces variance*. That is, when one repeatedly guesses the average income of a population, one “centers” in on a figure at the expense of a lot of variability around that figure.

So-called *ensemble* methods or *committee machines* combine weak learners to produce stronger ones, ones that are more *stable* than not. Unstable predictors or classifiers have high variance (due to overfitting) and low bias. As we will see, methods such as *decision trees* suffer from high variance such that if we tweak the data slightly, we are likely going to see a significant change in the outcome. This is what suffering from high variance means. It means if you *perturbate* (i.e., “change”) the data in different ways, even if rather slight, it results in much *perturbated* outcomes, as Figure 10.1 illustrates.

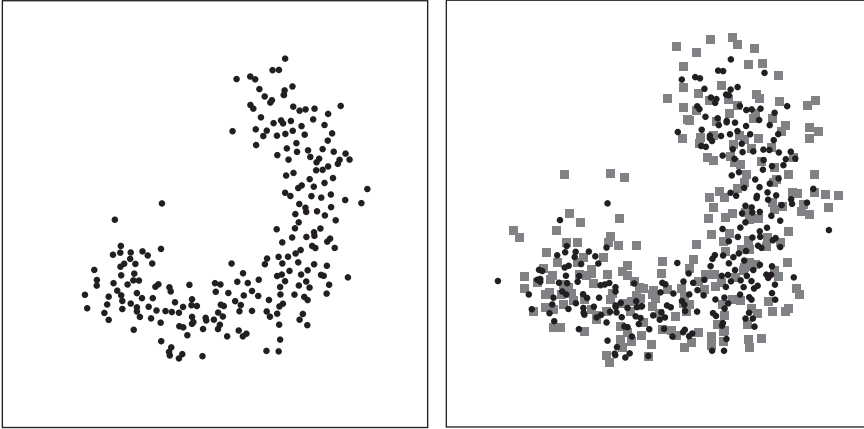


Figure 10.1 Example of perturbing a data set. Original data set (left). Perturbated data set by adding random points (right).

Decision trees and neural networks are relatively unstable methods, whereas methods such as linear regression and discriminant analysis are much more stable, having low variance and higher possibility of bias. Even with these techniques, however, *committee machines* can be used to combine learners to produce a more stable estimator. The stability is reflected in how well the learner generalizes to new test data, and hence, committee machines or *ensemble methods*, as they are sometimes called, can also be interpreted as a form of regularization (Goodfellow et al., 2016). For example, Perrone and Cooper (1993) have proposed producing committees of neural networks instead of selecting the single neural network with the best generalization performance. Indeed, it can be shown that the error generated by a committee of networks is less than or equal to the error generated by an average of those networks. Again, this is quite remarkable in and of itself – *a team of mediocre players performs better than a single stronger player; a pack of weaker wolves performs better than a single strong wolf*.

Hence, and perhaps somewhat paradoxically and unexpectedly, the instability of an algorithm can actually be desirable, in a sense, or at minimum not detrimental, and thus an important tool that can be used to exploit and improve the accuracy of that learning algorithm. Both *bagging* and *boosting*, as we will see, actually exploit this relative weakness, the weakness being the instability, in order to create a more accurate learning method. Weak(er) learners can be combined to generate a stronger one. By perturbing the learning set, these methods generate an “ensemble” of different base predictors or classifiers which are then combined into a single combined predictor or classifier. How successful is this combination? The success will depend in part on the degree of instability in the individual base predictors. Bagging has been shown to be beneficial when combining learners that tend to overfit the data and are relatively unstable (Kroese et al., 2020).

On the opposite end, as mentioned, a procedure is said to have low variance if it yields similar results on distinct data sets or even subsets of the same dataset. That is, if you perturbate the data somewhat, it still generally leads to a similar outcome. Recall the bias-variance trade-off that has dominated this book thus far, which we remind here:

*Good learners have low bias and low variance; however, as one decreases, the other is likely to increase; hence the trade-off.*

Before we consider committee machines in more detail, we need to survey the important topic of *decision trees*. We start with decision trees for two reasons: 1) decision trees are important in their own right and constitute a rather simple way to understand the statistical influence of predictors, and 2) our discussion of bagging and boosting will center around decision trees. As we will see, averaging several decision trees is a prime example of constructing a committee machine, though averaging and combining other procedures also has similar effects.

## 10.2 Decision Trees

Statistical learning methods based on decision trees have gained tremendous popularity due to their simplicity, intuitive representation, and to some extent, predictive accuracy, though often they come in second when compared to many other learning methods. Of all statistical learning methods, decision trees, a member of the tree-based family of algorithms, have been regarded as perhaps the most useful of so-called “off-the-shelf” procedures for data mining. In the context of regression, regression trees were first introduced by Morgan and Sonquist (1963) using a computer program called *Automatic Interaction Detection* (AID). For a history of decision trees, consult Ripley (1996, Section 7.4). As we will discuss shortly, boosting and bootstrap aggregation (*bagging*) can improve the efficiency of decision trees and other learning methods.

In Figure 10.2 is a basic decision tree, one that we will revisit shortly.

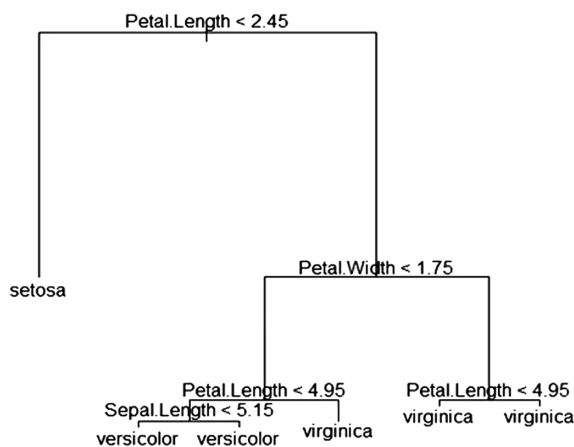


Figure 10.2 Example of a basic decision tree on the iris data.

The *root node* in the decision tree is the unique starting point of the tree, and where the tree ends is referred to as the *terminal node*. Along the way down the tree, branches are created that represent splits in the given variable. Of course, not all nodes are terminal. A nonterminal node splits into two daughter nodes, which creates a binary split. If the variable is binary, then defining the nature

of the split is easy (e.g., 0 vs. 1). If the variable is continuous, however, it is made discrete and binary, such as split at the median, for example. A single-split tree with only two terminal nodes is called a *stump*. The set of all terminal nodes is called the *partition of the data*.

Decision trees have a lot of benefits to them. For one, they can handle very large datasets. They can also handle a mixed bag of predictors, that of *quantitative* and *qualitative*. They can be used both for regression (where targets or response variables are *continuous*) and classification problems (where targets or predictors are *categorical*). They handle missing data quite easily and in general, trees are relatively easy to interpret, especially if the tree is small. Downsides to decision trees include the fact that larger trees can be difficult to interpret and the predictive performance of decision trees is often not as good as competing methods. Further, deep trees (i.e., those without pruning, also known as *saturated trees*) are often too heavily reliant on the training data used to construct them. Small changes in training data can result in a dramatic change in the resulting tree. Because decision trees often exhibit high variance, committee machines such as *bagging*, *boosting*, and *random forests* have come into vogue and are especially useful when applied specifically to decision trees.

### 10.2.1 How Should Trees Be Grown?

We have demonstrated a simple decision tree above, but how do we grow such a tree? How do we go about creating the relevant nodes and branches? At each node, the tree-splitting algorithm needs to decide which variable is “best” in terms of providing a split. That is, at each node, the algorithm needs to select a variable that provides differentiation with regard to the response variable. We must consider every possible split over all variables present at the given node, then enumerate all possible splits, evaluate each one, and decide which is ideal in some sense (Izenman, 2008, p. 285). If the variable at which a split is contemplated is ordinal or continuous, the number of possible splits at a given node is one fewer than the number of its distinctly observed values. If the variable is nominal or categorical, the set of possible splits at that node for that variable is the set of all subsets.

### 10.2.2 Optimization Criteria for Tree-Building

As is true with virtually all statistical methods, we should require a condition to optimize in creating the relevant branches for the trees. Since we cannot consider *every* partition of the predictor space (such would take too long and have too many possibilities), we need to adopt an overall approach to doing so. One approach that is often taken is a *top-down, greedy approach*, referred to as *recursive binary splitting* or more generally *recursive partitioning* (Izenman, 2008) which seeks to minimize the following criterion:

$$\sum_{j=1}^J (y_i - \hat{y}_{s_j})^2$$

which the reader will recognize as the familiar *residual sums of squares criterion* (RSS) where  $y_i$  are observed values and  $\hat{y}_{s_j}$  are predicted values within a given subsection of the decision surface for a combination of predictors. The predictor which results in the greatest reduction in RSS is chosen at each step. This process continues for each predictor and space until some criterion to stop the process is attained. The stopping criteria may include a consideration of sample size limitations. That is, tree splitting may continue until there is minimal  $n$  at each split, enough to halt the algorithm.

### 10.2.3 Why Not Multiway Splits?

In our construction of decision trees, note that we are performing *binary* splits at each node. But why not perform splits containing more than two possibilities? While it is also possible to split a tree node into more than two groups, generally referred to as *multiway* or *multivariate* splits, it generally produces inferior results when compared to the simpler binary split. As well, multiway nodes can often be difficult to interpret (Aggarwal, 2015; Alpaydin, 2020). The major reason for this is that multiway splits can lead to too many nodes near the tree root that have only a few data points, thus leaving insufficient data for later splits. Therefore, instead of attempting multiway splits, the generally preferred strategy is for numerous binary splits.

At times, it can be beneficial to preprocess the data before fitting a tree. This includes standardization of the data. Trying out both solutions, raw and standardized, may be advantageous to learn of any effects due to standardizing scores prior to building a tree.

*Minimizing residual sums of squares is a common criterion for choosing a predictor at each step of building a decision tree. The algorithm is “greedy” in that by minimizing RSS at a given split, it does not consider longer-term consequences for branches to follow.*

### 10.2.4 Overfitting, Saturation, and Tree Pruning

A single decision tree fit to data is likely to overfit the data because it is built too specific to the particular data set. This would be akin to fitting a regression curve (not limited to a line) a bit too closely to a scatterplot of points. Of course, the consequence of this is that an overfit tree is likely to not perform that well on test data for reasons we are already familiar with.

If the tree is grown to the point where none of the nodes can be split any further, the tree is said to be *saturated*. That is, saturated trees have exhausted their potential for performing splits. Especially in high-dimensional space, it is very easy to allow the tree to grow extremely large, and if we allow it to grow to the point of saturation, it can actually become unwieldily large. So, how does one prevent overfitting? In the earlier days of decision trees, tree growers attempted to restrict the growth of the tree by implementing so-called “stopping rules” to prevent overfitting. However, as shown by Breiman et al. (1984), such stopping rules are not generally advised and perhaps counter-intuitively, the preferred solution is often to build a very large tree and then perform what is known as *pruning* to obtain a *subtree* (Hastie et al., 2009, p. 308). That is, it can be shown that building a saturated tree and then *scaling back* the tree is often preferable to early stopping or trying to, more generally, *regularize* the tree too early.

How to prune then becomes a crucial process. Indeed, much of the research in the evolution of decision trees has not been so much about how to grow a tree, but rather *how to prune it back*. The general problem to overcome in pruning is that *pruning generates bias*. Why is this the case? Consider that when we prune a tree, we are now imperfectly estimating the data, the same way a regression line imperfectly estimates a swarm of data in a scatterplot. Another approach might be to forego tree pruning entirely and simply grow many trees quite large (via resampling) and then take an average of the trees to reduce variance. As we will see later, this technique of averaging is referred to as *bagging*, which is a short name for *bootstrap aggregating*. Not surprisingly, the “best” pruned tree would be the one that performs best on *cross-validation*. Doing so, however,

would involve entertaining many different trees. Hence, we need an alternative strategy for pruning and selecting a smaller set of trees for consideration.

*Decision trees are easily subject to overfitting. Implementing “stopping rules” to prevent the tree from growing too large is usually not the ideal solution. The preferred solution is often to allow the tree to grow to saturation and then prune it back.*

### 10.2.5 Cost Complexity or Weakest Link Pruning

The procedure of *cost complexity pruning*, which also goes by the name of *weakest link pruning*, is one way of providing us with the subset of trees we require when pruning. Instead of considering every possible tree, we consider only a sequence of trees that will be indexed by a nonnegative tuning parameter  $\alpha$ . Tuning parameters, of course, are not new by now, as we have seen them repeatedly in other procedures such as ridge and lasso regression. For each value of the tuning parameter  $\alpha$ , we will obtain a *subtree* such that the following criterion is minimized:

$$\sum_{j=1}^J (y_i - \hat{y}_{s_j})^2 + \alpha |T| \quad (10.1)$$

The number of terminal nodes is indicated by  $|T|$  of the given tree  $T$ . The tuning parameter  $\alpha$  controls the trade-off between the subtree’s complexity and its fit to the training data (James et al., 2013). We can see that as  $\alpha$  increases, it results in a relatively large increase in the overall error *especially when there are several terminal nodes to the tree* ( $|T|$ ) since  $|T|$  multiplied by a larger number than not in  $\alpha$  will result in a larger product  $\alpha |T|$  than not. Informally, what this means is that if you want a lot of nodes to your tree, *then there is a cost to doing so*, since we would expect RSS to decrease with increasing number of nodes unless we invoked a penalty. This idea parallels that of multiple regression where the addition of more predictors, without a penalty, generally decreases residual sums of squares. Recall the goal is to minimize  $\sum_{j=1}^J (y_i - \hat{y}_{s_j})^2$  which means smaller values are preferred to larger values. The choice of  $\alpha$  is generally made through cross-validation or an independent sample test set where an independent test set may be preferable to cross-validation if the data set is large enough and can be split accordingly. It should be emphasized as well that for any value of the penalty, there may be more than a single minimizing subtree which makes selection of the “best” tree in this situation a bit more challenging. As values for  $\alpha$  increase away from zero, the decision tree gets “pruned” and so obtaining an ideal value for  $\alpha$  is of paramount importance (James et al., 2013, p. 309).

*For increasing values of  $\alpha$ , the product  $\alpha |T|$  increases, where  $T$  is the number of terminal nodes. That is, as the complexity of the tree increases,  $\alpha |T|$  serves as a penalty and increases RSS. Hence, if the nodes included in  $T$  are not “worthwhile” in reducing RSS sufficiently, then penalized RSS will end up increasing rather than decreasing.*

### 10.3 Classification Trees

In our survey of decision trees thus far, we have discussed only those of regression trees. Decision trees, not surprisingly, are also applicable to classification. Classification trees are used to predict *qualitative* or *categorical* responses instead of *continuous* responses as was the case for the regression trees just surveyed. As for regression trees, *recursive binary splitting* is used to build the tree. However, in assessing the quality of the tree, we cannot use RSS as the quantity to minimize since RSS is applicable to continuous measures only.

What to use for estimating error rates instead of RSS? An alternative for classification trees is the *classification error rate*:

$$E = 1 - \max_k (p_{mk}) \quad (10.2)$$

where here  $p_{mk}$  is the proportion of observations (on the *training* data) in the  $m$ th region of the tree that are classified in the class denoted by  $k$  (James et al., 2013). Hence, (10.2) is the fraction of observations in error for the given class. The classification error rate is not always a perfect measure of the quality of the tree, however. A problem with the classification error rate is that it is not sensitive enough of a measure for growing a tree (James et al., 2013, p. 312). That is, it is a bit too “crude” of a measure. We require other measures to help in assessing the build of our tree. These two measures are the *Gini index* and the *deviance*, which we now discuss.

#### 10.3.1 Gini Index

The *Gini index* is a measure that can be used in place of, or to supplement the classification error rate and can take the place of residual sums of squares in the regression tree context. It is a measure of *node purity* (also often called *node impurity*), defined by

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) \quad (10.3)$$

which is a measure of total variance across the  $k$  classes, where recall  $\hat{p}_{mk}$  is the proportion of training observations in the  $m$ th region that are from the  $k$ th class. As a measure of total variance across the classes, the index takes on a smaller-than-not value the closer the  $\hat{p}_{mk}$  values are to 0 or 1. That is, when  $p_{mk}$  is equal to 0, then  $G$  reduces to

$$\begin{aligned} G &= \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) \\ &= \sum_{k=1}^K 0(1 - 0) \\ &= 0 \end{aligned}$$

On the other extreme, when  $\hat{p}_{mk} = 1$ , then

$$\begin{aligned} G &= \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk}) \\ &= \sum_{k=1}^K 1(1 - 1) \\ &= 0 \end{aligned}$$



A smaller-than-not value for the Gini index indicates that a node contains predominantly observations from a single class (James et al., 2013). The variance is maximized as  $\hat{p}_{mk} \rightarrow 0.5$  across the classes, which equals  $0.5(0.5) = 0.25$  per class. Any deviation from this maximum, for instance,  $0.4(1 - 0.4) = 0.24$  produces a lesser product and hence contributes less variance to the sum (Bishop, 2006). For  $\hat{p}_{mk} = 0.1$ , for instance, the given product is  $0.1(0.9) = 0.09$  which is quite small indicating hardly any variance across the classes. This again is why  $\hat{p}_{mk} \rightarrow 0.5$  maximizes the variance of the classes.

An alternative to the Gini index is *cross-entropy* or *deviance*, defined by

$$D = -\sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk} \quad (10.4)$$

Recall that  $\hat{p}_{mk}$  is bound between 0 and 1, which implies that  $-\hat{p}_{mk} \log \hat{p}_{mk}$  must be equal to or greater than 0 (James et al., 2013, p. 312). Entropy takes on a value close to 0 for  $\hat{p}_{mk}$ 's approaching 0 or 1. That is, as was the case for the Gini index, overall *node purity* is suggested by a small  $D$  (James et al., 2013). Entropy can be interpreted as the minimum number of bits of information required to encode the class code of an instance or observation (Alpaydin, 2020).

*The Gini index and entropy are very similar to one another. Both attempt to provide a measure of how much variance exists across the classes. They can both be regarded as measures of node purity.*

### 10.3.2 Decision Trees in R

We will use the *tree* package in R and code (Ripley, 2023) to compute a decision tree on the iris data:

```
> library(tree)
> ir.tr <- tree(Species ~., iris)
> ir.tr
```

The code `Species ~., iris` requests a tree based on sepal length, sepal width, petal length, and petal width on the iris data. The output below details the splitting decisions made at different points in the tree. What will be most useful is the summary given at the end of the output, not the specific details of the splitting:

```
node), split, n, deviance, yval, (yprob)
* denotes terminal node
1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
  2) Petal.Length < 2.45 50 0.000 setosa ( 1.00000 0.00000 0.00000 ) *
  3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
    6) Petal.Width < 1.75 54 33.320 versicolor ( 0.00000 0.90741 0.09259 )
      12) Petal.Length < 4.95 48 9.721 versicolor ( 0.00000 0.97917 0.02083 )
        24) Sepal.Length < 5.15 5 5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
        25) Sepal.Length > 5.15 43 0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
```

```

13) Petal.Length > 4.95 6 7.638 virginica ( 0.00000 0.33333 0.66667 ) *
7) Petal.Width > 1.75 46 9.635 virginica ( 0.00000 0.02174 0.97826 )
14) Petal.Length < 4.95 6 5.407 virginica ( 0.00000 0.16667 0.83333 ) *
15) Petal.Length > 4.95 40 0.000 virginica ( 0.00000 0.00000 1.00000 ) *

> summary(ir.tr)

Classification tree:
tree(formula = Species ~ ., data = iris)
Variables actually used in tree construction:
[1] "Petal.Length" "Petal.Width" "Sepal.Length"
Number of terminal nodes: 6
Residual mean deviance: 0.1253 = 18.05 / 144
Misclassification error rate: 0.02667 = 4 / 150

```

We see above the residual mean deviance (0.1253) as well as the misclassification error rate of 0.02667. We now plot the tree (Figure 10.3), where `pretty = 0` keeps the level names of factor splits unchanged (see the package for how this parameter can be adjusted):

```

> plot(ir.tr)
> text(ir.tr, pretty=0)

```

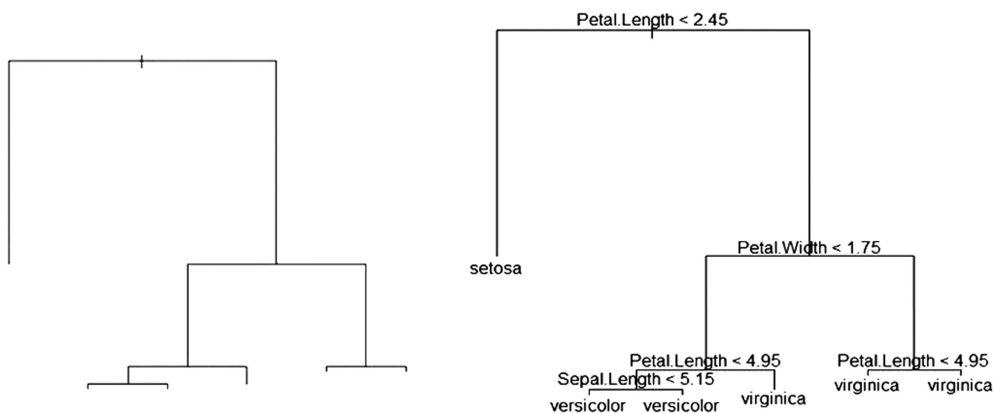


Figure 10.3 Plot of decision tree on the iris data. Tree without labels (left). Tree with labels (right).

We can also obtain the misclassifications of the tree:

```

> ir.tr1 <- tree(Species ~., iris)
> misclass.tree(ir.tr1)
[1] 4

```

The number 4 agrees with the misclassification error rate reported above. We also see that the number of terminal nodes agrees with the tree we generated. That is, the endings of each branch terminate with either *setosa*, *versicolor*, *versicolor*, *virginica*, *virginica*, or *virginica*.

We now prune our original tree, which will allow us to observe misclassification error rate as a function of tree size:

```
> cv.iris = cv.tree(ir.tr, FUN=prune.misclass)
> cv.iris
$size
[1] 6 4 3 2 1
$dev
[1] 8 8 11 95 112
```

The output `$dev` refers to the cross-validation error rate (“misclass” in the plot below). We see that it is equal at a value of 8 for trees of size 6 and 4 terminal nodes, as we can more easily see in the plot in Figure 10.4 (`plot(cv.iris, type = “b”)` will produce a similar plot):

```
> plot(cv.iris)
```

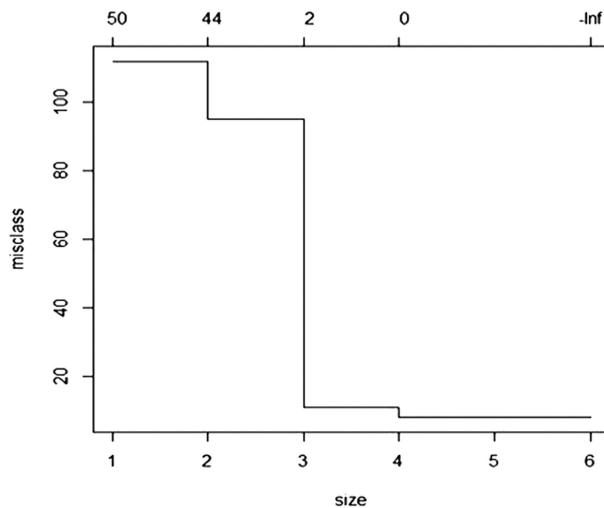


Figure 10.4 Misclassification error as a function of size of decision tree.

We now obtain the pruned tree with 4 nodes (Figure 10.5):

```
> prune.ir.tr = prune.misclass(ir.tr,best = 4)
> plot(prune.ir.tr)
> text(prune.ir.tr)
```

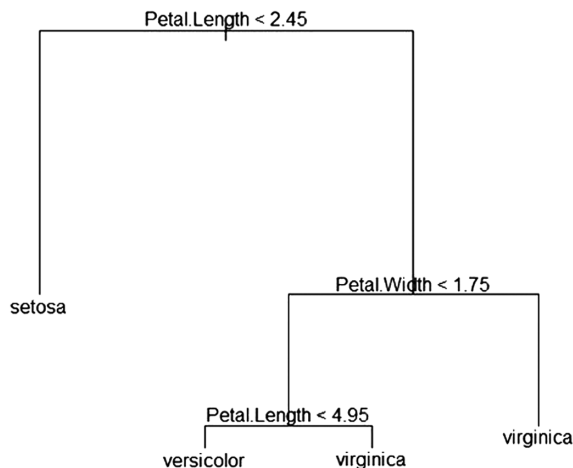


Figure 10.5 Pruned decision tree on the iris data.

We now obtain misclassifications on the tree,

```

> prune.ir.tr = prune.misclass(ir.tr, best = 4)
> misclass.tree(prune.ir.tr)
[1] 4

```

The number of misclassifications is equal to 4. Note that had we chosen a much more crude tree, say with 2 nodes, the misclassification error rate would have skyrocketed, such as in the following where we request only 2 nodes (`best = 2`):

```

> prune.ir.tr = prune.misclass(ir.tr, best = 2)
> misclass.tree(prune.ir.tr)
[1] 50

```

The package *rpart* (Therneau et al., 2023) can also be used to fit decision trees. We will fit a tree to the iris data:

```

> library(rpart)
> iris.rpart <- rpart(formula = Species ~., data = iris, method = "anova")
> iris.rpart
n= 150

node), split, n, deviance, yval
  * denotes terminal node
1) root 150 100.0000000 2.000000
 2) Petal.Length< 2.45 50 0.0000000 1.000000 *
 3) Petal.Length>=2.45 100 25.0000000 2.500000
   6) Petal.Width< 1.75 54 4.5370370 2.092593
     12) Petal.Length< 4.85 46 0.9782609 2.021739 *
     13) Petal.Length>=4.85 8 2.0000000 2.500000 *
   7) Petal.Width>=1.75 46 0.9782609 2.978261 *

```

```
> library(rpart.plot)
> rpart.plot(iris.rpart)
> plotcp(iris.rpart)
```

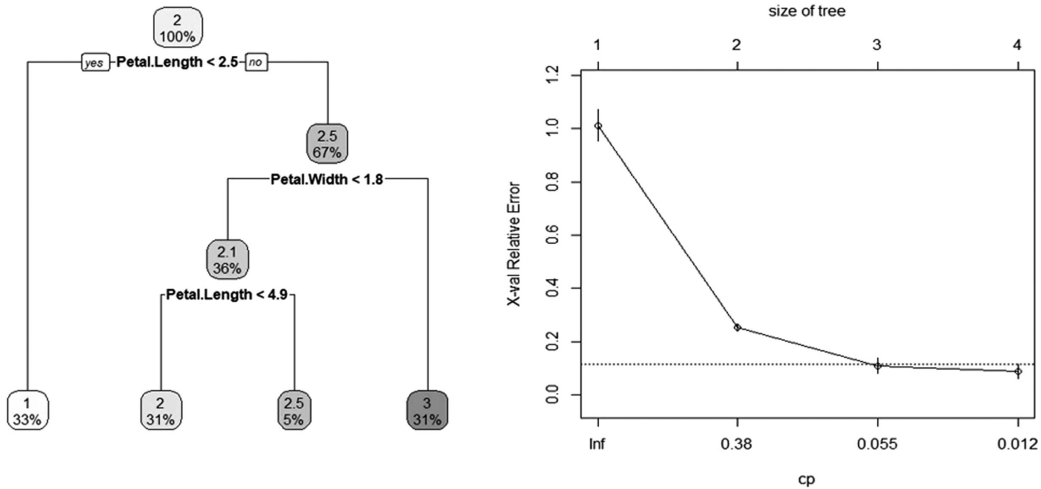


Figure 10.6 Decision tree on iris data (left). Error as a function of decision tree complexity parameter (right).

Percentages of sample size in respective nodes are depicted in Figure 10.6. We note from the figure that a tree size of 3 or 4 gets the relative error down to fairly low levels, where “cp” on the x-axis is the cost complexity parameter (Therneau et al., 2023). In general, setting cp to a smaller value will generate a fuller tree (Boehmke and Greenwell, 2020). See the package for further details regarding the computation of cp.

There has been some research done on multivariate responses for decision trees (Canete-Sifuentes et al., 2021, Zhang & Ye, 2008). Survival trees (Izenman, 2008), also known as *conditional inference trees*, have also been advanced. *Multivariate adaptive regression splines* (MARS), due to Friedman (1991) is a generalization of recursive partitioning and is a nonparametric regression approach which uses basis functions. For details, see Izenman (2008, p. 311). Work has also been done on *Bayesian additive regression trees* (Chipman et al., 2010).

## 10.4 Committee Machines

As mentioned at the outset of this chapter, “committee machine” is a general term referring to combining weaker learners to generate a stronger learner. By far the most popular and useful of committee machines are those of *bagging* and *boosting*.

## 10.5 Overview of Bagging and Boosting

We have so far in this chapter discussed decision trees and have focused on their construction. We have emphasized that a single tree, especially if not pruned, likely significantly overfits the data. That is, it is easily subject to random perturbations in the data such that if we changed a few data points here or there, the outcome of the tree would be different. In statistical learning theory, as we

know by now, this is referred to the method having *high variance*. Note carefully this is not the *data* having high variance. We are referring here to the *method*. The data are what they are. We are more interested in evaluating the *method* used to evaluate the data. Unstable methods are usually not preferred to more stable ones. Decision trees are not the only statistical method that is unstable, but in what follows, we will use them as an example in our discussion of bagging and boosting.

How can we reduce the variance of an otherwise unstable method? Though there are many ways of accomplishing this reduction in variance, two methods are often used: *bagging* and *boosting*; however, boosting prioritizes reducing *bias* over reducing variance. Bagging, which is short for *bootstrap aggregating*, is random and probabilistic. By essentially averaging bootstrapped samples, one could get different bagging results depending on the given sampling procedure. Boosting, on the other hand, is not random or probabilistic. It is instead *systematic* or *deterministic*. Boosting generates predictable non-random perturbations of the training data by successive reweighting of the training set, where the most current and updated weights depend on misclassification history. We survey boosting shortly. First, we more fully develop the idea of bagging.

### 10.5.1 *Bagging*

Bagging is essentially a procedure of taking many bootstrapped samples and then aggregating (averaging) them. By averaging results, the goal is to reduce variance. For example, as already discussed, in a typical decision tree, any *particular* tree likely suffers from high variance, which basically means that a given tree is subject to random fluctuations in the data. That is, if we perturbed the data somewhat, the resulting tree would be different. By averaging trees (or whatever method we are using), the hope is to reduce the overall variance. In fact, the success of bagging actually depends on how originally unstable the data are; *the more unstable, the more perturbing the data set causes a change in prediction, the more advantageous it is to bag to improve accuracy* (Breiman, 1996). Be sure to understand what we have just said. We emphasize that *the more unstable a learning method, the more it stands to benefit from aggregating*. If the learning method is stable, bagging will, in general, be less efficient. In fact, for relatively stable methods, bagging has been found to work less well and has been found to actually *degrade* the performance of stable procedures (Breiman, 1996).

### 10.5.2 *A Familiar Example: Bagging Samples and the Variance of the Mean*

The simplest conceptual example of bagging would be to designate a sample of say, 100 observations, bootstrap 10 samples, on each compute a sample mean, then average those means. The average of those averages  $\mu_{\bar{y}}$  will have less variance than any of the individual averages,  $\bar{y}$ . *Averaging (in this case, of means) reduces variance*. If each of the subset samples has a variance equal to  $\sigma^2$ , then the variance of those means, as is well known, is equal to  $\sigma^2 / n$ , with standard error  $\sigma / \sqrt{n}$ .

Hence, when we think of bagging, it is convenient to simply think of the word *averaging*. *Averaging reduces variance*. As an analogy, imagine taking betting predictions for a sporting event from one forecaster vs. ten forecasters. If you averaged the results from ten forecasters, the prediction would have less variance. On average, the forecast might be more “stable” than if you relied on the prediction of a single, stronger forecaster. This is the idea of bagging, except that we will apply it to bootstrapped samples instead of to actual empirical divisions of the data. In theory, if we had enough data, we could feasibly take segments of the data and average results, but usually,

we do not have enough data to do this. This is why bootstrapping is utilized in bagging – *because of an insufficient amount of real data*.

### 10.5.3 A Deeper Look at Bagging

Let us look more closely into how bagging works. First, we begin with the most ideal and fanciful but unrealistic of situations. Suppose we apply a statistical learning method to some data and compute a function on many different empirical subsets of that data. Let's suppose for a moment that we were able to partition the data perfectly and have all subsets of the data represented. Then, the average of all those subsets would be the following:

$$\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x)$$

where  $B$  is the number of subsets that we are drawing and computing on each a function on  $x$ .

The estimated function is  $\hat{f}^b(x)$ . Thus,  $\hat{f}^b(x) / B$  is the average of all those computed estimated functions. But why is this situation unrealistic? It is so because we are likely to not have access to all partitions of the data, or, the number of partitions of the data would likely simply be too large to be practical. Data is usually expensive (at least in some fields), and so the idea of having large partitions of data available may not be pragmatic. However, it should be emphasized that if this were possible, it would be the ideal situation to shoot for. *Real data is always better than simulated data*.

The way around the above problem will be to bootstrap samples from the only sample we do have and compute the average of all such bootstrapped samples:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

Notice that the only thing that has changed is where before  $\hat{f}_{avg}(x)$  was the average of all partitions of the training data, now  $\hat{f}^{*b}(x)$  is the average of all the *bootstrapped* samples. Hence, we are averaging a subset of all samples via the bootstrap procedure. That is, we are *aggregating the bootstrapped samples*. This is precisely what bagging means. *It simply means taking an average of functions via bootstrapping*. For a good theoretical discussion of just why bagging works on a technical level superseding our intuitive conceptual explanation, see Breiman (1996, p. 129). Breiman justifies that the degree of instability in the method is paramount to determining whether bagging will or will not be effective.

*Bagging is simply taking an average of bootstrapped samples. The reason for taking bootstrapped samples is that dividing up the data into many divisions is impractical. Dividing the actual empirical data into subset samples would, theoretically, be our first choice, however, if practical.*

When working with decision trees, bagging can combine hundreds or even thousands of bootstrapped trees. Think of what this means for a moment – we have a single sample, we bootstrap many samples from this sample, and on each, we compute a separate decision tree. We then average the results of all the decision trees we compute. Each tree alone suffers from high variance,

but when we bag (i.e., average) all of the bootstrapped trees, *we reduce variance*. As already discussed, bagging is especially useful when dealing with predictors that tend to overfit the data, such as in decision trees, where the unpruned tree structure is very sensitive to small changes or perturbations in the training set.

#### 10.5.4 *Out-of-Bag Error*

Once we bag, we would like to know how well the bagging procedure performed. We need a way of evaluating the results of the bagging. To properly evaluate, it would make little sense to validate on the bagged decision trees (or whatever statistical learning method we used). This would be akin to evaluating how well a boxing fighter performs on practice opponents; it is not a true test of the boxer's skill.

To evaluate the goodness of bagging, *we require observations that were not part of the bagging procedure*. That is, as with other cross-validation, we require a kind of *test set* on which to evaluate bagging. Observations that were not part of the bagging procedure are referred to as *out-of-bag (OOB) observations*. These are the observations that are part of the training set, but not used to generate the given bagged tree. In spirit, they serve the same purpose as in typical cross-validation; however, the out-of-bag observations are simply the set of observations (roughly 1/3; James et al., 2013) in the training data not used in generating the given bootstrapped tree.

#### 10.5.5 *Interpreting Results from Bagging*

When we bag, there is no easy way to interpret the result since now we have a number of trees rather than a single tree. Hence, while bagging does tend to improve prediction accuracy, as is true of so many machine learning algorithms, it does so generally at the expense of *interpretability*. There is simply no easy way to combine trees and read off the “meaningfulness,” in some sense, of the individual predictors.

The situation is not entirely hopeless, however. There are some measures of importance that can be used. For bagging regression trees, as already discussed, a decrease in RSS can be used to evaluate the importance of a predictor due to its split inclusion. A large decrease in RSS suggests the given predictor is valuable to the decision tree. In a classification context, decreases in the *Gini index* can be used in a similar way to evaluate the importance of a given variable. Large decreases in the Gini index suggest the variable under consideration may be important.

It should be emphasized as well that bagging does not always improve performance and relies very much on the base models being *unstable*. There are some situations where bagging will not improve performance or prediction. For instance, Buja and Stuetzle (2000) found bagging to, at times, increase bias; however, the increase or decrease in mean squared error was contingent upon the given statistic and its distribution. There is no free lunch, and hence bagging on one data set may perform better or worse than bagging on another.

#### 10.5.6 *Bagging in R*

Many packages can be used to bag in R such as *baguette* (Kuhn, 2024). Bagging for classification in R can be performed using *ipred* (Peters & Hothorn, 2023) and *rpart* (Therneau et al., 2023). Below we feature an example from the package *ipred*, which uses data on breast cancer from the package *mlbench* (Leisch & Dimitriadou, 2024) featuring various characteristics of cells:



```

> library(ipred)
> library(MASS)
> library(survival)
> data("BreastCancer", package = "mlbench")
> head(BreastCancer)

```

|   | Id      | Cl.thickness | Cell.size | Cell.shape | Marg.adhesion | Epith.c.size |
|---|---------|--------------|-----------|------------|---------------|--------------|
| 1 | 1000025 | 5            | 1         | 1          | 1             | 2            |
| 2 | 1002945 | 5            | 4         | 4          | 5             | 7            |
| 3 | 1015425 | 3            | 1         | 1          | 1             | 2            |
| 4 | 1016277 | 6            | 8         | 8          | 1             | 3            |
| 5 | 1017023 | 4            | 1         | 1          | 3             | 2            |
| 6 | 1017122 | 8            | 10        | 10         | 8             | 7            |

|   | Bare.nuclei | Bl.cromatin | Normal.nucleoli | Mitoses | Class     |
|---|-------------|-------------|-----------------|---------|-----------|
| 1 | 1           | 3           |                 | 1       | benign    |
| 2 | 10          | 3           |                 | 2       | benign    |
| 3 | 2           | 3           |                 | 1       | benign    |
| 4 | 4           | 3           |                 | 7       | benign    |
| 5 | 1           | 3           |                 | 1       | benign    |
| 6 | 10          | 9           |                 | 7       | malignant |

We now bag the following model in a single step, where *class* is modeled as a function of cell characteristics:

```

mod <- bagging(Class ~ Cl.thickness + Cell.size
               + Cell.shape + Marg.adhesion
               + Epith.c.size + Bare.nuclei
               + Bl.cromatin + Normal.nucleoli
               + Mitoses, data=BreastCancer, coob=TRUE)

```

The designation `coob=TRUE` instructs to use the out-of-bag error rate. We now print the results from the bagging:

```

print(mod)
Bagging classification trees with 25 bootstrap replications
Call: bagging.data.frame(formula = Class ~ Cl.thickness + Cell.size +
  Cell.shape + Marg.adhesion + Epith.c.size + Bare.nuclei +
  Bl.cromatin + Normal.nucleoli + Mitoses, data = BreastCancer,
  nbagg = 100, coob = TRUE)
Out-of-bag estimate of misclassification error: 0.0381

```

We can see that the out-of-bag misclassification error estimate is equal to 0.0381. That is, the classifier only misclassifies approximately 3.8% of cases on unseen data. This result is consistent with what Breiman (1998) found on these same data. The number of bootstrap replications is 25 by default. We can change that number by including `nbagg = 100` just before the `coob = TRUE` statement:

```
mod <- bagging(Class ~ Cl.thickness + Cell.size
               + Cell.shape + Marg.adhesion
               + Epith.c.size + Bare.nuclei
               + Bl.cromatin + Normal.nucleoli
               + Mitoses, data=BreastCancer, nbagg = 100, coob=TRUE)
> mod
```

Bagging classification trees with 100 bootstrap replications

```
Call: bagging.data.frame(formula = Class ~ Cl.thickness + Cell.size +
                          Cell.shape + Marg.adhesion + Epith.c.size + Bare.nuclei +
                          Bl.cromatin + Normal.nucleoli + Mitoses, data = BreastCancer,
                          nbagg = 100, coob = TRUE)
```

Out-of-bag estimate of misclassification error: 0.041

We see that the misclassification error is now equal to 0.041. We can easily obtain predictions based on the bagged model regarding whether the tumor is predicted to be benign or malignant:

```
> predict(mod)
[1]      benign malignant benign malignant      benign malignant malignant
[8]      benign      benign benign      benign      benign      benign      benign
[15] malignant malignant benign      benign malignant      benign malignant
```

The package *baguette* package (Kuhn, 2024) can also be used for bagging in R.

## 10.6 Random Forests

We have seen how bagged trees (or whatever statistical learning method we are using) can help reduce variance in the method by averaging results. The question now is, *can we improve on bagging?* It turns out that we can. The technique of *random forests* is yet another committee machine. Random forests take bagging to the next level, seeking to improve upon it. To appreciate how random forests improve on bagging, we first need to consider what is lacking in bagging in the first place.

### 10.6.1 The Problem with Bagging Decision Trees

We know that in building decision trees, overfitting can occur because the nodes will contain the “best option” at each split. That is, a decision tree is, in a sense, *optimized* to produce the best outcome and hence is capitalizing on the training data a great deal. Thus, we can expect trees to be *similar* in many respects when bagged because each tree is built with the same optimization strategy in mind. Of course, if you build trees with the same optimization, this will quite naturally and quickly lead to overfitting.

*Random forests* are an ingenious way to combat the overfitting problem of combining decision trees. How they do this is by introducing a *randomization feature* into each node split in an effort to combat overfitting. That is, each node is split in a random manner where a random sample of predictors is chosen as split “candidates” from the full set of possible predictors. This improves over bagging alone because when we bag, if there is, say, one strong predictor, most of the trees developed will use this strong predictor in the top split, and hence, *all bagged trees will look similar, which produces predictions that are highly correlated*. Where bagging reduces variance and potentially bias for full-grown trees, a random forest reduces the correlation between

tree-structured classifiers. That is, random forests essentially *decorrelates* the trees through this randomization process. Random forests forces each split to consider only a *subset of the predictors*, which is what “decorrelates” the trees.

*The problem with bagging decision trees is that the trees are correlated. Averaging many highly correlated quantities reduces variance, but not as much as averaging many uncorrelated quantities. Random forests overcomes this problem by introducing a randomization process regarding how predictors are chosen at each split.*

### 10.6.2 Equivalency of Random Forests and Bagging

The observant reader may have noticed that there is a certain condition in which random forests must be equivalent to bagging. Indeed, random forests have a natural relationship to bagging *when the random forest is built using the number of trees equal to the number of predictors*. Under this condition, *random forests equate to bagging*. Random forests then can be considered, in this context, simply a *special case* of bagging where a randomization feature is introduced, or vice versa, one can also consider bagging a special case of random forests where no randomization feature is implemented.

*Random forests can be considered a special case of bagging in which a random sample of decision trees are selected from the bootstrapped “bagged” samples in an effort to decorrelate the trees. When all predictors are used in the random forest and hence there is no randomization element, this then reduces to bagging.*

### 10.6.3 Random Forests in R

The package *randomForest* (Liaw & Wiener, 2002) can be used to build a random forest in R. We demonstrate on the iris data, where we combine an ensemble of trees (see package for more details on this example):

```
> library(randomForest)
> data(iris)

> rf1 <- randomForest(Species ~ ., iris, ntree=50)
> rf2 <- randomForest(Species ~ ., iris, ntree=50)
> rf3 <- randomForest(Species ~ ., iris, ntree=50)
```

We can look at each of the 50 trees before combining them. For instance, for `rf1`, we have:

```
> rf1
```

```
Call:
```

```
  randomForest(formula = Species ~ ., data = iris, ntree = 50)
                Type of random forest: classification
                Number of trees: 50
```

```
No. of variables tried at each split: 2
```

```
  OOB estimate of error rate: 4.67%
```

```
Confusion matrix:
```

|            | Setosa | versicolor | virginica | class.error |
|------------|--------|------------|-----------|-------------|
| setosa     | 50     | 0          | 0         | 0.00        |
| versicolor | 0      | 47         | 3         | 0.06        |
| virginica  | 0      | 4          | 46        | 0.08        |

We see from the above that the out-of-bag error estimate is 4.67%, and we see from the confusion matrix that a few cases have been misclassified. Following Liaw and Wiener (2002), we now combine all of the individual random forests into a single ensemble, where we note now the number of trees is equal to 150 (i.e., 50 + 50 + 50 trees):

```
> rf.all <- combine(rf1, rf2, rf3)
```

```
> print(rf.all)
```

```
Call:
```

```
  randomForest(formula = Species ~ ., data = iris, ntree = 50)
                Type of random forest: classification
                Number of trees: 150
```

```
No. of variables tried at each split: 2
```

The `getTree()` function extracts the tree from the forest. If we apply the function to the ensemble, we obtain the details on all the splits and the associated predictions (not shown):

```
> getTree(rf.all)
```

If we wanted to look at only a particular tree, we code the following, where we specify only the third tree (not shown):

```
> getTree(randomForest(iris[, -5], iris[, 5], ntree = 10), 3, labelVar=TRUE)
```

We can obtain a measure of importance for each variable by the following, which will report the extent to which there is a mean decrease in the Gini index as a result of the inclusion of the variable:

```
> data(iris)
> iris.rf <- randomForest(Species ~., iris, ntree = 50)
> iris.rf <- grow(iris.rf, 50)
> print(iris.rf)
```

```

Call:
  randomForest(formula = Species ~ ., data = iris, ntree = 50, norm.votes = FALSE)
      Type of random forest: classification
      Number of trees: 100
No. of variables tried at each split: 2

> importance(iris.rf)

              MeanDecreaseGini
Sepal.Length      10.096447
Sepal.Width        2.163171
Petal.Length      43.838533
Petal.Width       43.109449

```

We see above that petal length is the most important variable having the greatest mean decrease in the Gini index (43.83). Petal width is the second most important variable (43.10), with sepal width being the least important (2.16). We can plot the variable importance results (Figure 10.7):

```
> varImpPlot(rf.all)
```

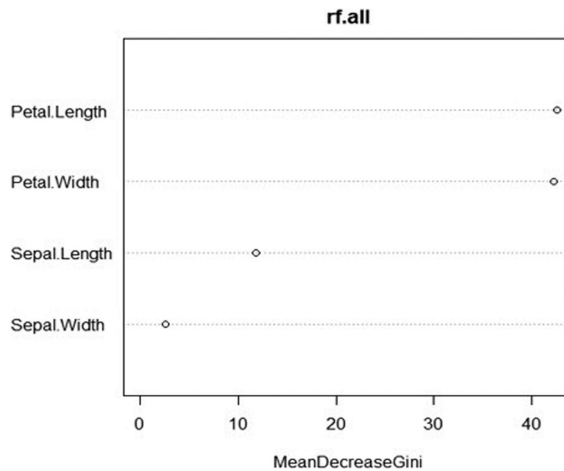


Figure 10.7 Mean decrease in Gini index by iris feature. Petal length results in the greatest mean Gini index decrease, sepal width, the least.

We can also get predictions of class by using the `predict()` function (we print only a few cases):

```

> predict(rf.all)

[1] setosa setosa setosa setosa setosa
[6] setosa setosa setosa setosa setosa

```

Random forests in Python can be fit using `RandomForestClassifier`.

## 10.7 Boosting

Boosting is another committee machine that again essentially operates on combining weak learners, analogous to how bagging averages many bootstrapped samples. However, boosting does this *systematically* and *deterministically*, not randomly. As explained by James et al. (2013, p. 321) in reference to boosting with regression trees:

Each tree [or single predictive tool, more generally] is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead, each tree is fit on a modified version of the original data set.

Boosting, in general, is what is known as a *slow learner*, which in general, performs better than fast or greedy learners. An analogy to boosting would be you and your friend drawing a picture. Suppose you get the picture started and your strengths are on full display, but also your weaknesses. For instance, suppose you are good at drawing the general shapes of objects, but not so good at filling in some of their more specific characteristics. After you are finished doing what you can, your friend steps in and focuses specifically on drawing the elements that you were poor at drawing, including giving specific attention to errors that you made in your contribution. The combination of both of you generates a rather strong painting even though you were both weak at drawing particular individual elements of the painting.

Another prototype example for understanding how boosting works is the spam e-mail paradigm described by Izenman (Izenman, 2008, p. 511). Suppose we wish to know whether an e-mail is spam or not before opening it. If the e-mail contains the word “money,” this by itself is a weak identifier because identifying it as spam might appear as random and misclassify the result. In other words, this is a *weak classifier*. Indeed, another classifier that is weak is one that classifies as spam if the e-mail contains the word “credit,” for instance. Taken on their own, weak classifiers are not very good and are bound to misclassify many cases. However, as was the case for bagging, *if we combine these weak classifiers, we may be able to substantially improve classification*. This is what is behind the idea of boosting. In this example, classification based on the word “money” will be imperfect and rather weak, but the second classifier trained will try to make up for the error generated by the first classifier. When combined, they make a pretty decent classifier. Boosting relies on the assumption that each of the weak hypotheses has accuracy that is just a bit better than chance levels. This assumption is generally referred to as the *weak learning condition* (Schapire, 2001).

We draw on James et al. (2013, p. 323) quite heavily as well as Bishop (2006, p. 658) in giving a formal, if not brief, summary treatment of boosting. Suppose in a regression tree problem  $f(x)$  is the estimated function. At the initial step of the algorithm, we will set  $f(x) = 0$  and residuals  $r_i = y_i$ , that is, residuals equal to the values of the response variable in the training data. We then fit a regression tree with  $d$  splits to the training data. We call this tree  $\hat{f}^b$ . We then update the original function  $\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x)$  with a shrunk (weighted) version of the new tree:

$$\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x) \quad (10.5)$$

where in (10.5)  $\lambda$  is a shrinkage parameter regulating the rate of learning, which notice is being applied to the new tree,  $\hat{f}^b$  that has  $d$  splits. Next, we update residuals by

$$r_i = r_i - \lambda \hat{f}^b(x_i) \quad (10.6)$$

This process is repeated for all  $b$  models and the final boosted model is produced, which is the sum of *all* weighted estimated functions,  $\hat{f}^b$ ,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x) \quad (10.7)$$

Hastie et al. (2009, p. 341) show that boosting is a simple example of an additive model and yet another instance of a *basis expansion*. There are many boosting algorithms in existence. However, one particular algorithm has historically dominated boosting, that of *AdaBoost*, short for *adaptive boosting* and generally used to classify into one of two binary categories (Schapire, 2001). The algorithm works by modifying probabilities of drawing instances as a function of error similar to the generic boosting algorithm described earlier (Alpaydin, 2020). AdaBoost has been found to be equivalent to forward stagewise additive modeling where an *exponential loss* function is minimized (Hastie et al., 2009). Gradient boosting, and extensions, are more general than AdaBoost where additional loss functions can be minimized. See Izenman (2008) for details.

### 10.7.1 Boosting Using R

In R, we can use the package *ada* (Culp et al., 2016) for AdaBoost for binary responses. A very popular package for gradient boosting in R is *gbm* (Ridgeway et al., 2024).

We adapt the example for boosting on the iris data in the *ada* package:

```
> library(ada)
> data(iris)
```

We will drop the *setosa* variable from the data:

```
> iris[iris$Species!="setosa",] ->iris
```

By dropping *setosa*, we will have removed 50 cases from the data set, so that the new data set should contain 100 cases. We can confirm this by requesting the dimension (`dim()`) of the new data:

```
> n <- dim(iris)[1]
> n
[1] 100
```

We assigned the dimension (`dim`) of the data because we are going to use `n` in what follows. We will sample 60 observations (i.e., `.6*n`; Culp et al., 2016, p. 5):

```
> trind <- sample(1:n, floor(.6*n), FALSE)
> trind
```

```
[1] 9 57 50 12 47 3 49 66 48 77 19 40 98 35 44 87
[17] 43 28 7 55 32 76 84 75 83 46 36 21 72 74 20 5
[33] 60 78 4 53 85 61 79 64 38 33 31 17 97 45 86 93
[49] 68 42 81 100 59 99 71 96 89 73 88 15
```

We now obtain the balance of what was not selected by `trind`, where the function `setdiff` stands for *set difference of subsets*:

```
> teind <- setdiff(1:n, trind)
> teind

[1]  1  2  6  8 10 11 13 14 16 18 22 23 24 25 26 27 29 30 34 37 39
[22] 41 51 52 54 56 58 62 63 65 67 69 70 80 82 90 91 92 94 95
```

We now create the associated factor levels as well as numeric variables:

```
> iris[,5]<- as.factor((levels(iris[,5])[2:3])[as.numeric(iris[,5])-1])
```

We are now ready to fit the trees. The following will fit eight-split trees (Culp et al., 2016, p. 5). We specify the number of iterations to equal 20 (`iter=20`) and set the shrinkage parameter (`nu`) equal to 1:

```
> gdis <- ada(Species~., data=iris[trind,], iter=20, nu=1, type="discrete")
> gdis
```

Call:

```
ada(Species ~ ., data = iris[trind, ], iter = 20, nu = 1, type = "discrete")
```

Loss: exponential Method: discrete Iteration: 20

Final Confusion Matrix for Data:

| True value | Final Prediction |           |
|------------|------------------|-----------|
|            | versicolor       | virginica |
| Versicolor | 33               | 0         |
| Virginica  | 0                | 27        |

Train Error: 0

Out-Of-Bag Error: 0.017 iteration= 6

```
train.err1  train.kap1
      9      9
```

We note the confusion matrix results in perfect classification. We can also see from the above output that the out-of-bag error is equal to 0.017 for iteration = 6. As iterations increase, overfitting becomes a concern. These results can be visualized to observe fluctuations on the training error across iterations (Figure 10.8). Note that by iteration 9, the training error has bottomed to 0, which agrees with the output above. We note as well that *kappa* (Figure 10.8), which measures the agreement between observed and predicted classification, is also maximized by iteration 9 (these figures are reported in the above output, `train.err1 = 9`, `train.kap1 = 9`), where `TRUE`, `TRUE` controls the reporting of training and kappa curves:

```
> plot(gdis, TRUE, TRUE)
```



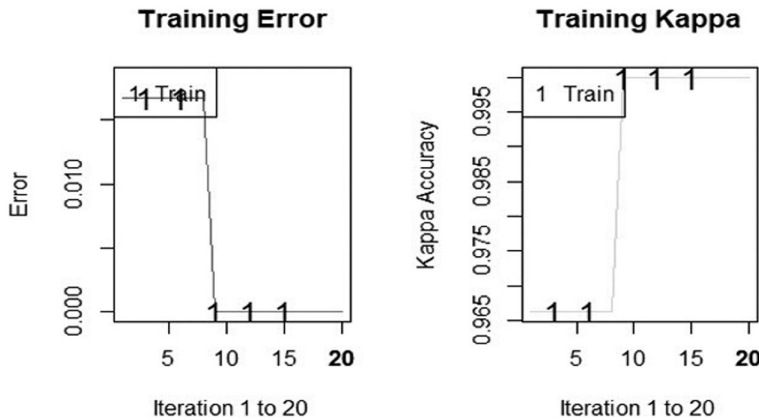


Figure 10.8 Training error (left) and training kappa (right) as a function of iteration. As the number of iterations increases, training error decreases, while kappa increases.

Now, we add the testing data and evaluate both training and testing error:

```
> gdis=addtest(gdis, iris[teind, -5], iris[teind, 5])
> plot(gdis, TRUE, TRUE)
```

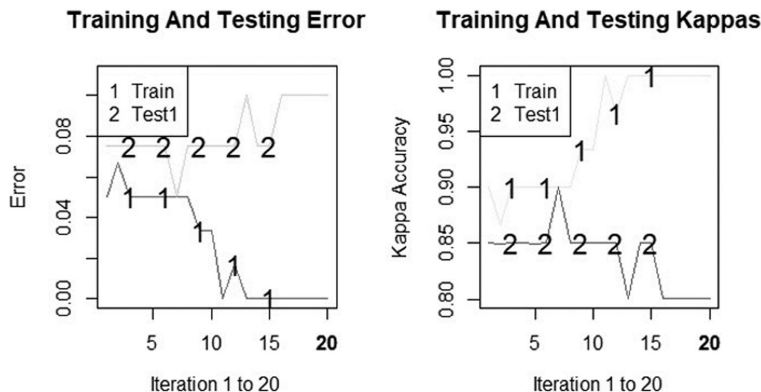


Figure 10.9 Testing and training error as a function of iteration (left). Training and testing kappas as a function of iteration (right).

We can see from the left plot in Figure 10.9 that the training error decreases while the testing error remains relatively constant until it elevates toward the end of the iterations, likely due to overfitting. The *kappa accuracy* reflects the extent to which there is agreement between the observed classification and the predicted classification. We see from the plot that kappa increases on the train data as the number of iterations increases, which is what we would expect, as the higher the kappa score, the greater the degree of agreement. Kappa decreases, however, on the test data as we near the final iteration, which is likely again a sign of overfitting.

Finally, we can obtain a variable importance plot as in Figure 10.10.

```
> varplot(gdis)
```

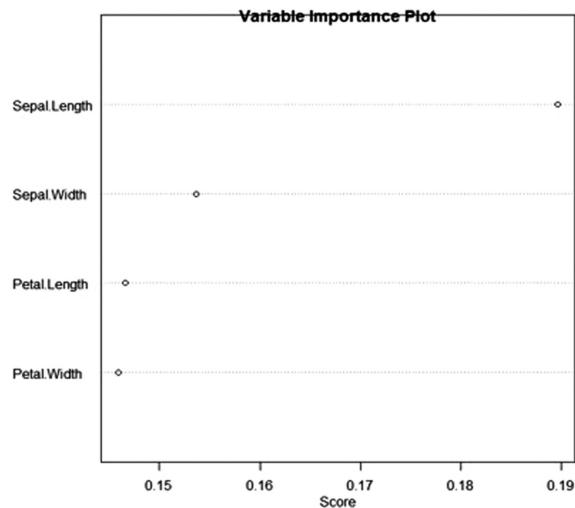


Figure 10.10 Variable importance measures for different features on the iris data.

We can see from Figure 10.10 that in the boosted sample, *sepal length* is most important, followed by *sepal width*. See Hastie et al. (2009, Chapter 10) for the computation of variable importance in this context. Re-running the model at various iterations may be useful both to regulate overfitting and also to observe changes in variable importance.

Whereas bagging and boosting combines weak “base” learners to produce a stronger learner, so-called *stacked models* seek to combine weak to strong learners into an even stronger one via optimizing the weighting of each model, a so-called *super learner*. This is generally referred to as *stacking* or *stacked generalization* (Hastie et al., 2009). Stacked learners will often outperform individual base learners. Breiman (1996, p. 51) summarizes the benefits of stacking:

The results show that stacking can produce predictors with substantially reduced prediction errors. Stacking never does worse than selecting the single best predictor. We note that the biggest gains came when dissimilar sets of predictors were stacked, i.e., subset selection and ridge regressions. The more similar the predictors, the less advantage there is in stacking.

In R, we can use the package *h2o* (Fryda et al., 2023) for model stacking. For details on stacked generalization, see Hastie et al. (2009).

## 10.8 Chapter Review Exercises

1. Distinguish between a *weak* vs. *strong* learner. Which, in general, is preferable? How are several weak learners potentially advantageous over fewer strong learners?
2. Describe the nature of a *decision tree*, both for classification and regression.
3. What does it mean to “overfit” a tree? How does *pruning* a tree reduce the possibility of overfitting?
4. Discuss the influence of the number of terminal nodes as it pertains to cost complexity pruning. How does the tuning parameter serve to regularize error?
5. Compare and contrast the *classification error rate* to the *Gini index*. How are they similar? Different?
6. Generate a decision tree on the iris data using the *tree* package in R using only features sepal length and sepal width. Report the *residual mean deviance* of the tree as well as the *misclassification error rate*. Are these numbers much different from the numbers reported for the decision tree generated in the chapter using all features?
7. Differentiate between *bagging* and *boosting*. How are they different? Similar?
8. Perform bagging in R on the *breast cancer* data, however using only *cell size* and *cell shape* as predictors in the model. Compare your results to those obtained in the chapter on the same data.
9. When are *bagging* and *random forests* equivalent?
10. Perform boosting on the iris data following the example in the chapter, however instead of dropping *setosa*, drop instead *virginica*. Summarize your findings.

# 11 Principal Components Analysis, Blind Source Separation, and Manifold Learning

In this chapter, we survey the method of *principal components analysis*, or “PCA” for short, as well as other related techniques. As we will see, principal components analysis can be interpreted as a special case of more general methods that generally fall under the name of *blind source separation* and *manifold learning*. PCA can be regarded, in fact, as a special case of manifold learning where in the case of PCA the manifold is *linear*. The more general manifold can take on nonlinear shapes and most often the phrase “manifold learning” will refer to *nonlinear dimensionality reduction*. Virtually all dimension-reduction techniques can be crudely classified into *linear* vs. *nonlinear*.

The motivation for PCA is to take a set of observed variables and generate linear combinations of those variables. Substantively, PCA is often advertised as being able to “discover” underlying dimensions of data. However, as we will learn in this chapter, the word “discovery” is potentially misleading and problematic. All that PCA can guarantee, for certain, is to derive linear composites of variables that maximize variance in the original variables. Whether those composites are meaningful in any substantive sense is not a question that can be answered by PCA and is only answerable by the user with reference to scientific theory and the domain of application. What is more, is that maximizing variance in observed variables may not result in dimensions that are in any way *interesting*. This is an important point, as there are many other competing techniques that claim to discover more interesting patterns in data and have their own criteria for doing so other than maximizing variance. In short, *maximizing variance in components does not necessarily imply those components will be of value or interesting in themselves*. They may, from a substantive point of view be entirely meaningless given current theory or application. Indeed, from a scientific point of view, the results of any dimension-reduction or manifold learning technique should be interpreted with caution as it is entirely possible that the given algorithm “causes” interesting geometric patterns or trends that in the end are not supported by the data (Meila & Zhang, 2023).

Principal components is only one of many techniques that seek to reduce the dimensionality of data. That is, PCA is *a* (i.e., only *one*) dimension-reduction technique; it is not *equivalent* to dimension reduction. In its most general sense, dimension reduction may be said to be taking observed data in a high-dimensional context and reducing that data to a lower-dimensional context but ensuring that it still conveys similar information (Nanga et al., 2021). That is, a good dimension-reduction technique should retain essential properties of the original data. If the information contained in the lower dimensions is misleading or at odds with the information at a higher dimension, then the reduction may lead to faulty interpretation. Hence, this is one important reason for understanding the nature of the dimension-reduction technique one is using as the conclusion provided by one technique may differ from the conclusion reached from another. *Dimension reduction is as much a function of the method as it is of the data*. A question worth asking is if the visualization of one technique is different from another, which then is most *correct*? The only answer to this question is that none are “correct”; they simply accomplish different things.

Most dimension-reduction techniques are nonparametric in nature (Murray, 2022), which essentially implies that they focus heavily on the training data often without attempting directly to generalize to out-of-sample data. These include techniques such as the aforementioned *PCA*, but also *classical multidimensional scaling*, *kernel PCA*, *Isomap*, *Laplacian Eigenmaps* (Belkin & Niyogi, 2003), and *t-distributed Stochastic Neighbor Embedding (t-SNE)*, *non-negative matrix factorization (NMF)*, among others. Alternative names for PCA include *Karhunen-Loeve transform* (Karhunen, 1947), *empirical orthogonal functions*, *common principal components analysis* (Flury, 1998), and others. The different names are largely due to the same method being developed in various disciplines or as special cases or extensions of classic PCA. For instance, the Karhunen-Loeve transform features the option of using class information on data before applying PCA to each class, then takes an average of class PCA (Alpaydin, 2020). This is similar to *common PCA* where similar components are hypothesized to arise in different data sets. Hence, not all names are *identical* to classic PCA, but at their core are still “PCA.” Once more with jargon in statistics and machine learning, one can know a given method well but fail to “recognize” it in a given field because it is named something different!

Many dimension-reduction techniques are justified based on a fundamental theorem, the *Johnson-Lindenstrauss lemma*. It holds that for a number of points in high-dimensional space, such points can be mapped to a much lower dimension with an upper limit to how much pairwise distances can be distorted. That is, the lemma essentially holds that dimension reduction can be performed while still preserving fundamental properties of the data. For further details on the lemma, consult Mohri et al. (2018, p. 354).

### 11.1 Dimension Reduction and Jargon

As we have mentioned, what can make dimension reduction challenging is that many of the same techniques have evolved in various scientific fields under somewhat different names and for different purposes. For instance, in psychometrics, one may wish to produce orthogonal linear combinations to represent attitudes of respondents, whereas in more physical applications such as images in information theory, this same idea translates to the *compression* of an image. The same technique can almost have unrecognizable and diametrically opposed applications. While this can be said of virtually any technique and not just dimension-reduction ones, the jargon is especially “rich” in dimension analysis.

However, just as is true for simple arithmetical operations such as addition, it is not *what* is being added that matters when it comes to understanding addition; it matters more in understanding what addition *does*. The technique must be studied, at least to some extent, from an abstract point of view exclusive of application. Only in that sense can one understand what the technique actually accomplishes without being “swayed” by what we “wish” it to accomplish to satisfy our *Freudian id* on this or that data set.

If dimension reduction is so vast a subject, why begin with PCA? We begin with PCA simply because it is generally considered the most common and oldest method of the family of dimension-reduction methods, having its origins with Karl Pearson in 1901. It is also the easiest (though deceptively simple) to understand and hence forms a good grounding for understanding other more advanced techniques as extensions or derivatives of essential PCA. Since the time of Pearson, several other dimension-reduction methods have been developed, PCA now considered only a *special case* of a wider family of blind source separation techniques (Izenman, 2008). Many of the more modern techniques feature latent variables which are analogous to hidden variables in neural networks (Izenman, 2008) and are also featured extensively in exploratory and confirmatory factor-analytic models as well as structural equation models. In its most general interpretation,

blind source separation involves separating out signals from noise, such that the signals extracted possess certain mathematically desirable properties, such as orthogonality in the case of classical PCA and many versions of exploratory factor analysis, or independence in the case of independent components analysis or independent factor analysis.

Principal components analysis can also be considered purely from an essential linear algebra point of view, equivalent to *best subspace approximation* using the singular value decomposition, or “SVD” for short (more precisely, *reduced SVD*). Singular value decomposition itself dates back to Beltrami (1873) and is a method for obtaining eigenvalues and eigenvectors from a matrix, mathematical objects that are at the very crux of PCA. SVD is able to decompose via factorization of a rectangular matrix  $\mathbf{X}$  such that

$$\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{W}^T \quad (11.1)$$

where  $\mathbf{X}$  in a statistical application is a data matrix of observations on a number of variables,  $\mathbf{V}$  and  $\mathbf{W}$  contain orthonormal columns (i.e., orthogonal and of length equal to one), and  $\mathbf{D}$  is a diagonal matrix (Jolliffe, 2002, p. 44). From the SVD, singular values and singular vectors can be obtained of even rectangular matrices, and hence SVD can be considered a pathway to generating principal components.

We can also conceptualize PCA as a *rotation*, such that PCA transforms or rotates empirical data by deriving new axes from which the data are projected onto these new axes. These new axes are the *eigenvectors* of the rotation. The *best basis theorem* states the best orthonormal basis is given by the leading eigenvectors of the transformation.

Whether one uses eigen analysis or SVD to compute principal components will result in the same components, though SVD is often preferred due to computational benefits (Murray, 2022). For extremely large dimensional problems, a *randomized SVD algorithm* (Murray, 2022) can be used to handle increasingly large dimensional data. In Python, for instance, *sklearn* features a randomized solver. PCA is also associated with the *Eckart-Young* and *Schmidt-Mirsky* theorems in mathematics as these both account for the mathematics that underlie the dimension-reduction technique.

*In modern statistics and machine learning, there are a whole host of dimension-reduction techniques. What unites them all is the attempt to “uncover” a lower-dimensional manifold from higher-dimensional data. That data could be attributes on a scale or an image that needs to be “decoded” by color, or even sound signals from a musical symphony.*

## 11.2 Deriving Classic Principal Components Analysis

As mentioned, classical PCA has its roots with Pearson in 1901. The term “principal component” was coined by Hotelling in 1933. Classical PCA is motivated as follows. Suppose we have a set of observed variables  $x_1$  through  $x_p$ . From these observed variables, we wish to produce linear combinations of these variables that have desirable statistical properties. Recall more generally the nature of a linear combination from our earlier review of linear algebra:

$$\ell_i = a_1y_1 + a_2y_2 + \cdots + a_py_p$$

where a linear combination  $\ell_i$  is a *weighted sum of products* and where  $a_1$  through  $a_p$  are the weights applied to observed variables  $y_1$  through  $y_p$ . For our data on  $x_1, \dots, x_p$ , the linear combination becomes

$$z_i = a_1 x_1 + a_2 x_2 + \dots + a_p x_p$$

where now  $z_i$  is the name of the linear combination and  $x_1$  through  $x_p$  are the observed variables. A principal component is a unique kind of linear combination such that the weights  $a_1$  through  $a_p$  correspond to values of allied eigenvectors extracted from an eigen analysis performed on the covariance or correlation matrix of variables  $x_1, \dots, x_p$ . Hence, if we denote the weights  $a_1$  through  $a_p$  by  $e_1$  through  $e_p$  to denote eigenvector weights, the first principal component is defined as

$$z_1 = e_1 x_1 + e_2 x_2 + \dots + e_p x_p \quad (11.2)$$

The first principal component possesses an important property; *it is the linear combination that accounts for the most variance in the observed variables*. What this means is that the variance of the resulting  $z_i$  will have *maximum variance* compared to any other linear combination that we could have derived from the data. If you think about this for a moment, you might agree that it is a favorable quality to have for a linear combination. It suggests that instead of needing  $x_1, \dots, x_p$  variables, perhaps we could account for a great deal of the variance in those observed variables by considering only the first principal component. That is, the first component makes up a large proportion of those observed variables.

However, the first principal component will likely not account for *all* of the variance in the variables. To account for more, we will have to consider the second and successive principal components.

More formally, if the total variance  $E | \mathbf{x} - \boldsymbol{\mu} |^2$  is defined by

$$E | \mathbf{x} - \boldsymbol{\mu} |^2 = \sum_{i=1}^p \text{var } x_i = \sum_{i=1}^p \sigma_{ii}^2 = \text{tr} \boldsymbol{\Sigma} \quad (11.3)$$

where  $\text{tr} \boldsymbol{\Sigma}$  is the sum of variances  $\sigma_{ii}^2$  along the main diagonal, PCA essentially constructs orthogonal variables (i.e., linear combinations) the variance of which, in totality, equal  $\text{tr} \boldsymbol{\Sigma}$ .

*The first principal component is that which accounts for maximal amount of the variance in the original variables as possible. Ensuing components will account for increasingly less variance.*

### 11.2.1 The Second Principal Component

The second principal component is that linear combination that accounts for maximal variance in the variables, as was true for the first component, but with an additional important property, that it accounts for variance *not captured by the first principal component*. How can we express this idea mathematically? Recall that in linear algebra, linear combinations that account for different sources of *information* (in a sense) more generally in the original variables are called *orthogonal*. Hence, the second principal component is that linear combination accounting for maximal variance in the observed variables but *orthogonal* to the first component:

$$z_2 = e_1 x_1 + e_2 x_2 + \cdots + e_p x_p$$

where now  $e_1, e_p$  represent elements of the second eigenvector. More precisely, the eigenvectors are orthogonal, the components are uncorrelated, though it is fine to use the terms “uncorrelated” and “orthogonal” interchangeably in this context so long as one is aware of the difference. We need not stop there! We can continue to derive principal components from our data each accounting for a successive amount of variance unaccounted for by prior components. Note that in our development of classical PCA, we have applied the technique to the entire training data at once. That is, we have assumed the entire training data is readily available. An alternative to this approach is known as *incremental PCA* (Lippi & Ceccarelli, 2019), where the updated covariance matrix  $\Sigma_{n+1}$  is based on the “old” covariance matrix,  $\Sigma_n$ . That is, incremental PCA adds new observations (the  $+1$  in  $n+1$ ) to the covariance matrix and thus has computational advantages over traditional PCA. The benefit is that not all data has to be saved in memory and the algorithm can be applied when the data set is expected to grow quickly or if the stationary data set is itself very large, too large to fit into memory at once (Lippi & Ceccarelli, 2019). Incremental PCA is available in Python in `sklearn` via `IncrementalPCA`.

### 11.2.2 PCA as a Least-Squares Technique and Minimizing Reconstruction Error

As is the case with most statistical methods, there is more than a single interpretation of principal components analysis. Our interpretation thus far has been via eigen analysis or SVD, but we can also interpret it as a least-squares technique similar to that of least-squares regression, with which we are already familiar. Instead of projecting from higher-dimensional space to lower-dimensional space such that we minimize the sum of squared errors in a response as in the typical ordinary least-squares regression, in PCA we also seek a projection, but this time from higher-dimensional space to lower-dimensional space such that the sum of squared *perpendicular distances* are minimized, where “perpendicular” is equivalent to *orthogonal*.

For example, in Figure 11.1 is a projection from two-dimensional space to a single *hyperplane* such that the orthogonal squared data distances from the principal component are minimized.



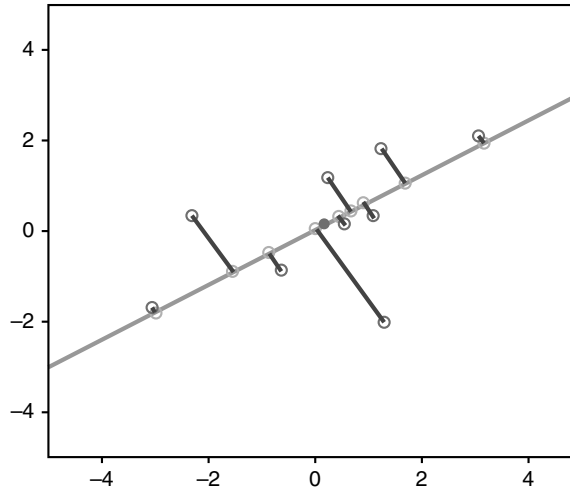


Figure 11.1 Hyperplane that minimizes squared orthogonal distances and maximizes variance in the original variables. Such a hyperplane is called a principal component.

In fact, in the typical PCA analysis, we seek to minimize what is known as the *reconstruction error* (Murphy, 2022). Not coincidentally, it can be shown that the reconstruction error is minimized when the resulting components are eigenvectors of an eigen analysis of the covariance matrix. Details of the derivation can be found in Bishop (2006) and Murphy (2022). The average reconstruction error is given by:

$$\ell(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{n=1}^N \|x_n - \mathbf{W}z_n\|^2 \quad (11.4)$$

where  $x_n$  are the original data points,  $z_n$  are the derived data points based on the projection, and  $\mathbf{W}$  is an orthogonal matrix. The optimal solution (i.e.,  $\ell(\mathbf{W}, \mathbf{Z})$ , which is a *loss*, is minimized) is obtained when  $\mathbf{W} = \mathbf{U}_L$ , where  $\mathbf{U}_L$  are the eigenvectors of a covariance matrix, and hence by definition for a symmetric matrix with distinct eigenvalues, orthogonal. Note that  $\mathbf{W}z_n$  is essentially trying to produce the least amount of deviation from  $x_n$  and under the most ideal of circumstances where the reconstruction of data based on the obtained eigenvectors (i.e., loosely, components) is perfect, then  $\ell(\mathbf{W}, \mathbf{Z}) = 0$ . Minimizing this reconstruction error is equivalent to maximizing the variance in the variables (Murray, 2022) which was the original interpretation of PCA we started out with. Again, as is the case with most statistical methods, there are a variety of different ways to derive the PCA technique, where in this case it can be shown that minimizing the reconstruction error is *mathematically equivalent* to maximizing the variance of the data projected by the components solution. Different ways of perceiving a statistical or machine learning technique from various angles and perspectives can pay dividends in truly understanding it in depth.

In addition to PCA being computed by eigen analysis or (truncated) SVD, components can also be gotten by the expectation-maximization (EM) algorithm and has advantages over eigen analysis such that it is often faster (Murphy, 2022).

### 11.2.3 Choosing the Number of Derived Components

Under most conditions of a well-behaved matrix, the number of components that we can obtain is equal to the number of observed variables subjected to the components analysis. That is, if  $x_1, \dots, x_p$  consists of say, 100 variables, then under usual (and most common) conditions, the number of components computed on these variables will equal 100. In brief, *a PCA on observed variables does not, by itself, reduce the information in the data*. In the case of the 100 variables, *choosing less than 100 computed components is where the actual data reduction occurs*, else all the PCA has done is disproportionately load most variance hopefully onto the first few components. What this means is that PCA is simply “packaging” (or *re-packaging*) the variance of observed variables into successive linear combinations. But it is not truly dimension “reduction” in an applied sense until we select less than the total number of components generated. This is the logic of PCA as a dimension-reduction technique. PCA is first and foremost only a *transformation*.

### 11.2.4 Why Reconstruction Error Is Insufficient for Choosing Number of Components

As a result of PCA attempting to account for as much variance in the variables as possible, when choosing the number of dimensions (components) to retain, one cannot simply use the *reconstruction error* or *variance explained* as a guide, since the more components one keeps, the more variance one will explain and the smaller will be the consequent reconstruction error. This is somewhat analogous to the effect of adding more predictors in a multiple regression – variance explained will virtually always increase (except in very unusual situations of zero correlation), and hence choosing the “best” regression model based on variance explained alone is not a good strategy. This limitation of using reconstruction error as a guide will also resurface when we consider cluster analysis later in the book and seek a measure to justify the number of clusters to retain. Since reconstruction error typically and necessarily reduces as the number of clusters increases, reconstruction error is not a sufficient measure for making decisions on how many clusters to retain.

To determine the number of components to keep, one should primarily use substantive judgment in addition to statistical indicators, and in the end, a bit of common sense. For example, if the first two components explain 70% of the variance, but the third component only explains an additional 1%, then obviously the two-component solution will dominate under *most* circumstances (depending on the substantive nature of the investigation). Graphical measures such as the *scree plot* can be used to visualize both the reconstruction error as well as the variance explained, and one typically looks for a “bend” in such plots to indicate where the reconstruction error (or variance explained) suddenly “drops.”

*PCA, under most circumstances, will generate as many components as there are variables. The actual “data reduction” occurs by selecting the first few components to account for as much of the variance in the variables as possible, or equivalently, minimizes the reconstruction error. Choosing the number of components to keep or maintain is largely dependent on the research or scientific context and often based on user judgment.*

### 11.2.5 Constraints on Components

Recall that we mentioned that the eigenvector weights  $e_1$  through  $e_p$  were applied to observed variables in order to generate components. In deriving these components, we shall impose the

constraint on these eigenvector weights so that the *sum of their squares is equal to unity*. That is, the sum of squared eigenvector weights we will set equal to one,

$$\sum_{i=1}^n e_i^2 = 1 \quad (11.5)$$

When we impose such a condition on each component, the component is said to have been *normalized*. There is no absolute requirement that the sum needs to be 1; it could in theory be any other constant positive value. Setting it equal to 1 implies that the length of the component is equal to unity. But why impose such a constraint at all? The purpose of this constraint is such that if we did not impose at least *some* constraint on the loadings, then in theory we could allow the component's variance to increase without bound. By imposing this constraint, or other positive constant for the variance, we are assured that the variance of the component will not simply be a reflection of increasingly large eigenvector weights.

As we have seen throughout this book, constraints such as the above are not atypical in statistical or machine learning methods. It is a regular occurrence in mathematical modeling more generally that some quantities are fixed or constrained to particular values so that other variable quantities can be more freely estimated under these constraints. Constraints such as these, in the end, *simplify problems*.

### 11.2.6 Centering Observed Variables

Observed variables are also often (though need not *necessarily* be) centered before being subjected to the components analysis, so that each variable has a mean equal to 0. By *centering* here we simply mean subtracting the mean of each observed raw variable. For example, if the observed variable is  $x_i$  with mean  $\bar{x}$ , then centering the variable means generating  $x_i - \bar{x}$  for each observation. The reader will recognize centering here as a precursor to computing *z*-scores where the centered deviation is then further divided by the standard deviation,  $s$ . As a simple example of centering, consider the following vector of data in R having original raw mean equal to 6.2:

```
> v <- c(0, 4, 8, 9, 10)
> mean(v)
[1] 6.2
```

To center this data, we simply subtract the mean from each observation:

```
> v.center <- v - mean(v)
> v.center
[1] -6.2 -2.2 1.8 2.8 3.8
```

We can then easily confirm that, within rounding error, the mean of the centered data is equal to 0:

```
> mean(v.center)
[1] -1.775489e-16
```

### 11.2.7 Orthogonality of Components

As mentioned already, one of the attractive features of components analysis is that derived components are guaranteed to be orthogonal (i.e., eigenvectors) and uncorrelated to one another. Recall what orthogonality means, it means derived components are at *right angles*, or *perpendicular* to one another (that of 90 degrees). From a substantive point of view, orthogonality ensures that each component accounts for, or *explains* a differing proportion of variance in the variables and that there is no overlap. It is important to emphasize that orthogonality is not the same as *statistical independence*. *Statistical independence is, in a sense, a stronger property than orthogonality in this case and principal components will only be both orthogonal and independent if the data are multivariate normal*. For components to be both orthogonal and independent beyond the assumption of multivariate normality one must go beyond traditional PCA and instead perform an *independent components analysis*, to be discussed later. Assuming all components are nonzero, however, orthogonality of components will imply *linear independence* (as opposed to *statistical independence*).

*Classic principal components analysis ensures orthogonality of components but does not guarantee their statistical independence. Statistical independence is the stronger property in this case, reserved for the technique of independent components analysis.*

### 11.2.8 Proportion of Variance Explained by Each Component: Covariance vs. Correlation Matrices

Having computed components in decreasing order of variance explained, a natural question worth asking now is: how much variance is actually accounted for by each component? Recall that components analysis does not *change* the amount of variance in the original variables. It merely reconstructs that variance along new dimensions (called the *components*, which are in effect, *eigenvectors* of the transformation). Hence, it stands that *the total variance in the totality of components should be equal to the total variance in the original variables*. That is, when we sum up the variance accounted for by all of the components, that sum should equal the total variance in the original variables. In a covariance matrix, that sum of variance will equal the sum of variances across all observed variables. In a *correlation matrix*, that sum will equal to the total number of *variables* because in a correlation matrix, the variance contributed by each variable is equal to one. Whether one analyzes a covariance or correlation matrix can result in quite vastly different outcomes. Principal components analysis is not invariant to type of matrix analyzed. This fact may at first seem to go against intuition, such that one might have assumed or expected that both the covariance and correlation matrices would yield similar results, or at minimum, one result would be a trivial transformation of the other. However, this is not the case. A PCA performed on a correlation matrix will usually yield *different* eigenvalues and eigenvectors than a PCA performed on a covariance matrix.

Given this distinction, the question becomes: which matrix to analyze? While some hold the covariance matrix is more in spirit with PCA (e.g., see Rencher & Christensen, 2012), others recommend that PCA should virtually always be performed on the *standardized* covariance matrix, otherwise known of course as the *correlation matrix* (Bilodeau & Brenner, 1999; Murray, 2022). Of course, there is no harm in trying both solutions and comparing them. The important point, however, is to recognize that performing PCA on one versus the other will, in general, not produce the same result. Principal components analysis also requires at least *some* covariance (or

correlation) among pairs of variables. In the absence of such covariance, that is, if all covariances or correlations are equal to 0 and only variances remain along the main diagonal, PCA will be of little use. At most, PCA will simply generate components that are a reflection of the original variables but in descending order of variance (Chatfield & Collins, 1980). For now, we will assume we are working with the covariance matrix. Given that the eigenvalues are variances of each component, then the following relation should then hold:

$$\sum s^2 = \sum \lambda_i \quad (11.6)$$

That is, the sum of variances of the original variables should equal the sum of eigenvalues from the components analysis. We can easily demonstrate this on empirical data. Consider some data on  $x, y$ :

```
> x <- c(2.5, 0.5, 2.2, 1.9, 3.1, 2.3, 2.0, 1.0, 1.5, 1.1)
> y <- c(2.4, 0.7, 2.9, 2.2, 3.0, 2.7, 1.6, 1.1, 1.6, 0.9)
```

The variances of each variable are the following:

```
> var(x)
[1] 0.6165556
> var(y)
[1] 0.7165556
```

The sum of these variances is

```
> sum(var(x), var(y))
[1] 1.333111
```

When we compute the components analysis on the covariance matrix from this data, we obtain via eigen analysis eigenvalues of 1.284 and 0.0491, which when summed total 1.3331, which we note is the same total as that generated by summing the original variances of the variables. Hence, we have demonstrated through empirical data that PCA has not *changed* the variance in the variables, it has merely *reconstructed* the variance onto new dimensions instead of on the original variables. *The variance accounted for by the new components is the same variance in the original variables.* This is why again we dislike the idea of PCA “discovering” dimensions. We prefer say that PCA is simply *reorganizing* the variables onto dimensions that are mathematically assured by the decomposition. For the majority of data sets and well-behaved matrices, PCA *will* generate components. It is whether such components can be *interpreted* and whether they are meaningful that is the *real* question of interest.

### 11.2.9 Principal Components as a Rotation of Axes

We have seen that principal components analysis can be explained by eigen analysis or by reference to a least-squares argument in which squared perpendiculars are minimized instead of vertical distances. The real “mystery” of principal components analysis lies in eigen analysis. Once we accept that eigen analysis can be performed, it is a relatively simple matter to accept that we can generate components. It is the substantive understanding of these components that then matters.

What we mean by this is that if one truly wants to understand PCA, one needs to truly understand eigen analysis.

Our understanding of PCA can also benefit from understanding some of its geometry. Geometrically, PCA is nothing more than a rotation of the original Cartesian axes. To demonstrate such a rotation, consider once more our data on PCA (Smith, 2002):

```
> x <- c(2.5, 0.5, 2.2, 1.9, 3.1, 2.3, 2.0, 1.0, 1.5, 1.1)
> y <- c(2.4, 0.7, 2.9, 2.2, 3.0, 2.7, 1.6, 1.1, 1.6, 0.9)
```

We now generate a dataframe from these vectors as well as the corresponding covariance matrix:

```
> data <- data.frame(x, y)
> data
```

|    | X   | y   |
|----|-----|-----|
| 1  | 2.5 | 2.4 |
| 2  | 0.5 | 0.7 |
| 3  | 2.2 | 2.9 |
| 4  | 1.9 | 2.2 |
| 5  | 3.1 | 3.0 |
| 6  | 2.3 | 2.7 |
| 7  | 2.0 | 1.6 |
| 8  | 1.0 | 1.1 |
| 9  | 1.5 | 1.6 |
| 10 | 1.1 | 0.9 |

```
> cov(data)
```

|   | X         | y         |
|---|-----------|-----------|
| X | 0.6165556 | 0.6154444 |
| Y | 0.6154444 | 0.7165556 |

Next, we perform an eigen analysis on this data to extract eigenvalues and eigenvectors from the covariance matrix:

```
> eigen(cov(data))
eigen() decomposition
$values
[1] 1.2840277 0.0490834
$vectors

      [,1]      [,2]
[1,] 0.6778734 -0.7351787
[2,] 0.7351787  0.6778734
```

We note the first eigenvalue is equal to 1.284 with eigenvector 0.6779, 0.7352 (the eigenvectors corresponding to each eigenvalue are in *columns* above, not rows). The second eigenvalue is equal to 0.0491 with eigenvector  $-0.7352, 0.6779$ . But what do these eigenvectors represent? We can consider them to be *factors of rotation* of the original axes (Young, 2023). The eigenvectors are first and foremost *vectors*, and as such, can be easily plotted in space (Figure 11.2).

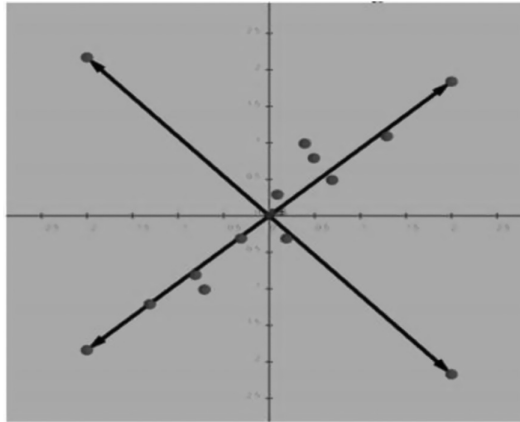


Figure 11.2 First and second principal components accounting for all of the variance in the variables. The two components are mutually orthogonal.

Note carefully that each eigenvector appears to account for the scatter of points. The first eigenvector, from bottom left to upper right, seems to account for *most* of the variance in the data. This is not a surprise, since the first eigenvalue is equal to 1.284 and is much larger than the second of 0.0491. The proportion of variance accounted for by the first eigenvector is  $1.284/(1.284 + 0.0491) = 0.96$ , or 96%. Clearly, the first eigenvector (component) is accounting for the majority of the variance in the original variables.

We can also see that the angle between the eigenvectors is one of 90 degrees. That is, the rotation of the original axes did not change the angle between them but instead kept it as orthogonal as were the original basis Cartesian axes. In linear algebra, this type of transformation is regarded as a *rigid rotation* of the axes and we have simply found an optimal new *basis* from which to project the data. Mathematically, this is all that PCA accomplishes.

#### 11.2.10 Principal Components, Discriminant Functions, Canonical Variates: Linking Foundations

As mentioned at the outset of this book, if there is a common theme running throughout most of it, one of many, it is that of *dimension reduction*. Even the analysis of variance, long before we knew it in undergraduate statistics, on a deeper theoretical level could be said to be generating new dimensions on which data can be projected. It may thus appear to the reader at some point that many of the methodologies surveyed in this book and beyond are performing a similar task. Indeed, dimension reduction is common to most in one way or another, though there are of course important differences regarding how various methods go about such a task.

Recall that we had said earlier in the book that discriminant functions were underlying dimensions such that they maximized separation between groups. Successive dimensions were

uncorrelated to one another. Both discriminant analysis and principal components analysis execute a “flavor” of dimension reduction, but they go about it in different ways. For one, in PCA, dimensions are obtained that maximize *variance*, not maximize *separation between groups*. Second, while discriminant functions are uncorrelated to one another, they are typically not orthogonal, while principal components are uncorrelated *and* orthogonal. Components have the stronger property of orthogonality (Rencher & Christensen, 2012). This is also true of canonical variates in canonical correlation; *they are uncorrelated but not orthogonal*. As mentioned earlier, components can be made independent (or nearly so), but that requires independent components analysis and not classical PCA. PCA only ensures that components are uncorrelated, which is a weaker property than statistical independence (Jolliffe, 2002, p. 396).

A second feature that distinguishes PCA from linear discriminant analysis (LDA) (and canonical analysis) is that obtaining components does not involve the computation of an inverse matrix. In discriminant analysis, we were required to obtain eigenvalues and eigenvectors from  $\mathbf{E}^{-1}\mathbf{H}$  and hence  $\mathbf{E}^{-1}$  had to exist. In PCA, no such inverse is computed. Of what consequence is this fact? Recall that inverses exist only for matrices that are *nonsingular*. A singular matrix has determinant equal to 0, which also implies or equates to linear dependence among select rows or columns. However, if the computation of an inverse is not required, then this implies that *PCA can be performed on a covariance matrix that is singular*. Under this condition some eigenvalues will equal 0 (Rencher & Christensen, 2012). In such a case, these components are to be ignored, as they explain zero variance. A second case in which singularity can arise is when the number of observations is less than the number of variables. PCA can still be performed on such data, whereas LDA or canonical analysis on such data would, analytically, be problematic.

### 11.2.11 Principal Components in Python

We consider a very easy example of PCA in Python on fictional data:

```
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[ -1, -1], [-2, -1], [-3, -2], [ 1,  1], [ 2,  1], [ 3,  2]])
pca = PCA(n_components=2)

X
Out [95]:
array([[ -1, -1],
       [-2, -1],
       [-3, -2],
       [ 1,  1],
       [ 2,  1],
       [ 3,  2]])

pca.fit(X)
Out [72]: PCA(n_components=2)

print(pca.explained_variance_ratio_)
[0.99244289 0.00755711]
```

We see from the above that the proportion of variance explained by the first component is 0.99, while the proportion of variance explained by the second component is 0.0075. Clearly, the first



component is most important to the analysis. Note that the cumulative sum of the proportions equals 1 (i.e.,  $0.99244289 + 0.00755711 = 1.0$ ).

As mentioned earlier, *incremental PCA* can be beneficial when the data are so large that it cannot fit in memory at once, and can be fit using `IncrementalPCA`. Heatmaps can also be used to visualize components (see Müller & Guido, 2017, pp. 148–149 for examples).

### 11.2.12 Principal Components in R

Conducting a basic PCA in R is very easy. For example, we can use the *Rdimtools* package (You & Shung, 2022) on the iris data:

```
> library(Rdimtools)
> data(iris)
> set.seed(100)
```

Below we will sample 50 observations from numbers 1 through 150:

```
> subid = sample(1:150, 50)
```

We then create the matrix *X* made up of variables 1 through 4 (in columns) in the iris data, which are the plant features (i.e., sepal length, sepal width, etc.), followed by the designation of species (i.e., label):

```
> X = as.matrix(iris[subid,1:4])
> label = as.factor(iris[subid,5])
```

We now conduct the PCA, requesting two dimensions (`ndim = 2`):

```
> PCA <- do.pca(X, ndim=2)
> PCA
$Y
```

|      | [, 1]       | [, 2]        |
|------|-------------|--------------|
| [1,] | 1.50866388  | -0.491302038 |
| [2,] | 1.90078112  | -0.156616064 |
| [3,] | -2.65041514 | -0.245114259 |
| [4,] | 1.18816125  | 0.114116172  |
| [5,] | 0.13988863  | -0.543760253 |

The above are the component scores resulting from the PCA (only the first five scores are shown). Next, we are given the variances associated with each component:

```
$vars
[1] 3.8646558 0.2000033
```

```
$projection
```

|      | [,1]        | [,2]        |
|------|-------------|-------------|
| [1,] | 0.37088025  | 0.63842394  |
| [2,] | -0.08806873 | 0.72546180  |
| [3,] | 0.84794544  | -0.24184196 |
| [4,] | 0.36834802  | 0.08736427  |

We can see that the variance of the first component is 3.86 while the variance of the second component is 0.200. The component weights (eigenvectors) are given next which are to be read vertically for each component. We can obtain a visualization of the projected component scores (Figure 11.3, left):

```
> plot(PCA$Y, pch = 19, col = label, main = "PCA")
```

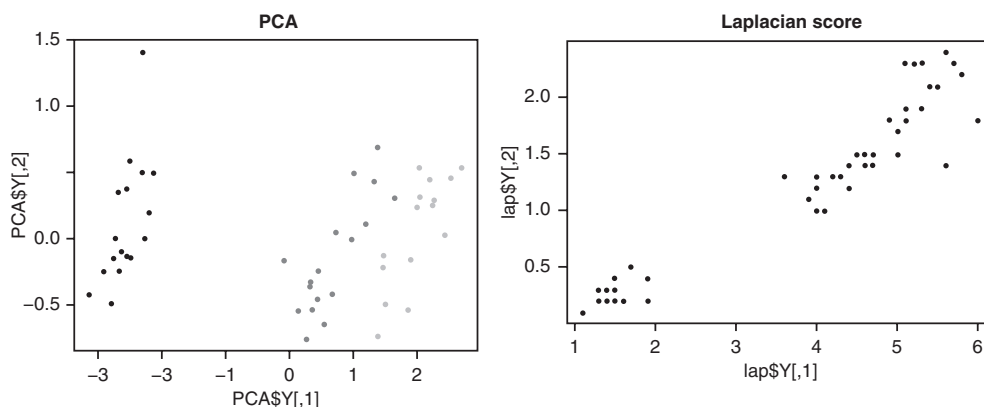


Figure 11.3 Principal component scores (left) and Laplacian score (right) on the same data, revealing quite different patterns. Dimension reduction using variable techniques will often generate quite disparate results.

Incidentally, we encourage the reader to try a competing alternative to the above PCA simply to get a glimpse and appreciation of how drastically the dimension reduction (or “embedding,” a term we will use shortly) can change as a function of method employed for the data reduction. For example, had we conducted a *Laplacian Score*, we would have obtained the plot to the right of the above PCA (Figure 11.3, right). Instead of `do.pca()`, we code `do.lscore()`:

```
> lap = do.lscore(X, ndim=2)
> plot(lap$Y, pch = 19, main = "Laplacian score")
```

Clearly, as is evident from the two different methods of dimension reduction, the expression “*what you find in data is largely a function of what you use to do the finding*” rings true!

Curvilinear component analysis is yet another option and can be fit by R with the package *ProjectionBasedClustering* (Thrun & Ultsch, 2021).

### 11.2.13 Cautionary Concerns and Caveats Regarding Principal Components

Principal components analysis can seem like a relatively simple procedure, but it has a lot of hidden complexities associated with it that often bypass the novice. In statistics, PCA is probably the simplest procedure that has the most complexity associated with it and things that can “trip you up” if you’re not careful. *Its simplicity is deceiving*. Understanding PCA is actually quite challenging, which might explain why there are entire books devoted to the subject. The graduate student learning PCA for the first time, concluding it is an “easy” procedure simply lacks the experience to appreciate its “latent” (pun intended!) complexity. We list a few of the more common caveats below. These are things you should be well aware of before using or interpreting the technique. Some we have already mentioned elsewhere in the chapter, and we emphasize here again only to reiterate their importance:

1. *Principal components are not scale-dependent.* That is, whether the PCA is performed on the covariance or correlation matrix usually results in different findings. Variables with greater variance will have greater influence on the PCA solution than variables with lesser variance. The typical way to get around this issue is to scale variables so that they all have the same variance, which amounts to analyzing the correlation matrix instead of the covariance matrix. While this may seem like a good solution to be adopted in all circumstances, if variables are not originally conceived to be of equal importance (almost in the sense of a Bayesian prior), then the covariance matrix should be analyzed over the correlation matrix (Chatfield & Collins, 1980, p. 71) as the analysis of covariance matrices is more in the spirit of PCA (Rencher & Christensen, 2012).
2. *The first principal component, as mentioned, is the one with the largest variance.* However, this does not necessarily imply that it is the component of *most interest*. There is no guarantee that the first principal component will be more theoretically interesting than the second, or the third, etc. It is the “best” component only in the sense of maximizing variance. Other components may be preferred based on their correlation with other variables or how they perform in ensuing regression analysis for instance, or even how they cluster in a cluster analysis (Marden, 2015). Variance explained is only one criterion that should be used in addressing the “importance” or “relevance” of a component.
3. *Principal components will under most circumstances generate as many components as there are original variables.* PCA, under most circumstances, will necessarily generate components. However, there is absolutely no requirement that the components obtained are meaningful or useful. The fact that you ran a PCA on variables and obtained components is not a “finding.” It is a mathematical *certainty* that existed long before you performed the components analysis. Whether you believe the transformation is in some sense “interesting” will likely depend on your substantive science and even philosophy of mathematics. That is, if the components look “meaningless” now substantively, does the mathematical transformation still imply they are nonetheless “real?” This isn’t a statistical question, it is one of the *philosophy of mathematics*. If you believe mathematics corresponds to “reality,” then regardless of whether you can “make sense” of the components or not you may still believe them to nonetheless “exist.” This is an exceptionally interesting topic, but far beyond the scope of this book.
4. *Related to #3, components are only as good or interesting as the input variables subjected to the components analysis.* Hence, the first and primary concern of interest is subjecting PCA to a set of variables that is inclusive of all the variables you wish to analyze. If the data set is not interesting or you leave out important variables from the ensemble of variables, conducting a PCA on such data will likely be limiting, or your components analysis will be misspecified.

5. *It is possible, even if unlikely, that a PCA generates eigenvalues that are equal or near equal in size.* Further, population eigenvalues are usually not exactly equal to eigenvalues generated by the sample. Inferential tests for eigenvalues exist (e.g., Camargo, 2022), but are rarely used. Tests have been developed and usually require the normality assumption (Collins & Chatfield, 1980; Peres-Neto et al., 2005).
6. Components derived in PCA depend more on the *ratios* of correlations in a correlation matrix than the correlations themselves. As noted by Collins and Chatfield (1980, p. 67), dividing all off-diagonal correlations by a constant greater than 1 reveals that the eigenvalues change but that the eigenvectors, which implies the components as well, do not.
7. PCA does not assess the dimensionality of the variables; it assesses the *linear dimensionality* of the variables. If the data are nonlinear, which amounts to the underlying manifold being nonlinear, then classical PCA will fail to capture this nonlinearity. As discussed later, techniques such as *kernel PCA* will be more effective at uncovering a nonlinear underlying manifold.
8. Related to the previous point (#7), PCA is, in its most general sense, a type of *manifold learning* where the manifold is assumed to be linear. Nonlinear techniques may be more effective at uncovering the true manifold. *A failure to uncover meaningful components using PCA should never be taken as conclusive evidence that deeper dimensionality is nonexistent underlying the variables.* Such interesting dimensionality may indeed exist, but it may be *nonlinear* in form, a structure that PCA will be limited in uncovering.

### 11.3 Independent Components Analysis

In the classical PCA just discussed, we required components to be linear functions and orthogonal to one another. We sought to reduce the rank of the matrix of observed variables into one of smaller rank, but that accounted for most of the variance in the original variables, preferably in the first few components. The fact that the components were orthogonal to one another was a key feature of principal components analysis.

In *independent component analysis* (ICA), we require components (i.e., the *latent variables*) to be mutually independent and typically non-Gaussian (Izenman, 2008). That is, the goal in ICA is to find a linear representation of non-Gaussian data so that the components are statistically independent, and if not perfectly so, then *as independent as possible*. *This is a different requirement than simply requiring orthogonality.* As noted by Jolliffe (2002), the links between classical PCA and independent components analysis are, perhaps surprisingly, not that strong, and ICA is closer in relation to factor analysis than it is to components analysis.

The ICA model (Izenman, 2008) can be stated as follows:

$$\mathbf{X} = f(\mathbf{S}) + \mathbf{e} \quad (11.7)$$

where  $\mathbf{X}$  is a vector of random observed “manifest” variables,  $\mathbf{S}$  is an unobservable random vector variate of sources whose components are independent latent variables, each having zero mean, and  $\mathbf{e}$  is a random vector representing noise, for instance, in the measurements taken on  $\mathbf{X}$ . The key point is that the independence of the sources implies that each individual source signal is hypothesized to be generated by a process unrelated to any other source signal (Izenman, 2008). Without  $\mathbf{e}$ , the model is referred to as *noiseless ICA*. When  $\mathbf{e}$  is included, we refer to the technique as *noisy ICA*.

In ICA, projected points  $\mathbf{P}$  are obtained as a product of the original data  $\mathbf{D}$  and an unmixing matrix  $\mathbf{M}^u$ , where the mixing matrix is equal to  $\mathbf{M}$ . Hence,  $\mathbf{P} = \mathbf{D}\mathbf{M}^u$ . ICA essentially looks for rotations of data that have (approximately) independent coordinates. This will necessarily be true for all rotations from samples from multivariate normal distributions (Venables & Ripley, 2002, p. 313), and hence ICA will not be of most interest for such a “trivial” situation. Instead, ICA is most valuable and useful for data arising from distributions that are not normal, that is, *non-Gaussian*.

### 11.3.1 Principal Components vs. Independent Components Analysis

How does independent component analysis relate to PCA? It stands that there should be some relation between them, and indeed there is. PCA can be considered a *special case of ICA*, where PCA yields orthogonal components while ICA generates independent components.

Jolliffe (2002, p. 396) best summarizes the difference between PCA and ICA:

PCA has as its main objective the successive maximization of variance, and the orthogonality and uncorrelatedness constraints are extras, which are included to ensure that the different components are measuring separate things. By contrast, independent components analysis (ICA) takes the “separation” of components as its main aim. ICA starts from the view that uncorrelatedness is rather limited as it only considers a lack of linear relationship, and that ideally components should be statistically independent. This is a stronger requirement than uncorrelatedness, with *the two only equivalent for normal (Gaussian) random variables* [emphasis added]. ... This is much closer to the set-up for factor analysis, and it is this “model” that ICA generalizes.

Jolliffe goes on to emphasize that simply because ICA is most appropriate in non-Gaussian contexts, this does not imply that classical PCA assumes normality, since it does not. One need not make any distributional assumptions to conduct a PCA as it is simply a mathematical decomposition of variance. If one chooses to make inferences of components to populations, then that is a different matter that would require corresponding parametric assumptions.

Hence, whereas PCA seeks to find or uncover a reduced-rank representation of variables, ICA attempts to represent data as independent sub-elements. ICA tries to separate the *signals* in one’s data, while PCA attempts to “load” the data onto the first few important or “principal” components. These are two quite different objectives. The reader will better appreciate how ICA is more similar to factor analysis when we survey factor analysis later in the book. Factor analysis is a proper statistical *model* and thus includes an error term, making it comparable to ICA. In PCA, no such error term is typically included nor is any statistical “model-like” structure advanced.

*PCA and ICA are not the same technique. PCA seeks a lower-dimensional linear representation of the variables, whereas ICA seeks to separate the observed data into independent signals. Independence is a stronger property than uncorrelatedness under the assumption of non-Gaussian distributions.*

As a simple example of independent components analysis, consider Figure 11.4.

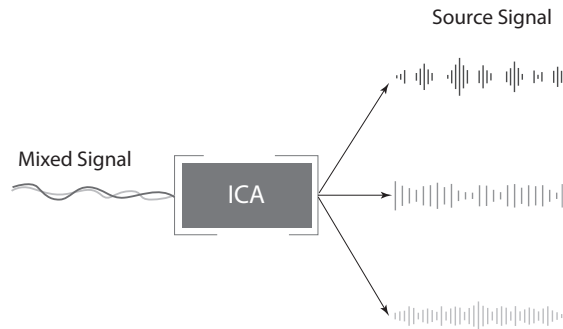


Figure 11.4 The objective behind independent components analysis is to “unmix” signals in a data set such that the resulting “source signals” are as independent as possible.

We begin with a *mixed signal*. The goal of ICA is to essentially “unmix” the signals. In sound analysis, for instance, a variety of components make up any sound, and ICA has been found useful for unmixing those composite sounds (e.g., “black box” recordings following an airplane crash). Independent components analysis can also prove useful in “unmixing” mixtures of image components in a picture.

For further details on ICA, see Izenman (2008). For an excellent discussion that summarizes the primary features of ICA and elaborates greatly on the required assumptions (and why they are necessary), see Hyvärinen and Oja (2000). For a concise discussion of the differences between PCA and ICA, see Jolliffe (2002, p. 396). *Kernel ICA* is a relatively recent development (Bach & Jordon, 2002) at least in comparison to classical PCA and is discussed in Izenman (2008, pp. 575–576). Independent components analysis can be fit using Python’s `FastICA` algorithm. In R, the package `ica` (Helwig, 2022) can perform ICA as well as the package `e1071` (Meyer et al., 2023).

We feature an example in R from the package *ProjectionBasedClustering* (Thrun et al., 2024), where the Hepta data (three-dimensional) contain pre-defined clusters but with different variances:

```
> library(ProjectionBasedClustering)
> data(Hepta)
> Data = Hepta$Data
> Proj=ICA(Data)
Centering
Whitening
Symmetric FastICA using logcosh approx. to neg-entropy function
Iteration 1 tol=0.728745
. . . . .
Iteration 12 tol=0.000065
> Proj
```

Below are the results of the projection (`$ProjectedPoints`) as well as the mixing (`$Mixing`) and unmixing (`$Unmixing`) matrices:

```
$ProjectedPoints

      [,1]      [,2]
[1,] 0.0392087884 0.043628947
[2,] 0.0058800277 0.047013653
```

```
[3,] 0.0549427049 0.040218053
[4,] 0.0243936200 0.015991544
[5,] 0.0476893068 0.022093236
```

```
$Mixing
```

```
      [,1]      [,2]      [,3]
[1,] -1.1748570 -1.169347 0.1137517
[2,] -0.8914634 1.022500 0.9786126
```

```
$Unmixing
```

```
      [,1]      [,2]
[1,] -0.5243656 0.8514932
[2,] -0.8514932 -0.5243656
```

```
$PCMatrix
```

```
      [,1]      [,2]
[1,] -0.05070429 0.5361333
[2,] 0.52604976 0.1678445
[3,] 0.27427254 -0.2228086
```

We can obtain a plot (in color) of projected points (Figure 11.5) for each of the seven pre-defined clusters (Hepta\$Cls). The reader may wish to contrast these results with those that would have been obtained using a PCA (`Proj.pca = PCA(Data)`) then (`PlotProjectedPoints(Proj.pca$ProjectedPoints, Hepta$Cls)`)

```
> PlotProjectedPoints(Proj$ProjectedPoints, Hepta$Cls)
```

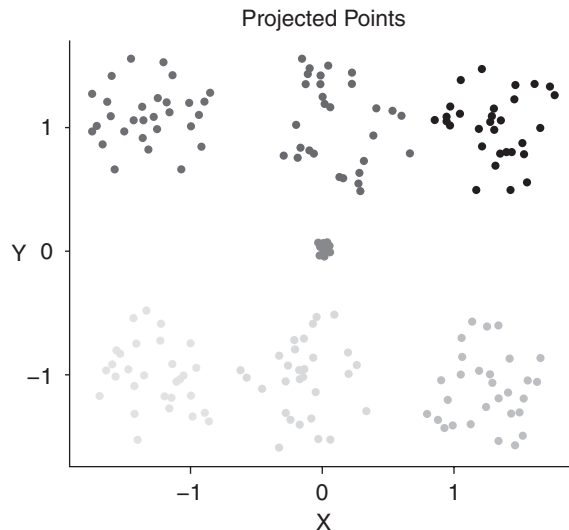


Figure 11.5 Projected points resulting from independent components analysis on seven well-defined clusters.

The `ProjectionBasedClustering` package contains several additional clustering and dimension-reduction options and should be consulted for further details.

## 11.4 Probabilistic PCA

In classical PCA, there is an absence of an associated probabilistic model for the data. That is, classical PCA is not probabilistic in any way; it is merely a mathematical transformation of the variables onto new axes such that it establishes a new *basis* (i.e., one that is orthogonal) from which the data are projected. As many writers on PCA have emphasized, PCA is not a “model” per se, it is simply a rotation of axes on which data are projected in correspondence to this rotation. Components are not obtained with reference to any kind of density-estimation framework. Be sure to note that in this distinction we are referring to the *construction of the components*, not the *estimation* of the components in the population. As mentioned, inferential procedures for components analysis do exist but are rarely used, with users preferring visualization techniques (e.g., scree plots) and eigenvalue magnitudes as aids to determining how many components to retain. When inference is employed, the inferential aspect of PCA makes components analysis probabilistic, but only in that particular sense of inference, not in the sense of actually deriving the components via eigen analysis.

It is possible, however, to derive the PCA approach based on a probabilistic formulation even without regard to estimation. Probabilistic PCA was first proposed by Tipping and Bishop (1999) to provide an alternative approach to PCA, one that is probabilistic and thus exists in a density-estimation framework (Tipping & Bishop, 1999).

We give the case of *probabilistic PCA*, which can also be regarded as a special case of factor analysis, where  $\mathbf{W}$  is a loading matrix and where the covariance matrix  $\mathbf{S}$  can be written as

$$\mathbf{S} \sim \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I} \quad (11.8)$$

Notice the noise term  $\sigma^2$  in the probabilistic model. Classic principal components analysis is a special case of probabilistic PCA as  $\sigma^2 \rightarrow 0$ , or at minimum is infinitesimally small and essentially converges to zero. Noise dimensions are also uncorrelated, which above is indicated by  $\sigma^2\mathbf{I}$ , since in the identity matrix only values of 1 lie along the main diagonal and zeroes everywhere else. The inclusion of  $\sigma^2\mathbf{I}$  as the “error term” here instead of only  $\sigma^2$  simply denotes the fact that we are working in the context of several variables and hence require a matrix representation.

The principal axes of probabilistic PCA can be derived as maximum-likelihood parameter estimates computed by the usual eigen-decomposition approach. There is no closed-form solution and hence parameter estimates for  $\mathbf{W}$  and  $\sigma^2\mathbf{I}$  must be obtained iteratively (Tipping & Bishop, 1999). We can also approach the construction of components via an iterative, computationally efficient expectation-maximization approach. For details, see Murphy (2022). For an excellent introduction to the probabilistic PCA model from the ground up starting with classic non-probabilistic PCA, see Tipping and Bishop (1999) who provide a very good overview.

### 11.4.1 Motivation for Probabilistic PCA

Why seek out a probabilistic approach to PCA in the first place? Why add a noise term similar to how factor analysis includes an error term? There are at least a couple benefits to doing so over the simpler and more conventional PCA approach:

1. Using a probability model allows us to extend the scope of conventional PCA and may also be useful in cases of missing data. It also allows for a comparison with other probabilistic approaches which dominate statistics, facilitating statistical testing and allowing for the application of Bayesian approaches (Tipping & Bishop, 1999).



2. Probabilistic PCA can be utilized as a constrained Gaussian density model (Tipping & Bishop, 1999). This allows for the efficient computation of maximum-likelihood estimates directly from the data principal components.

#### 11.4.2 Probabilistic PCA in R

Probabilistic PCA can be performed in R using the *Rdimtools* (You & Shung, 2022) package. The following is an example, adapted from the package (p. 184), that compares probabilistic PCA (PPCA) to PCA on the iris data:

```
> library(Rdimtools)
> data(iris)
> set.seed(100)
> subid = sample(1:100, 50)
> X = as.matrix(iris[subid,1:4])
> label = as.factor(iris[subid,5])
```

The above samples 50 observations from the data 1 through 100 (i.e., `subid = sample(1:100, 50)`), denotes variables 1 through 4 as `X` and the `label` as the fifth variable. We print a few of the first observations for `X` and `label`:

```
> head(X)
```

|     | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|-----|--------------|-------------|--------------|-------------|
| 102 | 5.8          | 2.7         | 5.1          | 1.9         |
| 112 | 6.4          | 2.7         | 5.3          | 1.9         |
| 4   | 4.6          | 3.1         | 1.5          | 0.2         |
| 55  | 6.5          | 2.8         | 4.6          | 1.5         |
| 70  | 5.6          | 2.5         | 3.9          | 1.1         |
| 98  | 6.2          | 2.9         | 4.3          | 1.3         |

```
> head(label)
```

```
[1] virginica virginica setosa versicolor versicolor versicolor
Levels: setosa versicolor virginica
```

We will now compare PCA and PPCA, on each requesting two dimensions:

```
> PCA <- do.pca(X, ndim=2)
> PPCA <- do.ppca(X, ndim=2)
```

Partial output from the PPCA now follows:

```
> PPCA
$Y
```

```
      [,1]      [,2]
74  6.247072 -5.342443
89  5.546579 -5.265592
78  6.856560 -5.837714
23  2.072220 -5.510457
86  6.071263 -5.771774
70  5.390608 -4.907913
 4  2.579355 -5.058637
55  6.407041 -5.611614
95  5.679656 -5.022154
```

```
$projection
```

```
      [,1]      [,2]
[1,]  0.3743456 -0.62842449
[2,] -0.1582456 -0.76536816
[3,]  0.8560253  0.13827068
[4,]  0.3194438 -0.01324609
```

```
$mle.sigma2
```

```
[1] 0.02567518
```

```
$mle.W
```

```
      [,1]      [,2]
[1,]  1.0820387 -0.119828001
[2,] -0.4574058 -0.145940423
[3,]  2.4743245  0.026365458
[4,]  0.9233460 -0.002525764
```

The above weights  $\mathbf{W}$  (`$mle.W`) correspond to the loading matrix weights discussed earlier. The value for  $\sigma^2 \mathbf{I}$  is given by `$mle.sigma2` (i.e., sigma squared, which corresponds to a variance).

We now visualize the results:

```
> opar <- par(no.readonly=TRUE)
> par(mfrow=c(1,2))
> plot(PCA$Y, pch=19, col=label, main="PCA")
> plot(PPCA$Y, pch=19, col=label, main="PPCA")
```

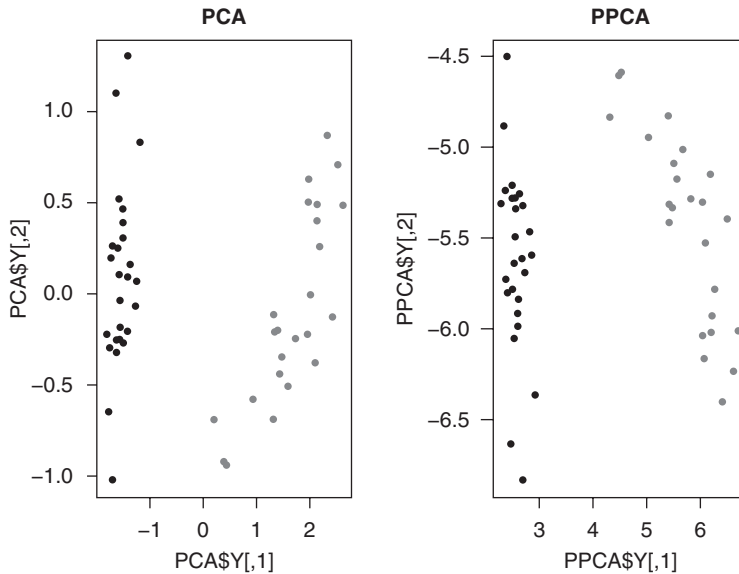


Figure 11.6 Principal components analysis (left) vs. probabilistic principal components analysis (right) on the same data.

We see (Figure 11.6) that both methods distinguish two components, yet the distributions are somewhat different.

### 11.5 PCA for Discrete, Binary, and Categorical Data

Up to now, we have assumed that the data subjected to PCA is numerical and relatively continuous in nature. This is fine, so long as the data in one’s study adheres to this nature. However, oftentimes data come in a variety of forms that are hardly continuous. For example, binary (e.g., yes vs. no) data are common in polling, and categorical data (e.g., A, B, C) are quite popular as well.

Can PCA be used for data of different types? According to Jolliffe (2002, p. 339), the answer is “yes”:

When PCA is used as a descriptive technique, there is no reason for the variables in the analysis to be of any particular type ... the variables could be a mixture of continuous, ordinal or even binary (0/1) variables. It is true that variances, covariances and correlations have especial relevance for multivariate normal data  $\mathbf{x}$ , and that linear functions of binary variables are less readily interpretable than linear functions of continuous variables. However, the basic objective of PCA – to summarize most of the “variation” that is present in the original set of  $p$  variables using a smaller number of derived variables – can be achieved regardless of the nature of the original variables.

While PCA can be conducted on a mixture of variables, *multiple correspondence analysis* (MCA) is usually the preferred technique for dimension reduction using categorical data, as PCA is most often used for dimension reduction featuring numerical measurements. Kalantan and Alqahtani (2019) also feature a method of PCA for mixed data. Murphy (2022, p. 670) gives the case of PCA for categorical data.

## 11.6 Nonlinear Dimension Reduction

### 11.6.1 Kernel PCA

As we have seen, classical or “standard” principal components analysis allows for linear dimensionality reduction. However, we must ask the question of how often the true dimensionality of the variables or data lies in a subspace that is precisely linear. Probably not that often just as most data subjected to regression models are hardly linear, which is why techniques such as polynomial regression, splines, and other nonlinear and nonparametric approaches were surveyed earlier in the book. Though linearity is a useful *approximation* for dimension reduction, the true underlying dimensionality of most sets of variables is probably *nonlinear*. That is, the true manifold is likely, in the end, a *nonlinear* structure, not a linear one. We emphasize again that the linear vs. nonlinear underlying dimensionality always refers only to the set of variables you are working with and have included in your set of chosen variables. Dimension analysis, as is the case with any mathematical technique, can only operate on the set of variables you have subjected to analysis. All analyses are on what might be called *restricted universes* of variables and their respective interrelationships. This seems like an obvious point to make, but is one worth emphasizing: *a PCA, or any statistical method can only provide results on the variables on which it is analyzed! If you leave out important variables, your entire analysis will be misspecified. What you “find” is entirely related to what you supply the “machine” doing the searching. All is context-dependent all of the time.*

*Kernel principal components analysis* (KPCA) allows one to generalize standard PCA to *non-linear dimensionality reduction*. As we will see, it is one of many nonlinear options to choose from when it comes to uncovering nonlinear dimensions.

### 11.6.2 How KPCA Works

Recall that the big idea of using kernel methods in general, not just for PCA (but for example, support vector machines), is that data that may not be separable in its current feature space may benefit from being projected into higher dimensions. For example, instead of having variables  $x_1$  and  $x_2$  only, an enlargement of the feature space might be  $x_1, x_2, x_1^2, x_2^2$ . *This enlargement of the feature space makes the data more linearly separable.* This is the very essence of what kernel methods, and kernel PCA in particular, seek to accomplish. Through the calculation of a kernel, we can then apply PCA in higher-dimensional space without ever having to really go there. When you think about it, this is truly a marvel! Indeed, kernel methods can be applied to many algorithms that can be formulated in terms of inner products in the feature space.

*By projecting data that is nonlinear into higher-dimensional space, we make otherwise non-linear data in the data space more linearly separable in the feature space. This is the essence of how kernel PCA (and other kernel methods) work.*

To further unpack how kernel PCA may be useful, consider Figure 11.7 consisting of data in two dimensions. Clearly, the data are not linearly separable. That is, we cannot draw a line through the data to separate the circles from the crosses. A more complex decision surface would provide much better separability. Mapping the problem from two dimensions to higher, we make the problem linearly separable, and it now looks as in Figure 11.8.

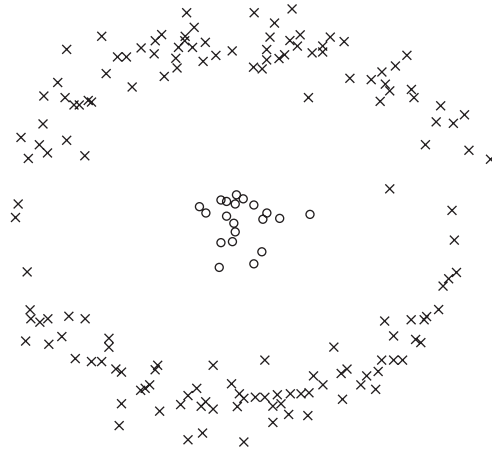


Figure 11.7 Data in two dimensions for which the data are obviously not linearly separable.

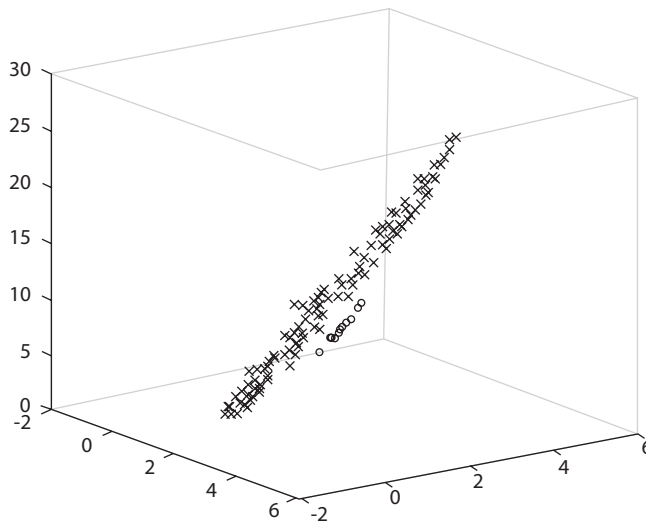


Figure 11.8 Mapping data to higher dimensions via kernels makes it more linearly separable.

The mapping from two dimensions to higher dimensions has made the data linearly separable, and thus a linear method can now be applied to the transformed “kernelized” data. This is the essence of kernelizing data in the context of PCA, that of transforming the data via kernels so that otherwise nonlinear data becomes linearly separable in the feature space and an “ordinary” PCA can then be applied to the transformed “kernelized” data.

### 11.6.3 Kernelizing and Computational Complexity

There is another benefit to kernelizing. When we move from two dimensions to three, or, more generally, from lower dimensions to higher, such can seriously increase computation time. For smaller problems, this is usually not a problem, but for larger dimensionality tasks it becomes prohibitive. Hence, an immediate problem with high dimensionality is that of *computational complexity*. We would like to get around this problem and still capitalize on the benefits of working in higher-dimensional feature space. If we can compute in lower-dimensional space but actually work in higher-dimensional space, such would be ideal. This is precisely what the kernel trick allows us to do in terms of facilitating computation. That is, we can take advantage of higher-dimensional representations *without having to actually work in those higher configurations*. Instead, we can remain in lower dimensions and nonetheless conduct our analyses there. This is accomplished by computing *kernels*, which even though computed in lower dimensions, contain information about higher-dimensional representation.

The above result is due to *Mercer's theorem*, which the reader may recall was discussed earlier in the book in relation to support vector machines. In brief, *the computational complexity of kernel PCA does not one-to-one increase with the dimensionality of the feature space* (Schölkopf et al., 1998, p. 4), and so even if we are working in very high-dimensional space theoretically, the computational complexity can be “tamed” somewhat by computing in a lower-dimensional space. This is thanks to the fact that we can compute kernels to represent this higher-dimensional inner product (Hilbert) space, and the fact that kernels are relatively simple to compute. Again, kernelizing is a marvel!

What is more, with kernel PCA no nonlinear optimization is involved. We can simply solve an eigenvalue-eigenvector problem as we would in traditional PCA *but now on the kernelized data*. The benefit of this is that getting “stuck” in local optima, in this case, *minimums*, is not a problem as it might be using other nonlinear methods. That is, *global optima* can be attained in kernel PCA (Belkin & Niyogi, 2003, p. 1374). Clearly, kernel PCA has computational benefits to employing direct nonlinear methodologies.

*The kernel trick allows us to work and compute in lower dimensions but benefit from higher-dimensional representation. The advantage of this is to both account for potential nonlinearity in the data but also to simplify computation time since we do not have to actually access the higher-dimensional space or surfaces in order to account for it. This is more generally known as the “kernel trick” and is due to Mercer's theorem.*

### 11.6.4 Reconstruction Error in Kernel PCA

As was the case for classic PCA, the goal in kernel PCA will be to minimize the reconstruction error. It should make sense as to why this goal has not changed when moving from classical PCA to kernelized PCA; both methods are essentially doing the same thing, but simply on different transformations of the variables and data. At first glance, it may appear that enlarging the feature space would *increase* the dimensionality in which we need to compute. However, contrary to what we may expect, this is not what occurs, thanks again to the *kernel trick*. As we have emphasized, due to the kernel trick, we will not actually have to do any calculations in the higher-dimensional feature space. *Computation of kernels occurs in the original lower-dimensional space of the data*. As summarized by Hastie et al. (2009, p. 547):

Kernel PCA expands the scope of PCA, mimicking what we would obtain if we were to expand the features by non-linear transformations, and then apply PCA in this transformed feature space.

To add to all this, the higher-dimensional space could very well turn out to be *infinite dimensional*. This is why it's called a "trick," because we can simply compute kernels in lower dimensions and learn something of the higher dimensions without actually doing complex computations in higher dimensions. By analogy, it would be somewhat like being able to add two numbers, a very simple operation, to learn more about how the numbers relate in very complex ways (e.g., whether the distance between the numbers is a prime number). *The kernel trick allows for a simple computation to represent a much more complex reality or "space" behind those numbers.* Inexact analogy to be sure, but approximates the idea somewhat in spirit. The takeaway message is that the kernel trick is a *big deal* in statistics and machine learning.

### 11.6.5 The Matrices of Kernel PCA

Mathematically, kernel PCA relies on the fact that principal components can be obtained from a data matrix  $\mathbf{X}$  by computing inner products of that data matrix,

$$\mathbf{K} = \mathbf{X}\mathbf{X}^T \quad (11.9)$$

Eigen analysis is then performed on the double-centered version of  $\mathbf{K}$ , where  $\mathbf{K}$  is now equal to

$$\mathbf{K}_c = (\mathbf{I} - \mathbf{M})\mathbf{K}(\mathbf{I} - \mathbf{M}) \quad (11.10)$$

and where  $\mathbf{M} = \mathbf{1}\mathbf{1}^T / N$  which serves to center  $\mathbf{K}$  (Hastie et al., 2009, p. 547). That is, we are computing a kernel matrix  $\mathbf{K}$  that contains all pairwise dot products between data points. The double-centering here ensures the mean of every row and column is equal to 0. Whereas classical PCA uses the covariance matrix and scales regarding the number of input dimensions, kernel PCA scales with respect to the number of *data points* rather than dimensions in that the eigen analysis occurs on the  $\mathbf{K}$  matrix and not the covariance matrix as in PCA.

As is true when using kernels, a variety of kernels are available, such as the *radial kernel* and others. However, the choice of kernel is critical, and the expression "*what you find is a function of what you put in*" is very relevant here; one kernel over another can drastically generate different findings (for an example of this, see Hastie et al., 2009, p. 549). What does the eigen problem look like? Recall classic eigen decomposition:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

For kernel PCA,  $\mathbf{A}$  is now the kernel matrix  $\mathbf{K}$ , and by simple substitution (Bishop, 2006, p. 588) the eigenvalue problem becomes,

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \rightarrow \mathbf{K}\mathbf{x} = \lambda N\mathbf{x} \quad (11.11)$$

or as simply

$$\mathbf{K}\mathbf{x} = \lambda\mathbf{x}$$

if  $\lambda = \lambda N$  (Izenman, 2008, p. 611). Note that we are performing eigen analysis on the  $\mathbf{K}$  matrix and not the covariance or correlation matrix as in traditional PCA (Bishop, 2006). Once we have the eigenvalues and eigenvectors, we can now project the data onto each new dimension.

*Kernel PCA can be conceptualized as an “ordinary” PCA, but instead of weighted by input variables by values of the eigenvector, it is the kernel of the data points that is weighted by values of the eigenvector. Hence, KPCA simply replaces the variables (dimensions) with the kernel of observations, so that the kernel trick can “do its magic.”*

#### 11.6.6 Classical PCA as a Special Case of Kernel PCA

We have seen that kernel PCA allows for nonlinearity. Hence, it stands that with some constraints imposed, classical linear PCA should be able to be obtained and operationalized as a *special case* of kernel PCA. Indeed, it should be evident from our earlier discussion that classical PCA must simply be a special case of KPCA (Bishop, 2006, p. 589) in which the *kernel is the linear kernel*. This is powerful insight. *In general, many linear methods can be regarded as special cases of a more general nonlinear framework.* The set of nonlinear statistical “possibilities” is much wider than the smaller set of linear models, and hence linear methods are a subset of that wider set, not only in the case of kernel PCA in relation to PCA but for most statistical methods in general.

#### 11.6.7 Kernel Trick Is Not Simply about Cost

The kernel trick is useful since it saves on computational cost in accessing the higher-dimensional feature space. However, it should be emphasized that that is not the only reason for its utility. The primary benefit to using kernel methods is not only as a way around computational complexity. To reiterate, *the kernel trick allows us to work in higher dimensions without knowing the specific details of that dimensionality. That is its key benefit.* Hence, though often kernel methods in general are advertised or evaluated for their computational properties in “accessing” higher dimensionality, applying kernel methods in general is far from simply a way to ease computation. Even if the computations are laborious, which they can be in computing dot products of large matrices, kernel methods are useful since they allow access to those higher dimensions in the first place. *In general, kernel methods allow access to higher dimensionality while working in lower dimensions, often all for a more reasonable computational cost than if we were to access those higher dimensions directly.*

#### 11.6.8 Kernel PCA in Python

We will perform kernel PCA in Python using the `KernelPCA` function:

```
import numpy as np
from sklearn.decomposition import KernelPCA
X = np.array([[ -1, -1], [ -2, -1], [ -3, -2], [ 1, 1], [ 2, 1], [ 3, 2]])
X
Out[24]:
array([[ -1, -1],
```



```
[-2, -1],
[-3, -2],
[ 1,  1],
[ 2,  1],
[ 3,  2]])
```

We will request a kernel PCA for two components using a *radial basis function* (i.e., `kernel = "rbf"`):

```
kernel_pca = KernelPCA(n_components = 2, kernel = "rbf", gamma = 0.03)
```

We have requested two components (`n_components = 2`) as well as the radial basis kernel (`kernel = "rbf"`). We have also set `gamma` equal to 0.03, which is the kernel coefficient for the radial basis function (rbf).

```
final = kernel_pca.fit_transform(X)
print(final)
[[-0.3149893 -0.17944928]
 [-0.46965347 -0.0475298 ]
 [-0.62541667  0.22697909]
 [ 0.3149893  -0.17944928]
 [ 0.46965347 -0.0475298 ]
 [ 0.62541667  0.22697909]]
```

The above are the two components resulting from the kernel PCA. Of course, the result looks very similar to what we might have obtained had we performed a classic PCA (i.e., two columns of eigenvectors), but remember that because it is a kernelized PCA, eigenvectors were computed on transformed “kernelized” data, not the original (and typically centered) data we would ordinarily use for a traditional PCA.

### 11.6.9 Kernel PCA in R

We can use the *kernlab* package in R (Karatzoglou et al., 2024) to perform kernel PCA. The following example is adapted from the package’s documentation, where the first and second components are plotted:

```
> library(kernlab)
> data(iris)
> test <- sample(1:150, 70)
> test

[1] 46 44 31 62 130 80 136 47 60 48 94 21 78 23 67 112 42 58 95
[20] 115 120 36 22 149 139 68 141 57 84 14 61 109 3 97 133 106 70 76
[39] 103 107 64 100 30 66 146 85 34 96 29 17 74 15 41 119 135 27 105
[58] 28 54 77 24 138 118 6 20 9 38 132 147 129
```

The above has produced 70 values from data with a range of 1 through 150. We now specify the kernel PCA, requesting that two features be extracted. We set the kernel equal to the *radial basis “Gaussian” kernel* (`rbfdot`) and *eta*, which is the Hebbian learning rate, equal to 0.01 (the default is 0.005). Sigma (`sigma = 0.2`) is the inverse kernel width for the radial basis kernel function, which we can set as a flexible parameter. We set the maximum number of iterations equal to 65:

```
> kpc <- kha(~., data=iris[-test, -5], kernel="rbfdot",
+ kpar=list(sigma=0.2), features=2, eta=0.01, maxiter=65)
```

To print the components, we code:

```
> pcv(kpc)
```

The above provides the component vectors (we print only a few rows):

|      | [,1]         | [,2]          |
|------|--------------|---------------|
| [1,] | 0.0057810608 | -0.0024630263 |
| [2,] | 0.0055920541 | 0.0021378803  |
| [3,] | 0.0053570607 | 0.0037087163  |
| [4,] | 0.0057192053 | -0.0031808879 |
| [5,] | 0.0059032737 | -0.0032838050 |

We can request to plot the obtained components as follows. The specification `iris[, -5]` removes the species variable from the data while `col=as.integer(iris[, 5])` adds it as an integer-valued column:

```
> plot(predict(kpc, iris[, -5]), col=as.integer(iris[, 5]),
xlab = "1st Principal Component", ylab = "2nd Principal Component")
```

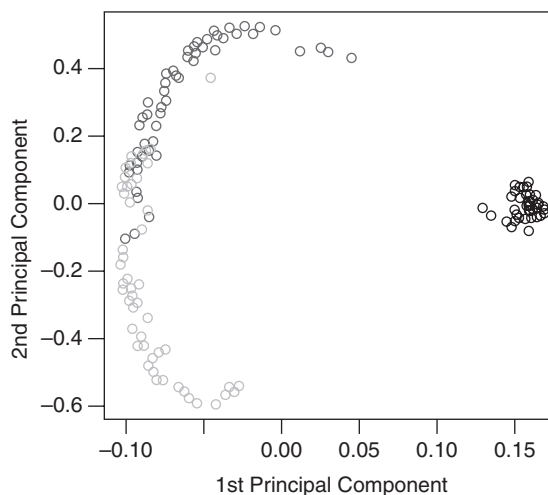


Figure 11.9 Kernel PCA plot of first and second components.

We can see from the plot (Figure 11.9) that a classic linear PCA would have likely never captured the complex configuration of the data. However, kernel PCA was able to quite nicely uncover the nonlinear and rather complex dimensionality.

## 11.7 Principal Curves

Recall that in classic PCA, we fit a hyperplane such that the sum of the squared perpendicular distances is minimized. Principal curves are more general than PCA. As we will see, PCA in fact can be considered a special case of the wider principal curve framework just as we considered it a specialized case of kernel PCA. Instead of providing a line of “best fit” as in PCA, principal curves seeks to find a curved *approximation*, a curved *manifold* of dimensionality two or greater. However, as an estimation algorithm principal curves does not result in dimension reduction (Meila & Zhang, 2023, p. 9). More simply, principal curves fits a curve to a swarm of data points, as illustrated in Figure 11.10.

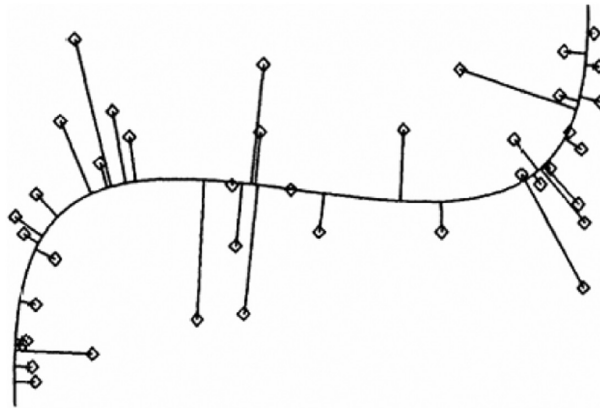


Figure 11.10 Principal curves fit to data. From Hastie, T., & Stuetzle, W. (1989). Principal Curves. *Journal of the American Statistical Association*. Copyright © 1989 American Statistical Association, reprinted by permission of Informa UK Limited, trading Taylor & Francis Group, <https://www.tandfonline.com> on behalf of American Statistical Association.

The fit provided by principal curves is *nonlinear* and *nonparametric*, meaning that their exact form is suggested or “learned” by the data (Hastie & Stuetzle, 1989). A precise definition of a principal curve was given by Hastie and Stuetzle (1989) who first proposed the method: “an infinitely differentiable finite length curve that passes through the middle of the data.”

A feature of the curve is that it is *self-consistent*, meaning that every point on the curve is the expected value of the data points projecting onto this point (Ozertem & Erdogmus, 2011). What this means is that if we arbitrarily selected a point on the curve and amassed all the data that project onto this point, the average of all those points would coincide with the point on the curve (Hastie & Stuetzle, 1989, p. 502); hence, “self-consistent.” We can immediately see that *if the curve is a straight line and is self-consistent, then the principal curve will be that of a principal component*. Algorithms used for principal curves essentially iterate until this convergence of self-consistency is achieved (Hastie & Stuetzle, 1989). Details of the algorithm are given in Hastie and Stuetzle (1989, p. 506).

Note that the definition of a principal curve is different from that of a principal component or even ordinary least-squares line in regression. Both of these latter methods fit a constant straight-line hyperplane to the data regardless of what the data are. There are no “bends” in the curve to follow the data. That is, in PCA and ordinary least-squares (OLS), *we are imposing a linear form*

on the data; the data are not telling us what form of polynomial we should fit. In contrast, principal curves provide a rather “smooth” fit to the data; the fitted curve is essentially “guided” by the data.

*In classic PCA or OLS regression, we impose a linear form on the data, regardless of whether the data actually follow a linear form or not. In contrast, in principal curves, the form of the curve is suggested or “learned” from the data, and is hence, nonparametric in nature.*

Essentially, principal curves find the “middle” of the data in some sense (the “self-consistency” sense) when fitting the curve. Not surprisingly, there are various notions of how “middle” is defined. *Global* or *top-down* algorithms begin with an initial line, which is usually the first classic principal component, then through iterations, “bend” this line until some convergence criteria are met. The curve that is fit is a result of successive iterations and is a *smooth local average of points*, where the definition of “local” is based on the distance in arc length of the projections of the points onto the curve found in the previous iteration (Hastie & Stuetzle, 1989). As summarized by Hastie and Stuetzle (pp. 502–503):

We formally define principal curves to be those smooth curves that are self-consistent for a distribution or data set. This means that if we pick any point on the curve, collect all the data that project onto this point, and average them, then this average coincides with the point on the curve ... the algorithm for finding principal curves is equally intuitive. Starting with any smooth curve (usually the largest principal component), it checks if this curve is self-consistent by projecting and averaging. If it is not, the procedure is repeated, using the new curve obtained by averaging as a starting guess. This is iterated until (hopefully) convergence.

Other more local, or *bottom-up* approaches estimate the curve essentially “step by step” moving through the data cloud (Delicado, 2001). These more local approaches are generally more flexible but have higher variance since small perturbations in the training data can drastically affect the location of the curve. *Local principal curves* is one technique that models segments of the curve within a certain *bandwidth*. The concept of principal curves can be extended to that of *principal surfaces* when working in higher dimensions.

### 11.7.1 *Principal Components as a Special Case of Principal Curves and Surfaces*

We can now see how principal components can be conceptualized as a special case of principal curves in which the “curve” in PCA is linear (Sandilya & Kulkarni, 2002). Principal curves are the more general procedure. That is, while principal curves uses the data to fit a curve, principal components in a sense still uses the data to determine the placement of the components, but *the curve is always a hyperplane*. Indeed, if we extend curves to surfaces, then PCA becomes a special case of principal curves which in turn is a special case of principal *surfaces*. This is also clear from the fact that the principal curve algorithm begins with fitting a principal component, until it iterates to successfully fit the corresponding curve based on the arrangement of the data (Hastie et al., 2009).

### 11.7.2 *Principal Curves in R*

The package *princurve* (Hastie et al., 1989) can be used to fit a principal curve in R. We adopt an example from the package and produce a plot of the resulting curve:

```
> library(princurve)
> x <- runif(100, -1, 1)

> head(x)
[1] -0.81697944 -0.08094903 0.19879631 0.83944382 0.96564815 -0.92439484
```

The above generates 100 cases of random data from a uniform distribution with limits  $-1$  to  $1$ . We will now transform the data to make it quadratic by including an  $x^2$  term:

```
> x <- cbind(x, x^2 + rnorm(100, sd = 0.1)
+ )
```

We now fit the principal curve on  $x$  and plot the result along with lines and whiskers characterizing the fit:

```
> fit <- principal_curve(x)
> plot(fit)
> lines(fit)
> points(fit)
> whiskers(x, fit$s)
```

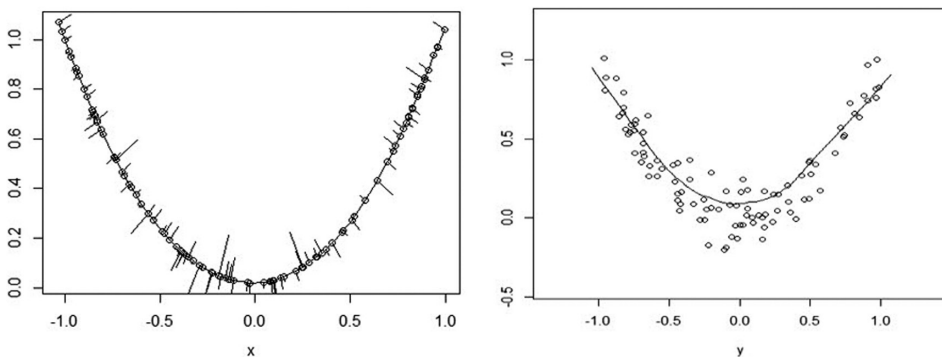


Figure 11.11 Principal curve fit to data with whiskers (left) and smoother (right).

The output generates a plot (Figure 11.11) with whiskers (left), where the whiskers correspond to deviations of data points from which the curve was fit. We can also fit a lowess smoother to the data (right), which reveals the actual data points:

```
> fit_smooth <- principal_curve(x, plot = TRUE, smoother = "lowess")
```

## 11.8 Principal Components Analysis as an Encoder

An *encoder* is a general term that has many variations and connotations depending on the exact context, but in its most general sense, is some function that translates or transforms data or information into another form, usually, and most generally, into a form with desirable properties. There are two parts to an encoder, the *encoding* element and the *decoding* element.

An *autoencoder* is a special type of encoder that encodes through some latent path such as a *hidden layer* in a neural network. If the autoencoder has more than a single hidden layer, it is generally referred to as a *deep autoencoder*. The key feature of a *classic autoencoder* is that there are *as many inputs as there are outputs* (Alpaydin, 2020). An autoencoder in which the new function is required to have a lesser dimension than the inputting function is sometimes referred to as an *undercomplete autoencoder* (Goodfellow et al., 2016). Autoencoders generally belong to the family of unsupervised techniques.

A classic example of an autoencoder at a conceptual level is that of encoding cell phone data into the cloud, then decoding that data back to the cell phone. The important feature here is that, for the classic autoencoder at least, the data has not really changed dimensionality; it has simply changed *forms*. Had we constrained the data in the cloud to assume a lesser dimensionality than the original cell phone data, this would have been an example of the aforementioned *undercomplete autoencoder*.

More formally, the encoder in an autoencoder can be given by

$$\mathbf{z} = \text{Encode}(\mathbf{x} \mid \mathbf{W})$$

where  $\mathbf{W}$  are the parameters of the encoder and  $\mathbf{x}$  is the vector being encoded. The decoder is thus given by

$$\mathbf{x} = \text{Decode}(\mathbf{z} \mid \mathbf{V})$$

where  $\mathbf{V}$  are now the parameters of the decoder and  $\mathbf{z}$  is the vector being decoded.

Autoencoders can be linear or nonlinear. For example, consider the two images in Figure 11.12, one a linear encoder and one a nonlinear encoder, the only difference being the form of function in the hidden nodes (i.e., linear vs. nonlinear).

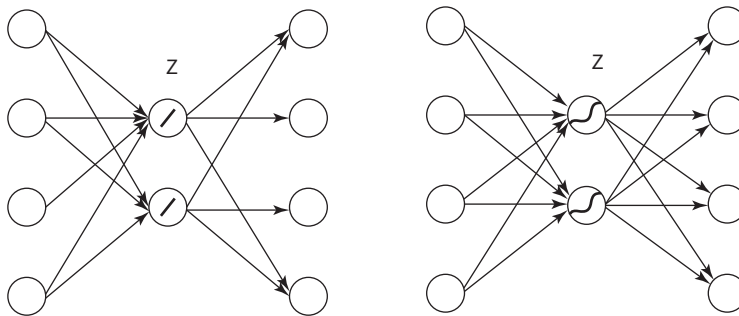


Figure 11.12 Two encoders, one linear (left), one nonlinear (right). In both cases, data from left to right is encoded into a new form.

## 11.9 Neural Networks and PCA as Autoencoders

The classic case of an encoder, as featured just above, is that of a neural network where the mappings are no longer linear but instead nonlinear. As we will see when we survey neural networks, there is nothing necessarily and intrinsically nonlinear about a neural network; technically the

activation function *can* be linear if chosen as such. However, neural networks typically feature nonlinear activation functions in an effort to capture more complex configurations in the data.

This is where now PCA comes in. An autoencoder with a single fully connected hidden layer combined with a linear activation function and a squared error cost function is closely allied to classical PCA. Essentially, a neural network that is trained to learn the identity function is what an autoencoder actually is since we are not interested in “changing” the data at all, we are simply interested in *reconstructing* it (Plaut, 2018), such as the example with the phone data being transmitted to the cloud. This sounds very much like PCA! Indeed, it can be shown that the subspace formed by the first few components of a PCA is near equal to the vectors formed from encoding an autoencoder neural network and the reconstruction error from each is approximately the same (Oken, 2017). Autoencoders reconstruct the data as efficiently as possible, and hence PCA can be regarded as a special case of the more general autoencoder idea. Recall however that PCA does not by itself perform dimension reduction. *We do*, by choosing among the components the first few that explain most of the variance. Essentially, an autoencoder multilayer perceptron with one hidden layer of units (a *linear* encoder) implements PCA.

*Sparse PCA* has also been proposed (Zou et al., 2004) which seeks to regularize coefficients in the components model by penalization, such as  $\ell_1$  regularization. As we will briefly discuss in the following chapter, a similar regularization penalty can be adopted in factor analysis as well giving birth to the *sparse factor analysis* model.

## 11.10 Multidimensional Scaling

Multidimensional scaling (Kruskal, 1964; Torgerson, 1952, 1958) is a nonparametric technique used for dimensionality reduction with the goal of preserving pairwise distances between observations. It is perhaps the simplest type of manifold learning (Murray, 2022). Recall that the problem of visualizing data in high dimensions is essentially to reduce the dimensionality and visualize resulting projections in those lower dimensions. In this sense, dimension reduction, as it pertains to visualization, is a way of answering the question:

*What would this high-dimensional data look like if the points existed in a much lower dimension?*

This is the goal of many dimension-reduction techniques, to learn how high-dimensional data “live” or “exist” in lower dimensions. As an example, suppose we consider the distances between cities in miles. We then put these distances into a matrix and perform multidimensional scaling on that matrix. What we will obtain is the *coordinate map* in two dimensions with the city distances preserved as much as possible.

As summarized by Chatfield and Collins (1980, p. 189):

Multidimensional scaling is the term used to describe any procedure which starts with the “distances” between a set of points (or individuals or objects) or information about these “distances,” and finds a configuration of the points, preferably in a small number of dimensions, usually 2 or 3.

Or, as remarked by Venables and Ripley (2002, p. 306),

The most obvious of the distance methods is multidimensional scaling (MSD) which seeks a configuration in  $R^d$  such that distances between the points best match (in a sense to be defined) those of the distance matrix.

Most generally, MDS takes a distance matrix of observed variables and seeks to find a lower-dimensional representation of those distances. It takes a distance matrix of say, ten dimensions,

and asks the question “*Can we represent this ten-dimensional matrix in terms of only a few underlying dimensions, and if so, what are the coordinates that best give rise to the observed distances/dissimilarities?*” In this sense, MDS seeks to accomplish a similar (though not *exact*) goal as PCA and other dimension-reduction techniques. In fact, classical multidimensional scaling with a Euclidean distance metric is equivalent to plotting the first  $k$  principal components, without rescaling to correlations (Chatfield & Collins, 1980; Murray, 2022; Venables & Ripley, 2002). A related interpretation of MDS is to uncover underlying dimensions such that the observed distances between points are maintained or “preserved” as closely as possible (Meila & Zhang, 2023; Rencher & Christensen, 2012). Most embedding algorithms do not preserve Euclidean distances, instead resulting in a distorted geometry to at least some extent (Meila & Zhang, 2023, p. 19).

Depending on whether we have a distance matrix already computed, classical MDS can begin with a matrix of raw data,  $\mathbf{X}$ , or the distance matrix  $\mathbf{D}$ . Beginning with  $\mathbf{X}$ , we compute a matrix  $\mathbf{B}$ , which is the matrix of sums and squares and cross-products,

$$\mathbf{B} = \mathbf{X}\mathbf{X}' \quad (11.12)$$

It stands that the matrix  $\mathbf{B}$  can be converted into a matrix of distances,  $\mathbf{D}$ , where we can obtain the squared Euclidean distances between rows of  $\mathbf{X}$  as a function of the elements of  $\mathbf{B}$ :

$$d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij} \quad (11.13)$$

These are squared Euclidean distances (Chatfield & Collins, 1980, p. 198), yet in the above formulation, it can be difficult to appreciate what is being computed and it is helpful to recall what a Euclidean distance actually is,

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots (x_p - y_p)^2} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})} \end{aligned}$$

where  $d(\mathbf{x}, \mathbf{y})$  is the distance between vectors  $\mathbf{x}$  and  $\mathbf{y}$ . It should be emphasized that the distance matrix  $\mathbf{D}$  does not *need* to contain Euclidean distances. Many other options are available, which we briefly survey when we discuss cluster analysis later in the book. What *is* important for  $\mathbf{D}$  is that the distances featured satisfy basic essential properties of distances  $d_{ij}$  where  $i$  and  $j$  correspond to points  $i$  and  $j$ . These properties include  $d_{ij} \geq 0$ ,  $d_{ii} = 0$ , and  $d_{ij} = d_{ji}$  (Chatfield & Collins, 1980). Beyond such essential properties, MDS does not really “care” where  $\mathbf{D}$  comes from. Though, of course, *the scientist should*. Distance measures (sometimes called *similarity coefficients*) can also be obtained for binary data (see Chatfield & Collins, 1980, p. 194 for details).

Given the correspondence between  $\mathbf{B}$  and  $\mathbf{D}$ , an eigenanalysis can then be computed on  $\mathbf{B}$ , which, assuming  $\mathbf{B}$  is of full rank, will yield as many eigenvalue-eigenvector pairs as there are rows or columns of  $\mathbf{B}$ . The resulting eigenvectors are new dimensions from which the data can be projected. As in PCA, we can then evaluate the size of respective eigenvalues relative to the sum of eigenvalues to determine the “importance” of the given dimension, where proportions greater than 0.8 are suggestive of an at least reasonable fit (Everitt, 2007, p. 95), however such “cut-off criteria” should not be taken rigidly and like PCA will depend on the nature and context of the problem. To emphasize, the matrix of raw data in  $\mathbf{X}$  was not actually required had we already had  $\mathbf{D}$ . That is, MDS only requires a distance matrix, it does not operate directly on the raw data



(Hastie et al., 2009). Of course, in most cases, raw data is all the investigator has, and hence such will usually be the starting point to computing an MDS, that of computing the matrix  $\mathbf{D}$  from  $\mathbf{X}$ . But theoretically, one can perform an MDS having only knowledge of the distance matrix, similar to how one can perform a PCA having only knowledge of the covariance or correlation matrix. How these matrices “arise,” MDS cares little about.

As mentioned, MDS comprises several related techniques and versions. The method surveyed above is generally known as *classical scaling*. A competing version of MDS, called *Kruskal-Shepard scaling*, seeks values or points  $p_1, p_2, \dots, p_n$  that minimize the following so-called *stress function*:

$$S = \sum_{i,j} (d_{ij} - \|p_i - p_j\|)^2 \quad (11.14)$$

where  $d_{ij}$  is the distance between two data points  $i, j$  in the original data and  $\|p_i - p_j\|$  is the corresponding “distance”  $i$  to  $j$  between reconstructed or “embedded” data points  $p$  based on the dimension reduction. It should be noted these latter distances may not represent actual *distances* and may not agree with classical properties of distances; in the case of nonmetric MDS (to be discussed shortly), they are only used to help determine the monotonicity of the original distances,  $d_{ij}$  (Rencher & Christensen, 2012). On the other hand, in a version of metric MDS in which  $d_{ij}$  represent more than an ordinal ranking, interpreting  $\|p_i - p_j\|$  as the distance between observed and reconstructed points is more acceptable.

An alternative *scaled* way of defining the stress function is,

$$S = \frac{\sum_{i,j} (d_{ij} - D_{ij})^2}{\sum_{i,j} d_{ij}^2} \quad (11.15)$$

where  $d_{ij}$  is once more the distance between two data points  $i, j$  in the original data and  $D_{ij} = \|p_i - p_j\|$  is once more the distance between the newly constructed, estimated, or “mapped” points (Chatfield & Collins, 1980). The lower the stress, the better the fit. The most ideal situation is for  $d_{ij} = D_{ij}$  under which case  $\sum_{i,j} (d_{ij} - D_{ij})^2 = 0$  and thus  $S = 0 / \sum_{i,j} d_{ij}^2 = 0$ . Stress increases as  $\sum_{i,j} (d_{ij} - D_{ij})^2$  increasingly deviates from zero but has limits  $[0, 1]$  due to the denominator,  $\sum_{i,j} d_{ij}^2$  which serves to *scale* (not *standardize*) the squared distances. Stress decreases as the number of dimensions increases (Chatfield & Collins, 1980, p. 208).

When the distance is Euclidean, PCA and classical MDS will coincide. This is because  $\mathbf{S}$ , the sample covariance matrix of PCA is proportional to  $\mathbf{B} = \mathbf{X}\mathbf{X}'$ . This can be seen by inputting  $\mathbf{B}$  into the eigenvalue-eigenvector characteristic equation and observing these two equalities,

$$\mathbf{B}\mathbf{a}_i = \lambda_i \mathbf{a}_i$$

where  $\mathbf{a}_i$  are eigenvectors and  $\lambda_i$  corresponding eigenvalues. We see that the eigenvalues will be the same and eigenvectors will also be subject to a linear transformation (Chatfield & Collins, 1980, p. 200). As noted by Jolliffe (2002), PCA operates on a matrix of similarities between *variables*, whereas principal coordinate analysis (i.e., classical MDS) operates on a matrix of similarities between *observations* (i.e., *individuals*). However, both can lead to the same results.

Consult Chatfield and Collins (1980, p. 201) for a fruitful discussion of how PCA and MDS relate to one another and why these authors recommend that PCA, not MDS, be performed if

distances are Euclidean. Jolliffe (2002, p. 90) seconds this recommendation, except notes that when the number of cases is less than the number of variables (i.e.,  $n < p$ ), even though both MDS and PCA will yield results, MDS will have computational advantages over PCA.

*Multidimensional scaling is equivalent to classical principal components analysis when Euclidean distance is used as a metric on the non-standardized correlation matrix, that is, the covariance matrix. When non-Euclidean distances are used, the equivalency no longer necessarily holds.*

The classical form of MDS is called *principal coordinate analysis*, which is usually a metric version of scaling (see below for a discussion of metric vs. nonmetric approaches).

### 11.10.1 Merits of MDS

When compared with other dimension-reduction techniques (e.g., classical PCA), metric multidimensional scaling carries with it many advantages. For one, it is not restricted to linear projections of the data and hence can handle nonlinear problems. Second, it can be applied to any metric space, not only Euclidean (Demaine et al., 2021). Distances can also be arrived at subjectively, for instance, subjects stating their preferences for a given product as an estimate (Rencher & Christensen, 2012). This choice will depend on the preferred metric, which we discuss shortly. The issue of what the proximity matrix contains is a substantive one; to reiterate, MDS itself, or any other statistical procedure does not care where the numbers came from, which is why *measurement*, not *analysis*, should be the primary concern of any investigation. Poorly measured attributes are not usually worth analyzing, either statistically or any other way.

A last point worth mentioning regarding the merits of MDS is that it emphasizes, and thus attempts to minimize, *squared differences*. While this may be ideal in some contexts, it may be less so in others. In some contexts, smaller distances may actually matter more than larger ones. *Sammon mapping* is one technique that puts greater emphasis on preserving smaller distances and hence attempts to capture more of the *local structure* of the manifold than does classical MDS. The stress function for salmon mapping can be given as follows,

$$S = \frac{1}{\sum_{i < j} d_{ij}} \sum_{i < j} \frac{(d_{ij} - D_{ij})^2}{d_{ij}} \quad (11.16)$$

A comparison of the above stress function to that of MDS reveals that distances  $(d_{ij} - D_{ij})^2$  are divided by  $d_{ij}$  to provide a sense of “relativity” to the squared distances. That is, if  $(d_{ij} - D_{ij})^2$  is found to be large, the stress function for Sammon mapping recognizes that this may in part be due to the squaring of the distance, and so relates it to  $d_{ij}$ , which is not squared. It is in this sense that Sammon mapping puts more emphasis on smaller distances than MDS. This also has the potential for Sammon mapping to be able to generate a nonlinear map (Alpaydin, 2020). The

constant  $\frac{1}{\sum_{i < j} d_{ij}}$  is simply to ease the computation of the gradient (van der Maaten & Hinton, 2008, p. 2597).

The fact that Sammon mapping focuses more on smaller distances still does not prevent one or two distances from dominating the cost function. As we will soon see, *t*-SNE attempts to

remedy this by not focusing so much on these raw magnitudes of separation in the data (van der Maaten & Hinton, 2008, p. 2597) and instead more closely modeling them as having equal local to global importance.

The package *ProjectionBasedClustering* (Thrun et al., 2024) can perform Sammon mapping:

```
> library(ProjectionBasedClustering)
> data('Hepta')
> Data=Hepta$Data
> Proj = SammonsMapping(Data)
Initial stress : 0.07779
stress after 10 iters: 0.05981, magic = 0.092
stress after 20 iters: 0.04725, magic = 0.213
stress after 30 iters: 0.04026, magic = 0.491
stress after 40 iters: 0.03924, magic = 0.500
stress after 50 iters: 0.03923, magic = 0.500
```

We can see the stress numbers decrease as a function of iterations. After 50 iterations, the stress is equal to 0.03923. The magic number (i.e., magic = 0.500) is the initial value of the step size constant in diagonal Newton's method and need not concern us. We can obtain a projection of the points as follows (Figure 11.13):

```
> PlotProjectedPoints(Proj$ProjectedPoints, Hepta$Cls)
```

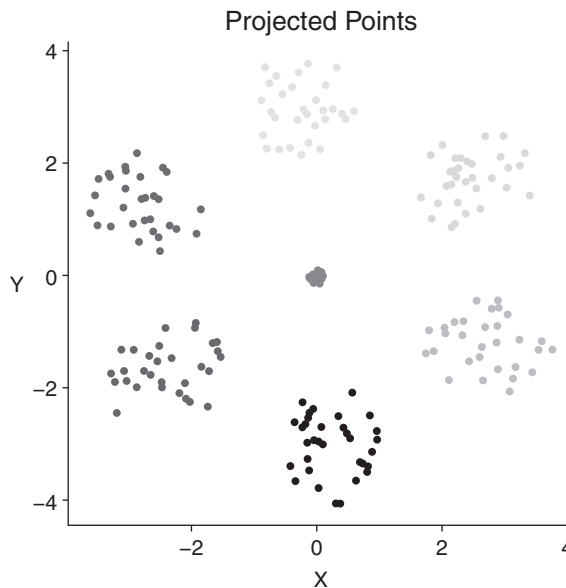


Figure 11.13 Projected points resulting from Sammon mapping.

The package *Rdimtools* (You & Shung, 2022) performs Sammon mapping via the function `do.sammon()`.

### 11.10.2 *Metric vs. Nonmetric (Ordinal) Scaling*

If we have actual distances available, that is, distances that are numerical or computational, this is generally referred to as *metric multidimensional scaling*, also generally known as *classical scaling* or *principal coordinate analysis* (Rencher & Christensen, 2012). If, on the other hand, the distances are arrived at via human judgment or other procedure such that all we can really know is how distances are *ranked* (i.e., we do not actually have the real numerical distances), this is generally referred to as *nonmetric* or *ordinal* multidimensional scaling (Chatfield & Collins, 1980, p. 190). In metric MDS, we attempt to dimension reduce based on actual numerical distances. In nonmetric MDS, we dimension reduce based on rankings or other more inexact distance derivation. Most commonly, for nonmetric, distances are often converted to ranks somehow, or in some applications, only the ranks exist in the first place without any access to more precise numerical measurements (Murray, 2022).

### 11.10.3 *Weakness of MDS: “Closeness” Can Be Arbitrary*

There is a limitation to multidimensional scaling, in that just because it considers two points to be “close” in some sense, this comes with the assumption that they are close *on a relatively simple manifold*. If the true manifold happens to be more complex, for instance a *curved manifold*, then the actual distance along the manifold might be much larger. Hence, one can perform MDS on data and obtain results, but those results are only as good as *the assumption of the nature of the underlying manifold*. MDS also may be limiting as a result of its objective function being nonconvex (Demaine et al., 2021). Recall what this implies. Optimization of a nonconvex function can result in there being multiple solutions in the form of minima instead of settling on a global minimum.

One technique that seeks to improve on MDS is that of *Isomap*. Isomap uses *geodesic* distances instead of Euclidean and thus is able to usually uncover manifolds that are more complex geometrically than what MDS is able to capture. In fact, this is true of most of the more advanced manifold learning algorithms. They all share a common goal in that *they seek to uncover a more sophisticated manifold than is possible with simpler techniques such as classical PCA or MDS*. This is precisely why these more advanced procedures, as we will soon see, are so attractive.

MDS also suffers from not having an explicit projection function (Yin, 2008). That is, if one adds even a single data point to the dataset, one must recompute the entire MDS algorithm.

*Multidimensional Scaling (MDS) typically assumes distances on the manifold are Euclidean, whereas more complex methods of dimension-reduction such as Isomap feature the use of non-Euclidean distances such as geodesic distances, which are not necessarily Euclidean. Allowing for greater complexity in how distances are defined is what makes more advanced methods of dimension reduction so attractive and popular.*

### 11.10.4 *Standardization of Distances*

Distances in MDS are often standardized so that each variable has a mean of zero and standard deviation of one. However, even with standardization, such does not by itself take into account

the correlation between objects (Chatfield & Collins, 1980). To take correlations into account, one could instead compute

$$D^2 = (\mathbf{x}_i - \mathbf{x}_j)' \mathbf{S}^{-1} (\mathbf{x}_i - \mathbf{x}_j) \quad (11.17)$$

which is a standardized version of Euclidean distance, typically called squared *Mahalanobis distances* (1936). Other measures of dissimilarity also exist (see Chatfield & Collins, 1980, p. 193). What is more, there is no “best” measure of similarity or dissimilarity. Instead, a researcher should choose one which seems most appropriate for his (or her) particular problem. There is also nothing at all wrong in trying out different distance measures in an exploratory sense to see how they compare in the final solution. Especially with the speed and ease of computing power, this should be easy to do. As usual when it comes to statistics, what you find in your data is often a function of what tools you use to do the searching. MDS is no different in this regard.

### 11.10.5 MDS in Python

We demonstrate MDS in Python on the `make_blobs` data set, where we extract two dimensions:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.manifold import MDS
```

We now create some data, specifying 100 samples on 3 features and print a few cases below:

```
X, _ = make_blobs(n_samples=100, n_features=3, centers=2, random_state=42)
X
Out [81]:
array([[ 3.8389442 , -6.40579427, -8.07141309],
       [ 1.25786597, -6.20002944, -7.61047622],
       [ 1.91364433, -10.12089453, -7.90449723],
       [ 2.2804692 , -6.06676507, -6.25048075],
       [-2.58120774, 10.01781903,  5.00151486],
       [ 1.98340274, -7.86113584, -6.41800612],
```

We confirm the dimensionality of the data:

```
print('Original Dimension of X : ', X.shape)
Original Dimension of X : (100, 3)
```

Now, we carry out the MDS:

```
mds = MDS(n_components=2, random_state=42)
X_2d = mds.fit_transform(X)
```

The above gives us the MDS scores, of which we only print the first few:

```
X_2d
Out[5]:
array([[ 0.29190348, 10.95109798],
       [ 0.85821636,  9.65165767],
       [ 3.45005476, 12.75571072],
       [ 1.66289895,  8.82820829],
       [-2.95232682, -10.82871729],
       [ 2.96070746, 10.10809041],
       [ 2.5896056 ,  9.54280101],
       [-0.68113354, -11.78283388],

plt.scatter(X_2d[:, 0], X_2d[:, 1])
plt.title("MDS Visualization")
plt.xlabel("MDS Dimension 1")
plt.ylabel("MDS Dimension 2")
plt.show()

Dimension of X after MDS : (100, 2)
```

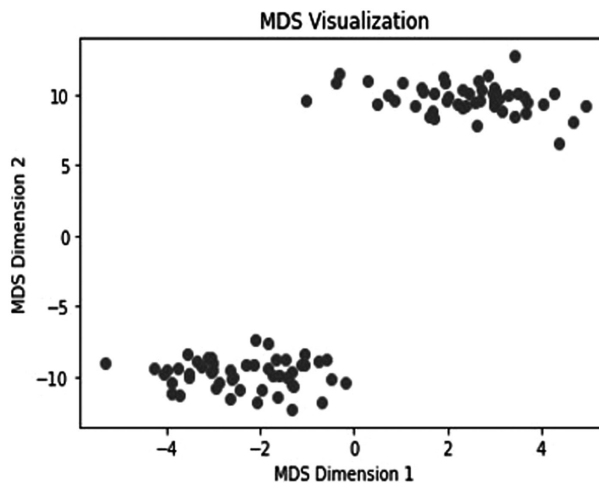


Figure 11.14 Multidimensional scaling plot on two dimensions.

The plot in Figure 11.14 reveals the dimensionality of the multidimensional scaling coordinates. We can see that the separation of the two components is quite good. We encourage the reader to adjust the number of components requested (`n_components =`) and observe how this changes the configuration.

### 11.10.6 MDS in R

We will use the package *smacof* (de Leeuw & Mair, 2009) to demonstrate an example adapted from the package on the *indvalues* data, which consist of responses on a survey measured on psychological variables such as power, achievement, hedonism, and others:

```
> library(smacof)
> require(plotrix)
> head(indvalues)
```

|   | Power    | achievement | hedonism | stimulation | self-direction | universalism |
|---|----------|-------------|----------|-------------|----------------|--------------|
| 1 | 4.719167 | 2.385833    | 2.0525   | 2.385833    | 3.185833       | 4.885833     |
| 2 | 5.283333 | 4.283333    | 3.616667 | 2.95        | 3.483333       | 3.158333     |
| 3 | 5.839167 | 3.6725      | 3.1725   | 3.839167    | 1.9725         | 2.6725       |
| 4 | 5.555833 | 3.389167    | 2.555833 | 3.2225      | 2.689167       | 3.389167     |
| 5 | 4.209167 | 3.459167    | 3.209167 | 3.875833    | 2.209167       | 3.959167     |
| 6 | 4.564167 | 2.8975      | 3.8975   | 2.564167    | 3.2975         | 2.6475       |

|   | benevolence | tradition | conformity | security |
|---|-------------|-----------|------------|----------|
| 1 | 3.385833    | 5.585833  | 3.635833   | 3.185833 |
| 2 | 2.883333    | 3.283333  | 2.783333   | 3.683333 |
| 3 | 1.9725      | 4.5725    | 3.9225     | 3.7725   |
| 4 | 3.089167    | 3.889167  | 4.139167   | 3.489167 |
| 5 | 2.809167    | 5.209167  | 3.459167   | 3.009167 |
| 6 | 3.2975      | 5.8975    | 2.6475     | 3.6975   |

We now generate a distance matrix from the individual data points. One option is to produce the correlation matrix (`cor(indvalues)`). The problem, however, is that the matrix will contain *negative* correlations, and MDS requires distances to be *positive* numbers. Therefore, following the package, we will subtract the correlation matrix from 1 (de Leeuw & Mair, 2009):

```
> valsD <- 1 - cor(indvalues)
> valsD
```

|                | power        | achievement | hedonism  | stimulation | self-direction |
|----------------|--------------|-------------|-----------|-------------|----------------|
| power          | 0.0000000    | 0.9737573   | 1.0832060 | 1.1455702   | 1.3090123      |
| achievement    | 0.9737573    | 0.0000000   | 1.0293717 | 1.0513279   | 0.9546741      |
| hedonism       | 1.0832060    | 1.0293717   | 0.0000000 | 0.9236783   | 0.8986514      |
| stimulation    | 1.1455702    | 1.0513279   | 0.9236783 | 0.0000000   | 0.8317990      |
| self-direction | 1.3090123    | 0.9546741   | 0.8986514 | 0.8317990   | 0.0000000      |
| universalism   | 1.5031586    | 1.2212833   | 1.1386171 | 0.9940873   | 0.6840954      |
| benevolence    | 1.5029965    | 1.0231187   | 1.1674710 | 1.2687173   | 0.8812556      |
| tradition      | 1.0281644    | 1.2807115   | 1.3759928 | 1.2768669   | 1.4140597      |
| conformity     | 1.0697333    | 1.0910985   | 1.3946426 | 1.3810823   | 1.3787115      |
| security       | 0.8646163    | 1.1252766   | 1.1084871 | 1.3232004   | 1.3915963      |
|                | universalism | benevolence | tradition | conformity  | security       |
| power          | 1.5031586    | 1.5029965   | 1.0281644 | 1.0697333   | 0.8646163      |
| achievement    | 1.2212833    | 1.0231187   | 1.2807115 | 1.0910985   | 1.1252766      |
| hedonism       | 1.1386171    | 1.1674710   | 1.3759928 | 1.3946426   | 1.1084871      |
| stimulation    | 0.9940873    | 1.2687173   | 1.2768669 | 1.3810823   | 1.3232004      |

|                |           |           |           |           |           |
|----------------|-----------|-----------|-----------|-----------|-----------|
| self-direction | 0.6840954 | 0.8812556 | 1.4140597 | 1.3787115 | 1.3915963 |
| universalism   | 0.0000000 | 0.7032021 | 1.0602872 | 1.2287480 | 1.2140122 |
| benevolence    | 0.7032021 | 0.0000000 | 1.0224550 | 0.8261320 | 1.1719149 |
| tradition      | 1.0602872 | 1.0224550 | 0.0000000 | 0.7394517 | 0.9831678 |
| conformity     | 1.2287480 | 0.8261320 | 0.7394517 | 0.0000000 | 0.7185342 |
| security       | 1.2140122 | 1.1719149 | 0.9831678 | 0.7185342 | 0.0000000 |

We note now that all distances are positive numbers and can proceed to fit MDS:

```
> fit <- mds(valsD)
> fit

Call:
mds(delta = valsD)

Model: Symmetric SMACOF

Number of objects: 10
Stress-1 value: 0.195
Number of iterations: 17

> plot(fit)
```

The above plotting function is enough to produce the configuration of psychological variables along both dimensions in Figure 11.15 (Can you possibly name the dimensions on which the configuration was constructed?):

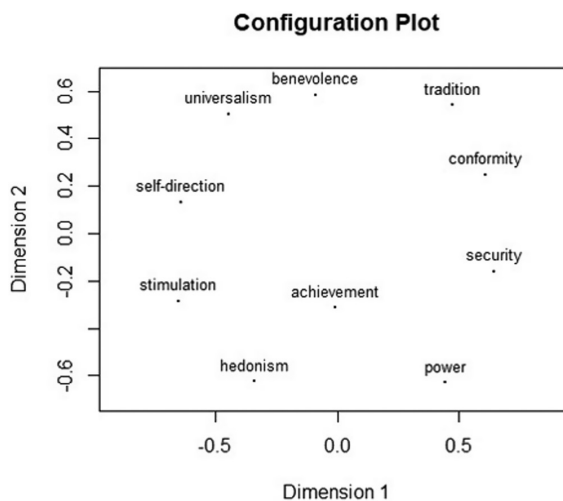


Figure 11.15 MDS configuration plot revealing variables on dimensions 1 and 2.



```
> circle <- fitCircle(fit$conf[, 1], fit$conf[, 2])
> draw.circle(circle$cx, circle$cy, radius = circle$radius, border =
"gray")
```

A further option is to use the `fitCircle()` function, which fits a circle into a two-dimensional point configuration (Figure 11.16). Using this option will not change the plot at all, it will only draw a circle on the existing plot (left):

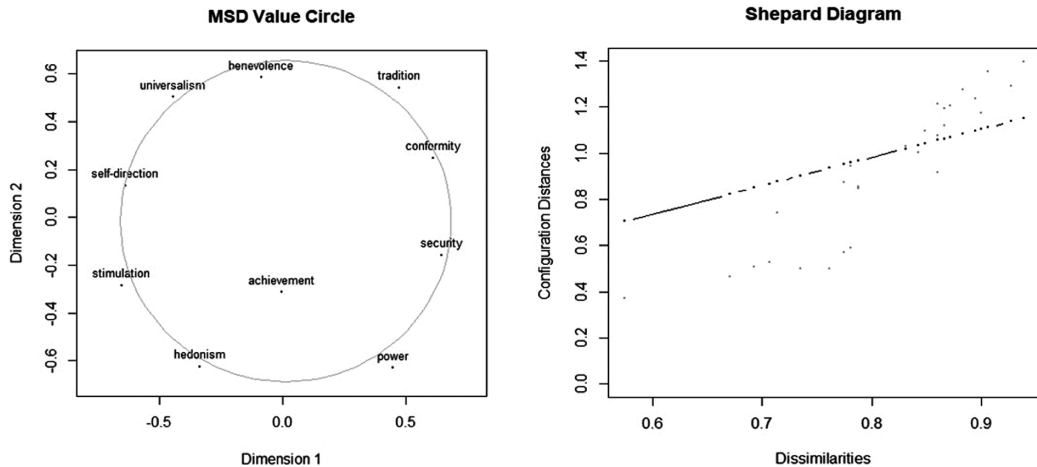


Figure 11.16 MDS plot with circle imposed (left). Shepard diagram relating the distances obtained by the MDS to the distances observed in the empirical data (right).

To the right of the dimension plot in Figure 11.16 is a *Shepard diagram*, obtained by

```
> plot(fit, plot.type = "Shepard")
```

The Shepard diagram relates the distances obtained by the MDS dimension reduction (on the *y*-axis, the *configured distances*) to the distances observed in the empirical data. The extent to which the slope is positive with minimal scatter is the extent to which configured or “reconstructed” distances relate well to the original empirical distances, which is the ideal. Many additional plots are available in the package. The *ProjectionBasedClustering* package can also be used to fit MDS.

The MASS package (Venables & Ripley, 2002) can also perform multidimensional scaling on categorical data. The `daisy()` function in the *cluster* package can produce dissimilarity matrices for both continuous and categorical data (Maechler et al., 2023).

## 11.11 Self-Organizing Maps

Multidimensional scaling can be slow because it must incorporate all distances between variables into its final computation (Venables & Ripley, 2002). An alternative to MDS, for large-scale problems especially, is the *self-organizing map* (Kohonen, 1990, 1995), which instead of using the entire matrix of distances, focuses on a rectangular or hexagonal two-dimensional lattice of

representative points. SOM can also be considered from the vantage point of cluster analysis, it being a special “constrained” case of  $k$ -means clustering where data are “encouraged” to lie in a one or two-manifold feature space (Hastie et al., 2009). While most of the work on SOMs occurred in the 1990s and beyond, the initial principle for SOM first appeared in the early 1980s (Kohonen, 1990).

There are several versions of algorithms for SOM. A simple version of the SOM algorithm operates similar to  $k$ -means clustering, in that given an observation from the data,  $x_i$ , the closest data point  $m_j$  is selected. Then, a neighbor of  $m_j$ , called the neighbor  $m_k$ , is moved toward  $x_i$  via the following update rule (Hastie et al., 2009, p. 528; Kohonen, 1990, p. 1467),

$$m_k \leftarrow m_k + \alpha(x_i - m_k) \quad (11.18)$$

where  $\alpha$  is the learning rate and where  $0 < \alpha < 1$ . The updating is run for each input vector and the process should be run at least 500 times the number of nodes (Izenman, 2008). The updating function can be more complex than a simple weight (Kohonen, 1990). For example, instead of simply a weight  $\alpha$ , one can specify a *function* of the weight,

$$m_k \leftarrow m_k + \alpha h_k(x_i - m_k) \quad (11.19)$$

where  $h_k$  now can be a function such as a kernel (e.g., a Gaussian kernel).

The algorithm is detailed in Yin (2008, p. 725). Self-organizing maps can also be generally considered in the family of neural network algorithms (Oliveira & Levkowitz, 2003) and can be integrated with other visualization techniques. The very idea of naming the method *self-organizing* arose from biology and sets of neurons essentially competing with one another, which is why often self-organizing maps are discussed under the machine learning topic of *competitive learning*. We quote Kohonen (1990, p. 1464):

It [self-organizing map] is a sheet-like artificial neural network, the cells of which become specifically tuned to various input signal patterns or classes of patterns through an unsupervised learning process.

Hence, in mimicking biological neuronal processes (or at minimum, *hypotheses* about such processes), self-organizing maps start with cells that “organize themselves” via updating themselves during training, a version of Hebbian learning (Yin, 2008). Such training concludes with the approximation of a smooth manifold (Bishop, 2006). Since there is no objective function to optimize, initial parameter estimates are crucial, perhaps “too important” since they could, in the end, essentially determine whether structure exists in the ensuing map (Bishop, 2006). The SOM algorithm can also be applied in a variety of contexts and customized to such applications. For example, various similarity measures can be adopted that suit the task at hand. That is, it need not be a distance function of the kind  $x_i - m_k$  discussed earlier, but instead can be an inner product (Yin, 2008, p. 726). SOMs have also been extended to incorporate mixture distributions. Work has also been done on kernelized versions of SOMs.

Self-organizing maps can be fit in Python using sklearn’s SOM and in R using the package *aweSOM* (Boelaert et al., 2022).

### 11.12 Manifold Learning

Principal components analysis assumes the data lay in a linear subspace. Sometimes however, perhaps actually most often, the data are best represented by a *nonlinear* subspace, such as in Figure 11.17, which is the classic “S-curve” shaped manifold.

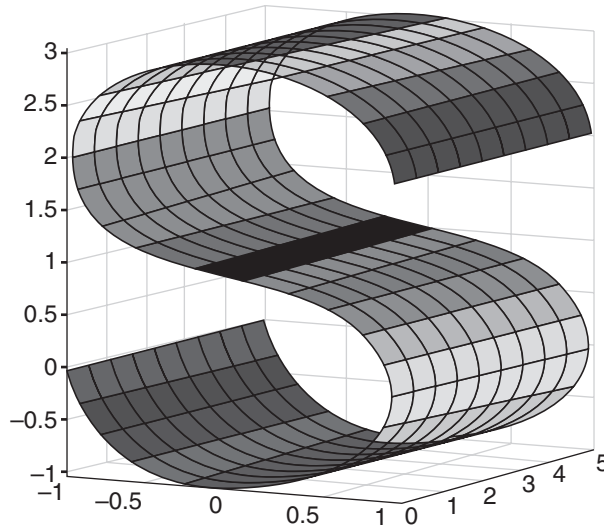


Figure 11.17 Classic S-shaped manifold.

In Figure 11.17, what is generally referred to as a *manifold*, sometimes called a *topological manifold*, since it satisfies certain topological conditions, such as being able to define continuous functions on that manifold. A manifold, first and foremost, is a topological space. A topological space is, roughly, a geometrical space where, for our purposes, measuring distances between points may prove challenging. A manifold is a topological space that is further *locally Euclidean* (Murray, 2022), which essentially generalizes the concept of a generalized surface in space. “Euclidean” here implies a differentiable mapping (Hubbard & Hubbard, 2002). By “locally Euclidean” we mean simply that at each point the space appears or resembles to be Euclidean, or, more crudely, *flat*. The space or neighborhood around each point should also consist of an *open set*, where by open set we may consider to be a higher-space generalization of an open interval on the one-dimensional real line in mathematics (Meila & Zhang, 2023).

A classic and practical example of such a manifold is the earth, a curved two-dimensional surface, but everywhere on the earth it appears locally flat (Murray, 2022). The local two-dimensional space is sometimes referred to as the *tangent space*. Hence, when you walk down the street on what appears to be a relatively flat surface on the earth, you are exploring the *tangent space* of a locally sectioned part of the earth. The earth is round, but the tangent you are on appears flat. Indeed, a two-dimensional map of a country is not a perfect representation since the actual surface is not actually flat but instead spherical.  $\mathfrak{R}^n$  is a manifold since everywhere you look you can be in  $\mathfrak{R}^n$  space. Any finite-dimensional vector space can be regarded as a manifold as well.

A one-dimensional manifold is a smooth curve, for example that of a wire or garden hose (Hubbard & Hubbard, 2002). An ordinary line is a special case of a curve and thus a one-dimensional manifold as well. A *smooth* manifold is generally one that is many times differentiable or even infinitely differentiable in some cases. The absolute value function  $f(x) = |x|$  is an example

of a simple manifold (curve) that is not smooth, as it is not differentiable at zero. The existence of a manifold is also independent of coordinates and can be easily translated.

Manifold learning and topology are largely about how to calculate on *local areas* of these more complex generalized surfaces. Differentiable manifolds are preferred as they allow one to perform calculus operations on the manifold. Topology essentially generalizes concepts of metric spaces (which are used to specify a distance function on that space) to the concept of *neighboring points* without having to specify a distance function explicitly.

The image in Figure 11.18 gives a clear picture of what manifold learning is about, that of taking a high-dimensional space and reducing it down to say, a two-dimensional scatterplot.

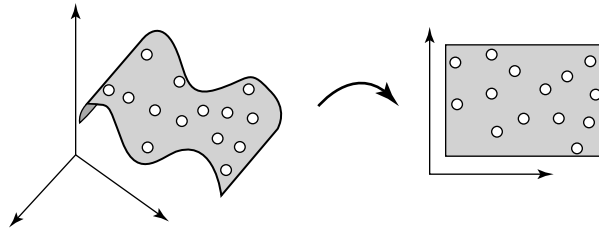


*Figure 11.18* The idea of manifold learning is that a high-dimensional space is reduced down to a lower-dimensional space, in this case a two-dimensional scatterplot. Used with permission of Springer Nature BV, from Pattern Recognition and Machine Learning by Christopher M. Bishop, © 2011; permission conveyed through the Copyright Clearance Center, Inc.

The hope of manifold learning is that the right-most plot contains pretty much the same *information*, in some sense, as the left plot, or that only a minimal amount of information is “lost” in the transformation (i.e., the dimension reduction). Another way of saying this is that in the dimension reduction, the *essential elements of the geometry are preserved*. Of course, we can immediately deduce that classical principal components analysis, for example, is a special case of manifold learning, since a high-dimensional space is reduced to a lower-dimensional topological space via linear transformation. The previously discussed multidimensional scaling, Sammon mapping, and self-organizing maps also can be included under the more general theme of manifold learning.

### 11.12.1 *Manifold Hypothesis*

The *manifold hypothesis* holds that even though data may *appear* in higher dimensions, it is possible to place it in a lower-dimensional manifold (Murray, 2022). That is, the *learning* of a manifold is an attempt to take high-dimensional multivariate data and reduce it down to a lower dimensionality. One can think of a manifold as being an *embedding* from a higher- to lower-dimensional  $\mathbf{R}^n$  space. Most embedding algorithms have the priority of identifying the set of eigenvectors associated with the most important (largest) eigenvalues. These methods are often referred to as *spectral embedding methods* (Izenman, 2008), however some reserve the classification of an embedding as distinct from other methods that necessarily preserve geometric and topological data properties.

Manifold learning is generally equated with *nonlinear* dimensionality reduction since in most cases the actual true manifold underlying the data will be nonlinear and the linear hypothesis is merely a special case of the more general nonlinear framework. It is not an exaggeration to say that most linear manifold methods seek out *global structure* whereas most nonlinear manifold methods seek out more *local nonlinear structure*. This parallels the idea of a line capturing global behavior of a scatterplot whereas a curve (or spline, for example) would capture potential nonlinear and more local behavior.

Lee and Verleysen (2005, p. 30) describe the goal of nonlinear dimension reduction:

The standard task in nonlinear dimensionality reduction consists of taking a set of data points sampled from a smooth, low-dimensional submanifold of a high-dimensional Euclidean space and to re-embed those points in a low-dimensional Euclidean space while preserving the local structure of the submanifold. The purpose is to find useful new coordinates for the data.

Of course, very little of the data the scientist will encounter will lie exactly on a linear or nonlinear manifold and hence *approximation of the manifold* will be a key concept in all of this.

### 11.12.2 Example of a Simple Manifold

It should be emphasized that while manifold learning generally refers to higher-dimensional spaces, manifolds themselves can be extremely simple in lower dimensions. For example, the following simple function, simple as it is, is nonetheless a manifold:

$$f(x) = (x/10)^3$$

The above will generate a one-dimensional plot (curve in this case) that is nonlinear (try plotting it). What dimension-reduction techniques will do is find a lower-dimensional representation. For example, we saw that PCA found the major axes underlying the variables in a very *global* fashion (i.e., based on the entire data). As we will see, *Isomap* will accomplish something similar, but will also preserve linearity *locally*, not only *globally* as is the case with PCA. That is, Isomap finds a nonlinear manifold but not directly through a global function as in PCA, but instead through connecting smaller local linear functions. *Locally linear embedding* does something similar to Isomap but goes about the task by finding weights that perform local interpolations between data points that closely approximate the data. We survey these approaches shortly.

### 11.12.3 Nonparametric Manifolds

When we perform dimension reduction of any kind, we can be said to be trying to find the intrinsic “true” dimensionality that underlays the data. *Nonparametric manifolds* are generally those that learn a low-dimensional embedding often at each point in the training data, but in general do not learn a mapping function that can be applied universally to out-of-sample data. That is, they typically do not impose an overall *functional form*, but instead focus heavily on *what the current data have to say*. Recall this has been one of our definitions of nonparametric, such that they utilize the current data maximally and do not, in general, impose implicit parametric forms on the data that come laden with prior assumptions.

*A nonparametric manifold learns a low-dimensional embedding often at each point in the training data but in general does not learn a concise and universal mapping function that can be applied to out-of-sample data. That is, nonparametric manifolds are, generally, “built” around the training data.*

### 11.12.4 Geodesic Distances

The shortest distance between any two points on a manifold is called the *geodesic distance*. Key to a geodesic distance is that it is the *shortest* distance or “path” connecting two points on a surface.

If that surface happens to be on a plane, then the geodesics of the plane are straight lines (Tanton, 2005). If the surface happens to be more complex, then the geodesic will likewise be more complex and may not even be unique (e.g., for a sphere the distance between poles is not unique). The use of geodesics assumes there are no gaps in the manifold on which they are measured, and also assumes the data do not lay outside of the data manifold (Ozertem & Erdogmus, 2011). Geodesics often have no closed-form formula such as Euclidean distances and hence are usually difficult to calculate, requiring numerical methods. In R, *geodist* (Padgham & Sumner, 2024) can compute such distances.

### 11.13 Locally Linear Embedding

Local linear embedding (LLE) recovers global structure by using linear “patches” in the wider structure. Each “patch” of the manifold is approximately linear. That is, LLE does what many other local methods do, whether in dimension-reduction or even nonparametric regression, and that is to “zoom in” on narrow *slices* or *patches* where things can be regarded as linear because the slice or patch is so small (think of a magnifying glass “zooming in”). As the slices become larger, the local linearity concept begins to vanish and so the slices cannot be too large. LLE uses this local linearity idea to essentially find a weight-based representation of each point of the manifold using its neighbors so that there are locally linear “neighborhoods” in the manifold. As the name suggests, the “embedding” is locally linear, even if the global manifold is not (i.e., the global manifold is typically highly *nonlinear*). The hope, however, is to capture the nonlinearity through smaller linear *areas* (i.e., *patches*) of the manifold.

Recall at the outset that one overarching theme we gave for statistical learning methods; they either impose a *global* structure onto data, or, they focus in more *locally* to observe segments of the data. Hastie et al. (2009) brilliantly introduce the whole edifice of statistical learning emphasizing this one key distinction. Nearest-neighbor methods essentially use local information in accounting for the data. These are generally nonparametric techniques because they heavily use the data that surrounds the given observation and rely exclusively on training data. This is in contrast to more global parametric methods that impose parametric structure on the data, such as linear or polynomial regression models. These methods are less concerned about the training data and are more interested in generalizing to future observations. The local parameterization of LLE is such that one can “embed” a new set of points in a lower-dimensional manifold. Such preserves, to the extent that is possible, the same relative positioning information (Ihler, 2003) via a nearest-neighbor approach.

More specifically, locally linear embedding hypothesizes that the best linear approximation around a given data point  $x_i$  can be found by predicting the data point  $x_i$  as a linear combination of its  $k$ -nearest neighbors using reconstruction weights. The weight vector is found by minimizing the following cost function,

$$\mathbf{W} = \min_{\mathbf{w}} \sum_{i=1}^N \|x_i - \sum_{j=1}^N w_{ij} x_j\|^2 \quad (11.20)$$

where the sum of weights  $w_{ij}$  equals 1, and where  $w_{ij}$  denotes the “contribution” of  $x_j$  to the eventual reconstruction (to be discussed shortly). We must choose a suitable neighborhood of points from  $x_i$  when computing  $\sum_{j=1}^N w_{ij} x_j$ . That is,  $\sum_{j=1}^N w_{ij} x_j$  essentially constitutes the *neighborhood* of the focal point,  $x_i$ . We wish to weight the neighbors via  $w_{ij}$  such that  $\mathbf{W}$  is minimized.

If the neighborhood is too large we lose the *local linearity* of local embedding. The neighborhood can be chosen by criteria such as  $k$ -nearest neighbors or by imposing a sphere around  $x_i$  with a minimal radius  $\varepsilon$  (Ihler, 2003). The possibility also exists to use different values for  $\varepsilon$  for different segments of the input space to help assure that  $\varepsilon$  is never too large to significantly weaken the embedding (Alpaydin, 2020; Saul & Roweis, 2003). Standard algorithms for  $k$ -nearest neighbors can be used as a more efficient method for determining neighbors. A randomized approximate nearest-neighbor algorithm can also be used (Jones et al., 2011).

A weight is also set to 0 if the neighboring point is the actual point we are building a neighborhood around, hence, this necessitates the idea of considering only points that are truly “neighbors” instead of the actual focal point (Roweis & Saul, 2000). The corresponding weights in  $\mathbf{W}$  are then used in reconstructing values of  $Y$ , which correspond to the lower-dimensional representation.

How many dimensions should  $Y$  consist of? The lower dimension projected points are obtained by minimizing a similar function as we did for the original higher-dimensional scores  $x_i$ , only that now, the minimization is relative to  $y_i$  instead of  $x_i$ ,

$$\mathbf{z} = \sum_i \|y_i - \sum_j \mathbf{W}_{ij} y_j\|^2 \quad (11.21)$$

such that  $\sum_{i=1}^n y_i y_i^T = \mathbf{I}$  and also that  $\sum_{i=1}^n y_i = 0$ . The first condition guarantees that points will be uncorrelated, whereas the second condition centers the output on the origin. Together, both conditions impose a unit covariance matrix for  $\mathbf{z}$  which now contains the new coordinates as a function of the weights in  $\mathbf{W}$ . That is,  $E(\mathbf{z}) = 0$  and  $Cov(\mathbf{z}) = \mathbf{I}$  (Alpaydin, 2020). The function for  $\mathbf{z}$  is then minimized via an eigen analysis on

$$\mathbf{E} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W}) \quad (11.22)$$

thus obtaining  $d$  eigenvalue-eigenvector pairs that correspond to the new embedded dimensionality of the original data (Belkin & Niyogi, 2003), keeping the eigenvectors that correspond to the lowest (nonzero) eigenvalues. Such will yield a set of orthogonal coordinates (Roweis & Saul, 2000) corresponding to the  $d$ -dimensions chosen for the eigen analysis, where  $d$  is a free parameter set by the user. In essence then, *LLE first establishes points as a function of some arbitrarily chosen definition of nearest neighbors, then reconstructs new points in lower dimensions, given the constraint that such “new” points are reconstructed subject to the aforementioned concept and definition of neighborhoods. In this sense, the reconstruction implements the concept of “linear patches” in reconstructing points in lower dimensions.*

An advantage of LLE is that it is unlikely to get “stuck” in local minima as the optimization is convex. A disadvantage is that it does not learn a new mapping such that we can predict new values for  $\mathbf{z}$  for new inputs (Alpaydin, 2020). Note that LLE also makes no attempt to measure geodesic distances. This will prove to be a significant difference when we compare it to procedures that do, such as Isomap.

### 11.13.1 LLE in Python

We consider an example of LLE in Python. We will see how the original data can be transformed into an embedding computed via local linear embedding. First, we load the relevant libraries:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import make_swiss_roll
from sklearn.manifold import LocallyLinearEmbedding
```

Next, we generate some data, specifying the number of neighbors for this problem equal to 10:

```
n_samples = 1000
n_neighbors=10
```

We now construct the data which will represent the classic “swiss roll”:

```
X, _ = make_swiss_roll(n_samples=n_samples)
X, _
Out[105]:
array([[ 7.80833394,  0.48066359, 11.03910558],
       [ 1.59837552,  2.65730412,  7.47431048],
       [-8.91500526,  3.57885813,  2.20113125],
       ...,
       [11.34728421, 11.47912824, -4.45900994],
       [ 4.7570084 , 13.86156279,  5.30120852],
       [ 2.33270015,  5.81407082,  7.16947349]])
```

Our goal is to reduce the dimensionality of the swill roll manifold into a two-dimensional structure. Now we perform the LLE and plot the result for two components (`n_components=2`):

```
lle = LocallyLinearEmbedding(n_neighbors=n_neighbors, n_components=2)
X_reduced = lle.fit_transform(X, _)
import matplotlib.pyplot as plt
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], cmap='viridis')
Out[54]: <matplotlib.collections.PathCollection at 0x18fd4114b10>
```

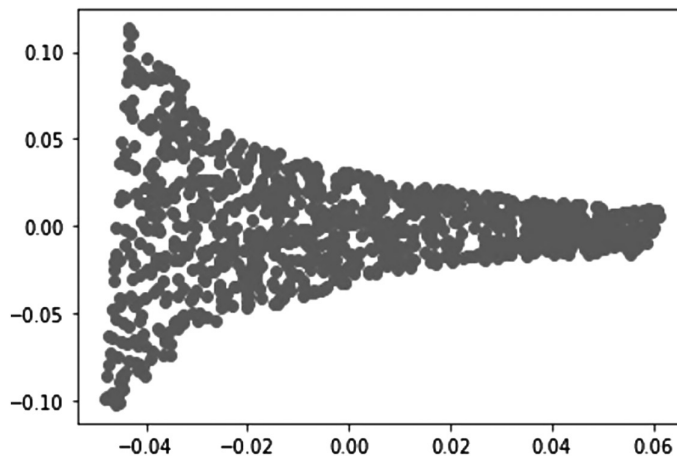


Figure 11.19 Locally linear transformed data.



We can see from Figure 11.19 that the reduced data, that is, the locally linear transformed data, captures an embedding in the original data that is much more interesting had we simply performed something like classic “linear” PCA. Classical PCA would have never “uncovered” such an interesting nonlinear embedding.

Locally linear embedding in R can be performed using the package *Rdimtools* (You & Shung, 2022).

### 11.14 Isomap

Recall that classical multidimensional scaling (MDS) attempts to output coordinates (i.e., the *principal coordinates*) that seek to preserve pairwise distances. If the true manifold is complex, then classical MDS using Euclidean distances may not recover it. What Isomap (*isometric feature mapping*) does is use the *geodesic distance* (not Euclidean distance as is usually used in classical MDS), which, recall, is essentially “the shortest curve length which respects the geometry of the data” (Ihler, 2003). *Isomap seeks to preserve this geodesic distance when reducing the dimensionality*. That is, if two points are distant in higher dimensions, we would like to preserve that distance in lower dimensions, and vice versa. The distance  $x_i - x_j$  is sought to be preserved when transformed to a geodesic distance. According to its innovators, “ours [Isomap] efficiently computes a globally optimal solution, and, for an important class of data manifolds, is guaranteed to converge asymptotically to the true structure” (Tenenbaum et al., 2000, p. 2319). The convergence is justified by a theorem that holds that as  $n \rightarrow \infty$  distances on a manifold (so-called *graph distances*), increasingly provide better and better approximation to true geodesic distances (Tenenbaum et al., 2000). This result is assured because the approximation can be achieved by much shorter Euclidean distance “hops.”

Since by definition a manifold is a structure that is locally Euclidean, meaning once more that at any area on the structure it is flat (again, think of the earth as an example), it stands that the  $k$ -nearest neighbors in the original high-dimensional space should, ideally, also be the  $k$ -nearest neighbors on the manifold. Hence, the first step in Isomap is determining which points are neighbors based on the familiar notion of distance,  $d_i = x_i - x_j$ . What constitutes a “neighbor” can be determined by a flexible radius  $\varepsilon$  parameter. Not surprisingly, for points that are very close, Euclidean distances approximate geodesic distances quite well, however this is not the case for points that are increasingly further apart. Selection of a suitable neighborhood size is a crucial input to conducting Isomap.

Next, geodesic distances are estimated between all pairs of points  $i, j$ . The more dense the sampling of points, the better estimates one will obtain. *Dijkstra's algorithm* (Dijkstra, 1959) or competing algorithm can be used to estimate the shortest paths, which are approximated by the sum of local distances. Lastly, classical MDS is then computed on the geodesic distances. Essentially then, *Isomap performs a multidimensional scaling (eigen analysis) on the high-dimensional data, but instead of using Euclidean distances as classical MDS, it performs the MDS on geodesic distances*. Using geodesic distances carries the benefit of capturing nonlinear and complex dimensions that could otherwise not be “tapped” using Euclidean distances.

The cost function that is minimized in Isomap is given by

$$E = \|\tau(D_G) - \tau(D_Y)\|_{L^2} \quad (11.23)$$

where  $D_G$  is a matrix of geodesic distances,  $D_Y$  is a matrix of Euclidean distances resulting from the embedding and preserving the manifold's essential geometry, and  $\tau$  is an operator that

converts distances into inner products. See Tenenbaum et al. (2000, p. 2322) for the precise details regarding the computation of  $\tau$ .

We can see then how Isomap is related to MDS. But what about LLE? Where does LLE fit into all this? While LLE and Isomap are similar, *LLE makes no effort to estimate the true geodesic distances, whereas Isomap does*. LLE instead attempts only to measure local linear “patches.” Furthermore, Isomap makes additional assumptions over LLE, such as the data being located in a convex region of  $\mathbb{R}^q$  and that the data was originally embedded in  $\mathbb{R}^p$  by some isometric transformation, both assumptions which may not hold for any particular data set. The assumption of convexity, however, has been challenged recently (see Trosset & Buyukbas, 2020 for details).

For all of its strengths, the problem with Isomap is that it first relies on finding “neighboring” distances on the manifold, which could be faulty. In seeking out the nearest neighbors, if  $k$  is too small, then the graph may not even be connected. If  $k$  is too large, then the graph may be too dense giving us wrong distances on the manifold. That is, finding the “right”  $k$ , if it exists at all, is quite hard. *Stochastic Neighborhood Embedding* (SNE), as we will soon see, attempts to address and solve some of these problems. Another issue with Isomap, as was true of MDS as well, is that neither approach learns a general mapping *function* that can be applied to new points. Hence, if a new data point is added to the original data point, the entire algorithm needs to generally be repeated to incorporate that point (Alpaydin, 2020).

*A key difference between local linear embedding and Isomap is that LLE makes no effort to estimate the true geodesic distances, whereas Isomap does.*

### 11.14.1 Isomap in Python

The following is an example of Isomap in Python. We first import the relevant libraries:

```
from sklearn.datasets import make_s_curve
from sklearn.manifold import Isomap
import matplotlib.pyplot as plt
```

Next, we generate the S-curve data:

```
X, color = make_s_curve(n_samples=1000, random_state=42)
X
Out[148]:
array([[ -0.92552951,  0.37026586,  0.6213245 ],
       [ -0.89404266,  1.08380189, -1.44798184],
       [  0.81637223,  1.74589167, -1.57752609],
       ...,
       [  0.27761546,  0.13791603,  1.96069228],
       [ -0.89201987,  0.11410944, -1.45199618],
       [ -0.48720261,  0.56437415,  0.12671104]])
```

Now, we run the Isomap and obtain a plot of the results (Figure 11.20), both the original and transformed data resulting from the dimensionality reduction in two dimensions:

```
isomap = Isomap(n_neighbors=10, n_components=2)
X_isomap = isomap.fit_transform(X)

fig, ax = plt.subplots(1, 2, figsize=(12, 5))
ax[0].scatter(X[:, 0], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax[0].set_title('Original 3D Data')

ax[1].scatter(X_isomap[:, 0], X_isomap[:, 1], c=color, cmap=plt.cm.Spectral)
ax[1].set_title('Isomap Reduced 2D Data')

plt.show()
```

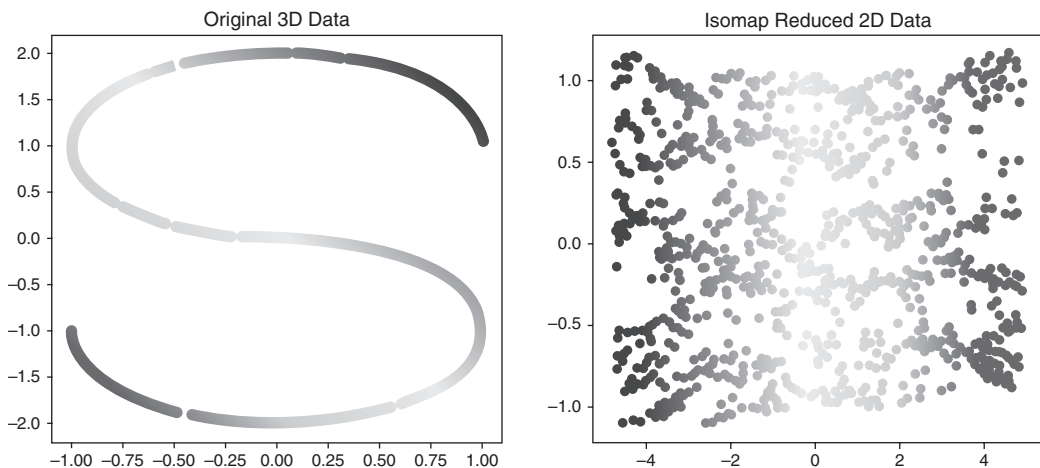


Figure 11.20 Original and Isomap reduced data in two dimensions (contrasting colors not reproduced in plots).

*Isomap* can also be fit in R using the *Rdimtools* package as well as the package *ProjectionBasedClustering*.

### 11.15 Stochastic Neighborhood Embedding

Up to now, we have dealt with techniques that have a deterministic idea of what a “neighbor” is. That is, whether with nearest neighbor (NN) or k-nearest neighbor (KNN), or Isomap, when we designate  $k$ , what constitutes a “neighbor” is thus determined. As an example, the neighbor to your house or apartment is determined. If you use  $k = 1$ , then your *neighbor is your neighbor*, there is no doubt about that. *It is what it is*. This is what we mean by your neighbor being deterministically determined. You don’t look at the house next to yours and ask yourself the probability that that house is your neighbor. As we will see, however, this is precisely what stochastic neighborhood embedding does.

*Stochastic Neighborhood Embedding*, as the name suggests, imposes a *stochastic* element to what constitutes a neighbor. Recall that “stochastic” essentially means “random.” That is, if we specify  $k = 1$  in SNE, then *your neighbor is not necessarily your neighbor*. Instead, the house a few feet away from you is *probably your neighbor*. That is, for every pair of points, a probability is assigned that they are neighbors such that close points are more likely (i.e., *probably*) to be neighbors than points that are more distant. That is, instead of *deterministically* determining neighbors, we instead seek a *stochastic neighborhood distribution*.

Stochastic neighbor embedding operates by translating high-dimensional Euclidean distances between data points into conditional probabilities that correspond to similarities (van der Maaten & Hinton, 2008). How is similarity between data points evaluated? Similarity is transformed into a conditional probability that a given data point,  $x_i$ , would choose  $x_j$  as its neighbor if neighbors were selected in proportion to their probability density under a Gaussian centered at  $x_i$ . If a data point is nearby, then  $p_{j|i}$  will be relatively high, whereas for data points that are separated a great deal,  $p_{j|i}$  will be increasingly smaller. The precise conditional probability we are modeling is given by the following:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad (11.24)$$

where  $x_i$  and  $x_j$  are two distinct data points and  $\sigma_i^2$  is the variance from the familiar Gaussian distribution, which, as a tuning parameter, effectively sets the size of the given neighborhood and can be different for each point  $x_i$ . Smaller values of  $\sigma_i^2$  will increase the probability that  $x_i$  will choose nearby  $x_j$  while larger values for  $\sigma_i^2$  will make that probability increasingly smaller since a higher  $\sigma_i^2$  implies more dispersion of points. The value for  $\sigma_i^2$  is determined by SNE conditional upon how the user sets another parameter known as the *perplexity*,

$$\text{perplexity}(P_i) = 2^{H(P_i)} \quad (11.25)$$

where  $P_i$  is the probability distribution of a sample such that  $P_i = \sum_j p_{ij}$  and where

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i} \quad (11.26)$$

is the *Shannon entropy* of  $P_i$  (Rodosthenous, Shahrezaei and Evangelou, 2024).  $H(P_i)$  is a measure of *information gain* based on knowing  $x_i - x_j$ . The perplexity parameter  $P_i$  can be considered the *effective number of neighbors*, which is usually chosen by the user in the range of 5 to 50 (van der Maaten & Hinton, 2008).

In computing  $p_{j|i}$ , we need a value on which the normal distribution is centered so we will choose  $x_i$ . Hence, we see in the numerator of  $p_{j|i}$  that the squared distance between points  $x_i$  and  $x_j$  is reflected by  $\|x_i - x_j\|^2$ . Ignoring the constant  $2\sigma_i^2$  for a moment, we see that  $p_{j|i}$  is nothing more than a ratio of terms. The first term, the numerator, consists of (squared) distance measured from  $x_i$  to  $x_j$ . We compare this distance to the sum of (squared) distances across all points  $x_k$  in the denominator. It is easy to see the fraction must be equal to or less than 1, making it a legitimate probability. Hence, when we compute  $p_{j|i}$ , we are computing the probability of  $x_j$  being a “neighbor” given (i.e., conditional) that it is  $x_i - x_j$  deviation from  $x_i$ . As the deviation  $x_i - x_j$

decreases relative to the overall sum of deviations in the denominator, it becomes more likely (i.e.,  $p_{j|i}$  increases) that it is regarded as a neighbor. Note that  $k \neq i$  so that a point cannot choose itself as its neighbor.

Having performed the dimension reduction, we now define a new measure of similarity based on the new lower-dimensional scores  $y$  :

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)} \quad (11.27)$$

where  $q_{j|i}$  is now the probability of point  $i$  selecting point  $j$  as its neighbor in the *lower-dimensional space*. Recall that in computing  $p_{j|i}$  earlier, we were able to freely choose values for  $\sigma_i^2$ .

In the embedding of  $q_{j|i}$ ,  $\sigma_i^2$  is set to  $1/\sqrt{2}$  (van der Maaten & Hinton, 2008, p. 2581) though other values can also be used.

But how is the new, now *embedded* point,  $y_i$  obtained? That is, how do we best determine the low-dimensional images  $y_i$ ? By what criterion or cost function  $C$ ? It is obtained by minimizing what is called the *Kullback Leibler (KL) divergence*. That is, SNE seeks to minimize the sum of Kullback-Leibler divergences over all new embedded data points using a gradient descent method, with cost function equal to:

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \quad (11.28)$$

where in (11.28)  $P_i$  is the conditional probability over all other data points conditioned on  $x_i$  and  $Q_i$  is the conditional probability distribution over all other map or “constructed” points given the mapped point  $y_i$  (van der Maaten & Hinton, 2008). Remember, we have two conditional probabilities now;  $P_i$  is based on the original high-dimensional data, whereas  $Q_i$  is based on the lower-dimensional embedding after performing the dimension reduction. The idea of SNE is for  $P_i$  and  $Q_i$  to be *as close as possible*. That is, the probability of  $j$  being selected given  $i$  should be as similar as possible as we go from the original distribution of data to the new, embedded distribution.

More specifically, the key to understanding  $KL$  is in understanding  $p_{j|i}/q_{j|i}$  where recall  $p_{j|i}$  is the conditional probability computed earlier based on the high-dimensional data, and  $q_{j|i}$  is a distance function based on the transformed induced lower-dimensional projections. Since  $\log(p_{j|i}/q_{j|i}) = 0$  only when  $p_{j|i} = q_{j|i}$ , this implies that  $KL$  is minimized when the embedding is successful at maintaining the distance between points  $x_i$  and  $x_j$  in the reconstruction  $y_i$  and  $y_j$  based on the dimension reduction.  $KL(P_i \| Q_i)$  is always equal or greater than zero and cannot be negative.

To minimize  $KL$ , we need to take the partial derivative of the cost function,  $C$ ,  $\frac{\partial C}{\partial y_i}$  (Hinton & Roweis, 1998, p. 2582). The result is

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (y_i - y_j)(p_{i/j} - q_{i/j} + p_{j/i} - q_{j/i}) \quad (11.29)$$

The goal of SNE then, is to find points that minimize  $KL(P, Q)$ , where  $KL$  essentially minimizes the divergence between probability distributions of the input space (i.e., the original high-dimensional

data) and that of the output space (i.e., the reduced low-dimensional data). Defining a stochastic neighborhood this way, however, suffers from the *crowding problem*, which we discuss shortly.

### 11.15.1 SNE in R

The package *Rdimtools* (You & Shung, 2022) can perform SNE with relative ease. Below we adapt an example from the package on the iris data, where we also observe the effect of changing the *perplexity* parameter on the outcome. First, we load the iris data:

```
> library(Rdimtools)
> data(iris)
> set.seed(100)
> subid = sample(1:150, 50)
```

We sample 50 cases by *subid* from the range of 1 through 150, then create a matrix *X* based on this sample:

```
> X = as.matrix(iris[subid, 1:4])
> head(X)
```

|     | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|-----|--------------|-------------|--------------|-------------|
| 102 | 5.8          | 2.7         | 5.1          | 1.9         |
| 112 | 6.4          | 2.7         | 5.3          | 1.9         |
| 4   | 4.6          | 3.1         | 1.5          | 0.2         |
| 55  | 6.5          | 2.8         | 4.6          | 1.5         |
| 70  | 5.6          | 2.5         | 3.9          | 1.1         |
| 98  | 6.2          | 2.9         | 4.3          | 1.3         |

The following generates labels for the iris data, more specifically, the *species* variable:

```
> label = as.factor(iris[subid, 5])
```

We confirm that labels have been created:

```
> label
```

|      |           |            |           |            |            |            |
|------|-----------|------------|-----------|------------|------------|------------|
| [1]  | virginica | virginica  | setosa    | versicolor | versicolor | versicolor |
| [7]  | virginica | setosa     | setosa    | virginica  | versicolor | setosa     |
| [13] | setosa    | versicolor | virginica | setosa     | setosa     | versicolor |

We now perform the SNE, first with *perplexity* = 5 and plot the result (Figure 11.21):

```
> out1 <- do.sne(X, perplexity=5)
> plot(out1$Y, pch=19, col=label, main="SNE perplexity=5")
```

We now run the *SNE* with a *perplexity* equal to 50 and again plot the result (top, right):

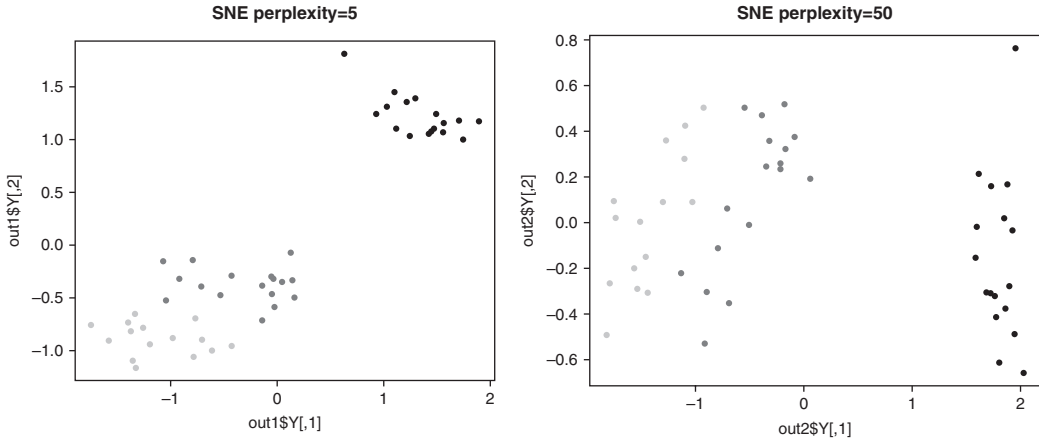


Figure 11.21 SNE with perplexity 5 (left) and perplexity 50 (right).

```
> out2 <- do.sne(X, perplexity=50)
> plot(out2$Y, pch = 19, col=label, main="SNE perplexity=50")
```

We can see from the plots that changing the perplexity can have an effect on the outcome. What is consistent in both plots, however, is that one species (*setosa*) clearly separates itself from the other two.

*Stochastic Neighborhood Embedding, or “SNE,” imposes a stochastic or random element into what constitutes a neighbor rather than a deterministic one as many other nearest-neighbor methods do. In SNE, for data points that are close, they have a higher probability of being neighbors than if they are distant. That is, SNE establishes stochastic neighborhood distributions, where closeness is defined by reference to Kullback-Leibler (KL) divergence. The goal of SNE is to find points that minimize KL divergence.*

### 11.16 *t*-SNE

*t*-SNE, or *t*-Distributed Stochastic Neighbor Embedding, is related to the technique of SNE; like SNE, *t*-SNE essentially converts high-dimensional Euclidean distances between data points into conditional probabilities. These probabilities then represent similarities between data points in higher dimensions. The “*t*” in *t*-SNE stands for the *t*-distribution, whereas traditional SNE, as we saw, uses the Gaussian (i.e., *z*) distribution. That is, SNE is, essentially, “*z*-SNE” where  $q_{ji}$  is now defined,

$$q_{ji} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}} \quad (11.30)$$

The idea behind *t*-SNE is, as was the case for SNE, to convert high-dimensional Euclidean distances into conditional probabilities that represent similarities. The technique provides a way to

visualize high-dimensional data by essentially giving each data point a “location” in a two or three-dimensional map (van der Maaten & Hinton, 2008). Given that the aim of dimensionality reduction is to preserve as much as possible the structure of high-dimensional data into a lower-dimensional map,  $t$ -SNE does this remarkably well, presumably better than SNE and is easier to optimize than SNE (van der Maaten & Hinton, 2008). The problem with SNE is that it is hampered by a cost function that is difficult to optimize and also by a problem referred to generally as the *crowding problem*.

The crowding problem is the idea that data points in higher dimensions may look far from each other, but when we reduce the dimensionality, say from two-dimensional to one-dimensional, we essentially do not have enough “room” to accommodate all neighbors to those points and they get “squashed” toward the center of the plot. Essentially, it’s another interpretation of the curse of dimensionality problem. This is one of the biggest problems with SNE.  $t$ -SNE allows more “wiggle room” to place points through, in part, using a heavy-tailed distribution ( $t$ ) instead of the Gaussian. For details on the crowding problem, see van der Maaten and Hinton (2008). The crowding problem is not unique to  $t$ -SNE however and occurs in other manifold procedures such as Sammon mapping.  $t$ -SNE aims to improve on these limitations.

Even the most common methods of reducing dimensionality, such as Sammon mapping, Isomap, and others, while they perform relatively well on artificial data sets, they are not that great at visualization (van der Maaten & Hinton, 2008). The main problem is that these techniques are not capable of maintaining both the local and global character of the data simultaneously. The benefit of  $t$ -SNE is that it is able to assess much of the local structure of the high-dimensional data while also somewhat uncovering global structure such as the presence of clusters (van der Maaten & Hinton, 2008).

#### **11.16.1 Performance of $t$ -SNE to Other Techniques**

Maaten and Hinton (2008) compared the performance of  $t$ -SNE to many other nonparametric visualization techniques such as Sammon mapping and Isomap, as well as locally linear embedding and found visualization via  $t$ -SNE to generally be superior in almost all cases. However, as always, there is no “free lunch” and no single manifold technique is guaranteed to universally work better than others (Izenman, 2008). Much depends on the data to which the given technique is being applied. A “multi-view” version of  $t$ -SNE and Isomap has also been proposed (Rodosthenous et al., 2024), where multi-view data comprises of multiple kinds of data generated by the same sample. Multi-view versions of SNE, Isomap, as well as LLE were found to produce better visualizations when compared to single-view versions of the same techniques. See Rodosthenous et al. (2024) for details.

#### **11.16.2 $t$ -SNE in Python**

The following is a demonstration of  $t$ -SNE in Python on the iris data:

```
from sklearn.manifold import TSNE
```



```
import plotly.express as px
df = px.data.iris()
```

We print a few cases of the iris data:

```
df
Out [75]:
```

|   | sepal_length | sepal_width | ... species | species_id |
|---|--------------|-------------|-------------|------------|
| 0 | 5.1          | 3.5         | ... setosa  | 1          |
| 1 | 4.9          | 3.0         | ... setosa  | 1          |
| 2 | 4.7          | 3.2         | ... setosa  | 1          |
| 3 | 4.6          | 3.1         | ... setosa  | 1          |
| 4 | 5.0          | 3.6         | ... setosa  | 1          |

We next define the features:

```
features = df.loc[:, :'petal_width']
```

The function of `df.loc[:, :'petal_width']` selects features up to and including `petal_width`:

```
features
Out [77]:
```

|   | sepal_length | sepal_width | petal_length | petal_width |
|---|--------------|-------------|--------------|-------------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         |

We now perform the *t*-SNE, requesting a solution in two dimensions (`n_components = 2`), and plot the result (Figure 11.22):

```
tsne = TSNE(n_components=2, random_state=0)
projections = tsne.fit_transform(features)

fig = px.scatter(
    projections, x=0, y=1,
    color=df.species, labels={'color': 'species'}
)
fig.show()
```

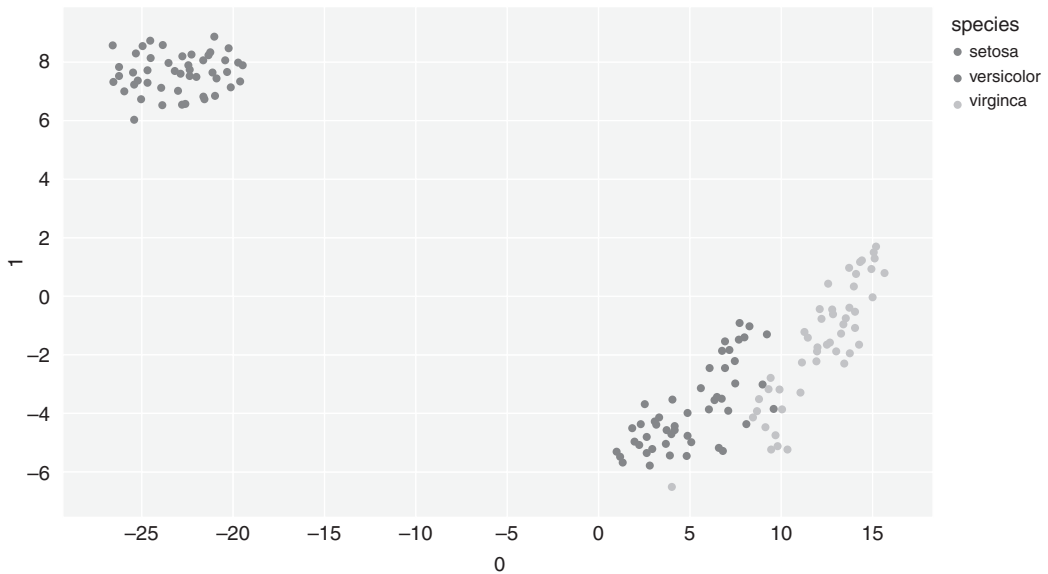


Figure 11.22 *t*-SNE projections on the iris data.

We note that *setosa* separates itself quite well from both *versicolor* and *virginica*, which is quite consistent with what we find with other dimension-reduction techniques on the iris data.

### 11.16.3 *t*-SNE in R

The package *Rtsne* (Krijthe, 2015) performs *t*-SNE. We adapt an example from the package's documentation once more using the iris data. We first remove potential duplicates from the data:

```
> iris_unique <- unique(iris)
```

We next create a matrix made up of features sepal length through to petal width:

```
> iris_matrix <- as.matrix(iris_unique[,1:4])
> set.seed(42)
```

We now perform the *t*-SNE:

```
> library(Rtsne)
> tsne_out <- Rtsne(iris_matrix, pca=FALSE, perplexity=30, theta=0.0)
```

As was the case for SNE, the *perplexity* controls how many nearest neighbors are taken into account when constructing the embedding in the lower-dimensional space (Krijthe, 2015). *Theta* is a numeric measure of speed/accuracy trade-off, such that an increase in *theta* amounts to less accuracy. The default for *exact t*-SNE is 0.0. See the package for further details.

```
> plot(tsne_out$Y, col=iris_unique$Species, asp=1)
```

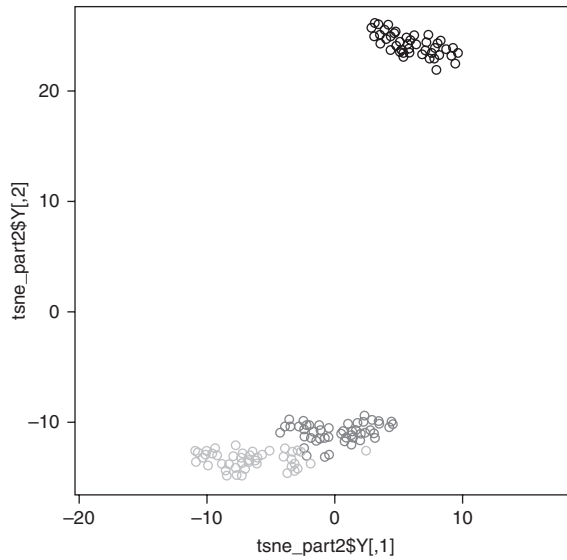


Figure 11.23 t-SNE plot with perplexity set at 30.

As mentioned, the perplexity parameter, set at 30 in Figure 11.23, controls how many nearest neighbors are taken into account when constructing the embedding in the lower-dimensional space. Instead of 30, let's set it at 10 (Figure 11.24), and notice that the distribution of points has changed:

```
> tsne_out <- Rtsne(iris_matrix, pca=FALSE, perplexity=10, theta=0.0)
> plot(tsne_out$Y, col=iris_unique$Species, asp=1)
```

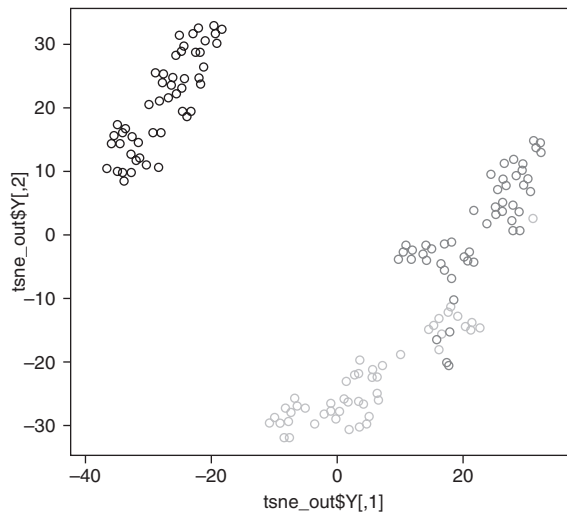


Figure 11.24 t-SNE with perplexity set at 10.

The package *ProjectionBasedClustering* (Thrun et al., 2024) can also be used for performing *t*-SNE.

### 11.17 Manifold Learning and Beyond

This chapter has been about dimension-reduction techniques, that is, to find a lower-dimensional embedding from higher-dimensional data. Dimension reduction can be seen as a special case of the wider *blind source separation* or *manifold learning* paradigm. The focus of the chapter was first on principal components analysis since it is the simplest of the dimension-reduction techniques, the most used, and understanding it is the gateway to understanding extensions of the PCA approach as well as more modern techniques for dimension reduction.

In the following chapter, we will discuss yet another approach to dimension reduction, that of the *exploratory factor analysis model*. Though we have mentioned exploratory factor analysis in the current chapter, it is such an extensive and popular technique that it merits its own chapter.

PCA has its origins with Pearson in 1901 and Hotelling in 1933. As discussed in the chapter, a limitation of PCA is that it can only extract *linear components* and also can be sensitive to multivariate outliers. *Robust PCA* (Locantore et al., 1999) and *Expectation Robust PCA* (Serneels & Verdonck, 2008) have been proposed as a means for dealing with outlying data. Other methods of PCA include *Local PCA*, *Generalized PCA*, and *Multi-linear PCA*. *Principal Components Analysis* by Jolliffe (2002) is perhaps misnamed, as it provides an extensive survey of not only PCA but also many other techniques related to PCA. These include *discriminant analysis*, *canonical correlation*, and even *neural networks*.

In our survey of dimension reduction and manifold learning, it is important to realize that we have merely skimmed the surface. As is the case with most statistical and machine learning topics, entire books and monographs have been written on the topic, many of which we have not had time or space to treat in this book. For instance, *latent semantic analysis* (LSA) is yet another dimension-reduction technique, this time for finding an embedding of words to decipher semantic meaning and thus specifically designed for text data. It is essentially an SVD on text data. As an example, suppose an author uses a group of words that have a “dark” tone to them. Do these words have an underlying dimension? LSA might be useful in uncovering such dimensionality. Many of these techniques are in one way or another related to SVD, and much statistical research has been targeted toward demonstrating parallels and equivalencies.

*Non-negative matrix factorization* is another dimension-reduction technique that once more creates a new basis for data that is originally in high dimensions. Such a basis is also guaranteed to be non-negative, which is where NMF gets its name. That is, the components and coefficients are non-negative (Müller & Guido, 2017). However, in NMF, the basis will not necessarily be orthonormal as is the case in PCA, however orthogonal versions of NMF have also been proposed (Yang & Xu, 2021). However, like PCA, “vanilla” NMF is restricted to uncovering only linear dimensions. Essentially, NMF reduces the data down to a non-negative weighted sum and may lead to more interpretable components than PCA. The procedure is effective at providing semantic interpretability for text analysis and dimension reduction (see Aggarwal, 2015, p. 191).

At a more advanced level, sometimes manifolds contain loops (e.g., torus or circle) in them which makes traditional dimensionality much more difficult, if not prohibitively challenging. Techniques for reducing the dimensionality of such complex manifolds have been suggested (Lee & Verleysen, 2005). For an excellent overview of various dimension-reduction techniques and where current developments stand, see Nanga et al. (2021).

### 11.18 Chapter Review Exercises

1. What is a *principal component*? How is a principal component different from any other linear combination?
2. For a set of observed variables on which a PCA is conducted, how many components are typically produced? When would potentially *less* be generated?
3. On the iris data, conduct a principal components analysis on all four features. Are any of the components “interesting?”
4. Demonstrate on the analysis performed in #3 that PCA is not *scale-invariant*, that is, conduct the PCA on both the *covariance* and *correlation* matrices and compare results.
5. How is *independent components analysis* (ICA) different from PCA? How are both different from *probabilistic PCA*? Explain.
6. Why is *kernel PCA* used for nonlinear dimension reduction? What is it about employing kernels that allows for such nonlinearity?
7. How is *principal curves* different from principal components? How are they similar?
8. In what way does *multidimensional scaling* relate to PCA? In what way does MDS relate to *Isomap*? Discuss.
9. How does a reduction in the *stress function* for MDS indicate a preferred mapping? That is, unpack and explain the stress function in MDS.
10. Define what is meant by a *manifold*. Can a simple line be considered a manifold? Why or why not? How about a more complex curve?
11. Cite one difference between *locally linear embedding* and that of *Isomap*.
12. Compare *stochastic neighborhood embedding* to *t-SNE*. What is their essential difference? How does *t-SNE* go about addressing the so-called “crowding problem?”

## 12 Exploratory Factor Analysis

In the previous chapter, we surveyed the method of principal components analysis, or PCA, as well as many extensions, derivatives, and special cases of the PCA method, including nonlinear approaches. The theme of all of these techniques was that of *dimension reduction* and could be considered special cases of an even more general problem, that of *manifold learning*. Whether it was principal components, multidimensional scaling, or other reduction technique, the overarching goal was to learn something about the hypothesized underlying lower-dimensional manifold that might subsume the observed data. That is, the goal was to seek out a lower-dimensional *embedding*. The purpose of PCA, in particular, was to uncover dimensionality in the variables by transforming from one *basis* to another such that the new variables (called *components*) have desirable properties, the most significant of which was to maximize variance. The first principal component was that which maximized the variance in the variables, and each component thereafter maximized a decreasing proportion of the variance. Components were constrained to be *orthogonal* to one another and the hope was to explain most of the variance in the variables by the first few components. Other techniques such as *independent components analysis* (ICA) held the stronger property that components be independent and not only uncorrelated. ICA was useful for such things as extracting the independent sources of variation from signals, speech, or images and proved especially useful for so-called *blind source separation* problems. Both classical PCA and ICA are examples of dimension-reduction techniques such that we believe the true (or at least our working hypothesis) dimensionality of the data lies in a lower *subspace* than in the original empirical representation. We noted that if the true dimensionality, or true manifold underlying the data was not linear, then a classical PCA could actually mislead more than it could enlighten. In this respect, PCA does not uncover *the* dimensionality underlying a set of data; it uncovers the *linear* dimensionality.

In this chapter, we survey the technique of *exploratory factor analysis* (EFA). Factor analysis is yet another dimension-reduction technique that we could have easily included as one of the several approaches surveyed in the former chapter. As we will see later, principal components analysis can actually be conceived mathematically as a special case of the wider factor analysis model. And though factor analysis is sometimes mistakenly conflated with PCA as being equivalent, it should be emphasized at the outset that *factor analysis is not the same as PCA*. First, factor analysis models *observed* variables as a function of *latent* variables, whereas PCA models components (loosely, “latent variables”) as a function of observed variables. Second, most versions of factor analysis hypothesize a *probabilistic model* that naturally includes an error or noise term, whereas classical PCA does not. In classical PCA, there is no “noise,” whereas in classical factor analysis, observed variables are modeled as a function of factors *plus* an error term. The advantage of factor analysis being probabilistic is, for one, that we can build more complex models such as *mixture* and *nonlinear factor analysis models* (Murray, 2022). Some software programs such as SPSS

routinely provide PCA results first before delivering on factor analysis findings and can lead some researchers to believe PCA results are *equivalent* to those of factor analysis, which, as we have noted, are not.

## 12.1 Why Treat Factor Analysis in Its Own Chapter?

Since we have said that factor analysis can be conceived as just another dimension-reduction technique, why devote a separate chapter to it instead of including it in the former chapter? There are a few reasons for treating factor analysis independently in a distinct chapter. These reasons include the fact that factor analysis has a long tradition in many social sciences (e.g., psychology) as a *centerpiece methodology*, and in many courses on multivariate analysis in the social sciences is given maximal attention as a methodology in its own right. Factor analysis grew out of a research context that initiated a tremendous amount of growth and innovation in what are generally referred to as *latent variable models*. Originally founded by Charles Spearman (1904) in psychometrics, the concept of a *latent variable* formed the very bedrock of many techniques that grew out of factor analysis, such as *structural equation modeling*, which we survey in the following chapter.

Hence, factor analysis has its own unique historical lineage and germination and arose to address very specific substantive research problems where defining and tapping into constructs proved quite challenging, difficult, and, at many times, contentious. For reasons such as these, to treat it only on a technical level as an alternative to PCA or yet as just another approach to dimension reduction, we believe, would be a mistake. Despite its controversy historically, it is well-deserving of its own chapter for its devoted emphasis on the *measurement problem* which has dominated the social sciences in which it first arose and is responsible for giving birth to numerous other latent variable modeling techniques.

Toward the end of the chapter we will draw links between it and other techniques to show how factor analysis relates to these other dimension reduction and manifold learning methods, especially as contextualized in modern machine learning. As we will see, in a machine learning context, factor analysis can be best conceived as yet another *autoencoder*, more specifically a *variational autoencoder* due to its inclusion of an error or “noise” variable. Before we generalize to such lengths, however, we begin with the model from the bottom up by noting conceptual parallels it has, not with principal components, but instead with the classic *regression model*.

## 12.2 Common Orthogonal Factor Model

The common orthogonal factor model is given by

$$\mathbf{x} = \boldsymbol{\mu} + \mathbf{\Lambda}\mathbf{f} + \boldsymbol{\varepsilon} \quad (12.1)$$

where  $\mathbf{x}$  is a vector of observed, or “manifest” variables that we seek to “explain” by the factor model,  $\boldsymbol{\mu}$  is an overall mean of the observed variables,  $\mathbf{\Lambda}\mathbf{f}$  are weights or “loadings” in  $\mathbf{\Lambda}$  applied to a vector of factors in  $\mathbf{f}$  and  $\boldsymbol{\varepsilon}$  is a vector of errors, that is, variation unexplained by the factor model. Of special note here is  $\mathbf{f}$ , a vector of *unobserved* variables, not *observed* variables as in  $\mathbf{x}$ . This is what is so special about the factor model, such that it attempts to account for observed variables by hypothesizing unobserved underlying variables. Note that this is not what occurs in PCA. Principal components analysis simply undergoes a rotation of axes resulting from an eigen decomposition; there is no *model equation* per se as in factor analysis. Note as well that  $\boldsymbol{\varepsilon}$  is a vector of unobserved variables just as it was in the classic linear model. The vector  $\boldsymbol{\varepsilon}$  is usually

hypothesized to contain “specific” or “unique” variances, that is, variation that is unaccounted for by the latent variables in the factor model.

### 12.2.1 Factor Analysis Is a Regression Model

One of the first things to note about the factor analysis model is that it is, in actuality, a *regression model*, but not the classic regression of previous chapters. Defining it as a regression model simply means we are modeling an observed response by weighted predictors, that is, *regressing the observed responses on the weighted set of predictors*, but this is a very general idea; what is specifically being regressed on what will determine the exact *type* of regression model. Hence, concluding that factor analysis is a regression model only implies that it has a regression *characteristic* to it, no more. To appreciate the differences between the factor analysis model and that of classical least-squares regression, it behooves us to compare them side-by-side:

$$\mathbf{x} = \boldsymbol{\mu} + \mathbf{\Lambda}\mathbf{f} + \boldsymbol{\varepsilon} \quad \mathbf{y} = \mathbf{x}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

We note similarities and differences between the models; both models have an observed vector as output of the function; in the factor analysis model, that vector is  $\mathbf{x}$  while in the regression model that vector is  $\mathbf{y}$ . Both models contain an *error* or *noise* term,  $\boldsymbol{\varepsilon}$ . Where the models differ is in the middle terms of each equation. In regression, note that we are weighting an observed vector  $\mathbf{x}$  by  $\boldsymbol{\beta}$  whereas in the factor analysis model, we are weighting *unobserved* factors  $\mathbf{f}$  by  $\mathbf{\Lambda}$ . Both models are regression models, but *the factor analytic model contains unobserved factors as an explanatory variable*, whereas the regression model contains observed variables as explanatory variables. This is the key difference between the models.

*The factor analysis model is a regression model, however unlike the classic regression model in which observed variables are regressed onto other observed variables, the factor analysis model features observed variables regressed onto unobserved factors or so-called latent variables.*

### 12.2.2 Assumptions Underlying the Factor Analysis Model

As in virtually all statistical modeling, models come with them a set of assumptions, assumptions that if not satisfied, can drastically weaken the application of the model and the conclusions that can be drawn when applying it to data. For the common factor analytic model, the assumptions are generally that  $E(\mathbf{f}) = \mathbf{0}$ , that is, the mean of factors is equal to 0, and  $E(\mathbf{ff}') = \mathbf{I}$ , the covariance of factors is an identity matrix. Note that this is only an assumption of the *orthogonal factor model*. When we relax this assumption and allow factors to covary, such gives rise to the *oblique factor model*.

We also usually assume the errors have a mean equal to 0, that is,  $E(\boldsymbol{\varepsilon}) = \mathbf{0}$ , and we assume the errors do not covary. We also typically assume that the covariance between the error term and the factors is equal to 0, that is,  $E(\boldsymbol{\varepsilon}\mathbf{f}') = \mathbf{0}$ . When we consider the confirmatory factor analytic model in the following chapter, we will see how these assumptions are not set in stone and can be relaxed such as to generate a factor model that is much more flexible.

### 12.2.3 Implied Covariance Matrix

The factor analysis model we have described implies the following structure to the covariance matrix,



$$\Sigma = \Lambda\Lambda' + \Psi \quad (12.2)$$

That is, the covariance of observed variables is equal to  $\Lambda\Lambda'$  loadings plus a matrix of errors. The above equality is *implied* by defining the factor analysis model as we did earlier such that we can reproduce the covariance matrix  $\Sigma$  based on  $\Lambda\Lambda' + \Psi$ . Recall that we said errors were uncorrelated, which implies that  $\Psi$  will be an identity matrix, with values of one along the main diagonal and zeroes everywhere else. Given that  $\Psi$  is set this way, this further implies that the covariances in  $\Sigma$  must be accounted for by the contents of  $\Lambda$ . Contrast this for a moment to PCA in which there is no  $\Psi$  term to “explain” the covariance matrix, which implies that PCA will attempt to not only explain covariances, but also variances (Alpaydin, 2020). Another way of looking at this is to allow  $\Psi$  to be in the PCA set-up, but set it to zero, that is,  $\Psi = 0$  for the PCA model but  $\Psi \neq 0$  for the factor analysis model. As we will discuss later, if all error variances are assumed to be equal, that is,  $\Psi = \psi\mathbf{I}$  (i.e., *isotropic error*), PCA becomes the aforementioned *probabilistic PCA* of the previous chapter, which, in the sense of the error structure, is more similar to factor analysis than it is to components analysis.

To understand and appreciate further why the above covariance identity holds, see Denis (2021a) or Johnson and Wichern (2007). The important point for our immediate purposes here is to appreciate that the covariance matrix can be “deconstructed” into the above as a result that follows from the very definition of the factor model. The equality in (12.2) implies that if we can estimate the contents of  $\Lambda$ , we can “account for” the covariance of observed variables, that is, the factor analytic model attempts to reconstruct the covariance matrix through its estimation of factors.

Is this idea new to factor analysis? Not at all! In virtually all modeling, we are trying to *reconstruct* something in terms of something else, some *translation* or *reorganization* of data or variables. This is what is behind most function statements (i.e., the left side and right side of equality signs in model equations) in statistical models as we are simply trying to account for something that is already observed, which in this case, is the covariance matrix. In factor analysis, we are trying to reproduce the observed covariance matrix based on the specification of the factor analytic model. In the case of linear regression, it was the observed vector of response values that we were trying to reproduce via our model equation. In the case of discriminant analysis, it was the binary or multi-coded response that we were, again, attempting to reproduce. In the case of neural networks, as we will see later, it is the continuous or categorical outcome, and so on with most other kinds of modeling. *Many models, whether in multivariate statistics or machine learning, attempt to reconstruct, in some sense, an outcome. How well they reconstruct that outcome is measured by how well the model “fits.” In many applications, the model serves as a “narrative” for how the data were generated just as physicists, for example, try to find a narrative for how the universe began or psychologists a narrative for how neurotic behavior arises. Those narratives are typically (though not necessarily always ideally) expressed via mathematical frameworks.*

### 12.3 The Problem with Factor Analysis

What would a model be if it did not come with some baggage? Perhaps more than any other statistical method, factor analysis comes with it a legacy of criticism and perhaps even more, a legacy of *misunderstanding*. The major criticism of factor analysis is that *factor loadings are not unique*. What this means is that we can change the factor loadings (through a *rotation* of the factor solution, as we will see later), and still reproduce the same covariance matrix. Both original factor loadings and transformed loadings (by an orthogonal matrix) can reproduce the same covariance matrix (Rencher & Christensen, 2012). Hence, the factor analyst is often left with trying to decide which are the “correct” loadings. The correct loadings will often be determined by which solution

makes the most *substantive* sense to the researcher, which can make factor analysis seem a bit too subjective for some and historically is often a reason for some of the disdain for factor analysis by critical audiences.

Full details about why this is the case can be found in Denis (2021a), but for now we cut right to the chase to the end result. The bottom line is that we can introduce an orthogonal matrix  $\mathbf{T}$  such that the following holds:

$$\begin{aligned}\mathbf{x} - \boldsymbol{\mu} &= \mathbf{A}\mathbf{f} + \boldsymbol{\varepsilon} \\ &= \mathbf{A}\mathbf{T}\mathbf{T}'\mathbf{f} + \boldsymbol{\varepsilon}\end{aligned}\tag{12.3}$$

That is, by simply including  $\mathbf{T}\mathbf{T}'$  into the factor model, we obtain an equivalent representation of  $\mathbf{x} - \boldsymbol{\mu}$ . The substantive implication of this is that the loadings we obtain in factor analysis are not *unique* and are subject to an orthogonal matrix. Substantively, such implies that if a researcher does not “like” the factor solution they obtained, they can modify it (via *rotation*) until they find a solution they do like! Had the solution been unique, then the factor analyst would be “stuck” with the loadings produced.

### 12.3.1 *The Problem Is the Users, Not the Method*

To conclude this thought on the chief difficulty with factor analysis, we should emphasize that first and foremost, there is nothing “wrong” with factor analysis. The lack of uniqueness of its loadings may be a limitation of the technique, but all statistical methods, when considered carefully and in much detail, can be said to have limitations. The key is to *recognize* those limitations, be acutely aware of them, and not conclude from the methodology more than is warranted. This is the real danger of not understanding a statistical method, that one will draw more conclusions from it than are warranted by the model. Hence, the chief problem as it concerns factor analysis is more to do with how it is *applied* and *interpreted*. That is, the problem is usually with the *users* of the method not interpreting factor analysis *ethically*.

The author has time and time again seen the factor analytic model *misapplied*, *misused*, *misinterpreted*, and substantive conclusions drawn from applications of the model that violate the model’s assumptions and are simply not epistemologically supported. But again, this can be said to be true of the use of most statistical methods at times. *Too often researchers believe statistics tell them something about data that it simply cannot*. With factor analysis, especially when applied to social data, the problem of misunderstanding is usually compounded. *It is often more important to understand and appreciate what a statistical model will not tell you than it is to learn what it can tell you about reality*.

### 12.3.2 *Factor Analysis Generalizes to Machine Learning*

Historically, factor analysis was slow to gain acceptance by the mathematics community. Some mathematical statisticians had disdain for the method due to some of its perceived technical pitfalls (such as loadings not being unique) but likely also due to the academic turf war it may have initiated with psychology where the method was originally conceived (Chatfield & Collins, 1980). Some books on multivariate analysis would even refuse to cover the technique at all and pretend as though it didn’t exist!

With the arrival and meteoric rise of machine learning however, along with the data science “revolution,” factor analysis has enjoyed a resurgence in virtually *all* fields, not just those in social science in which it has always been quite popular. Today, factor analysis and its many derivatives

and extensions is simply another method in the arsenal of the statistician or computer scientist. For example, *generative topographic mapping* (Bishop et al., 1998), a form of nonlinear factor analysis and alternative to self-organizing maps (Kohonen, 1982) is featured in machine learning and builds on the foundation of latent variable models, an idea originally popularized by factor analysis. Indeed, the concept of latent variables is a powerful one and though the *idea* of latent variables is not due to factor analysis per se (Mulaik, 1987), factor analysis is responsible for operationalizing the construct and putting it on a technical foundation. Factor analysis has also helped give rise to the field of psychometrics. For instance, *item response theory*, also known as *latent distribution theory*, *item characteristic curve* (ICC), and *item characteristic function* (ICF), or *latent trait analysis* (Mair, 2018) all build on the basic factor model. A good overview and summary of these models are provided by Mahmud (2017).

Hence, we argue that to simply conclude that factor analysis is problematic and should be avoided is a well-outdated viewpoint, especially in the context of the rise of computer science and machine learning where it has found tremendous application. What is not an outdated viewpoint is that unless one is willing to study, understand and appreciate some of the more salient theoretical underpinnings and caveats of the method, one stands a much greater chance of misunderstanding and misusing factor analysis and concluding from the method much more than is epistemologically warranted.

## 12.4 Factor Estimation

As with virtually all statistical methods, we require a way to estimate parameters of the model. Though there are numerous estimation methods in EFA, classical estimation methods include *principal factor* (or *principal axis factoring*) as well as *maximum likelihood*, which we now (very) briefly survey.

### 12.4.1 Principal Factor (Principal Axis Factoring)

To understand the principal factor method of estimation, it is first necessary to revisit PCA and define what is meant by a *communality*. Given that we are working with the correlation matrix, the main diagonal in a PCA will contain *unit variances* as a result of each variable initially contributing one “unit” of variance to the PCA. In the *principal factor* approach, also known as *principal axis factoring*, the residual matrix is minimized and features replacing the initial communalities along the main diagonal in the correlation matrix with weights that are derived from the given variable being regressed on remaining variables subjected to the factor analysis. That is, each communality weight is given by

$$\hat{h}_i^2 = R_i^2 = 1 - \frac{1}{r^{ii}} \quad (12.4)$$

where  $\hat{h}_i^2$  is the estimated communality for variable  $i$ ,  $R_i^2$  is the coefficient of determination for regressing variable  $i$  on all other variables being subjected to the given factor analysis, and  $r^{ii}$  is the  $i^{\text{th}}$  diagonal element of the inverse of the correlation matrix  $R$  (Rencher & Christensen, 2012). If the covariance instead of the correlation matrix is analyzed, then (12.4) becomes

$$\hat{h}_i^2 = ss_{ii} R_i^2 \quad (12.5)$$

where  $ss_{ii}$  is the  $i^{\text{th}}$  diagonal element of the covariance matrix,  $S$ .

### 12.4.2 Maximum Likelihood

The most common approach for estimating factors is, perhaps not surprisingly, that of maximum likelihood. It is easiest to carry out *maximum likelihood* (ML) estimation via the Expectation-Maximization algorithm (Choi et al., 2010; Rubin & Thayer, 1982). As was the case for the orthogonal factor model, the ML estimator is also invariant under rotation, and hence any rotation will generate the same ML estimates (Choi et al., 2010). As we will see later, this is not unconditionally true for all factor analysis models. For instance, imposing an  $\ell_1$  penalty on the factor model, yielding sparse “lasso” factor analysis, is not rotationally invariant (Choi et al., 2010).

To obtain the maximum likelihood estimator for factor analysis, we first need to define the log-likelihood function in a multivariate context:

$$\ln L(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -np \ln \sqrt{2\pi} - \frac{1}{2} n \ln |\boldsymbol{\Sigma}| - \frac{1}{2} (n-1) \text{tr}(\boldsymbol{\Sigma}^{-1} \mathbf{S}) - \frac{1}{2} n (\bar{\mathbf{y}} - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}) \quad (12.6)$$

The maximum likelihood estimator for  $\boldsymbol{\mu}$  is  $\bar{\mathbf{y}}$  and the assumed covariance structure for the population covariance matrix  $\boldsymbol{\Sigma}$  is  $\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \boldsymbol{\Psi}$ , from which we can rewrite the log-likelihood function as

$$\ln L(\boldsymbol{\Lambda}, \boldsymbol{\Psi}) = -np \ln \sqrt{2\pi} - \frac{1}{2} n \ln |\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \boldsymbol{\Psi}| - \frac{1}{2} (n-1) \text{tr}[(\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \boldsymbol{\Psi})^{-1} \mathbf{S}] \quad (12.7)$$

It can be shown that maximizing the log-likelihood function is equivalent to minimizing the following fit function,

$$F_{ML} = \ln |\boldsymbol{\Sigma}(\boldsymbol{\theta})| + \text{tr}(\mathbf{S} \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta})) - \ln |\mathbf{S}| - p \quad (12.8)$$

and upon substituting the orthogonal factor model, we obtain

$$\begin{aligned} F_{ML} &= \ln |\boldsymbol{\Sigma}(\boldsymbol{\theta})| + \text{tr}(\mathbf{S} \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta})) - \ln |\mathbf{S}| - p \\ &= \ln |\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \boldsymbol{\Psi}| + \text{tr}(\mathbf{S} (\boldsymbol{\Lambda} \boldsymbol{\Lambda}' + \boldsymbol{\Psi})^{-1}) - \ln |\mathbf{S}| - p \end{aligned} \quad (12.9)$$

## 12.5 Factor Rotation

Recall we had said earlier that factor loadings are not unique, or, put another way, they are unique *but only up to an orthogonal rotation*. Recall that a square matrix  $\mathbf{T}$  is orthogonal if  $\mathbf{T} \mathbf{T}' = \mathbf{T}' \mathbf{T} = \mathbf{I}$ . Given this quality of the factor model, it stands that we can reproduce the same covariance matrix  $\boldsymbol{\Sigma}$  but with alternative loadings. This translates into rotating the factor solution virtually at will yet still reproducing the same covariance matrix,  $\boldsymbol{\Sigma}$ . The image in Figure 12.1 depicts the idea of a factor rotation where we see the original axes  $x_1$  and  $x_2$  rotated to new axes  $y_1$  and  $y_2$ , respectively:

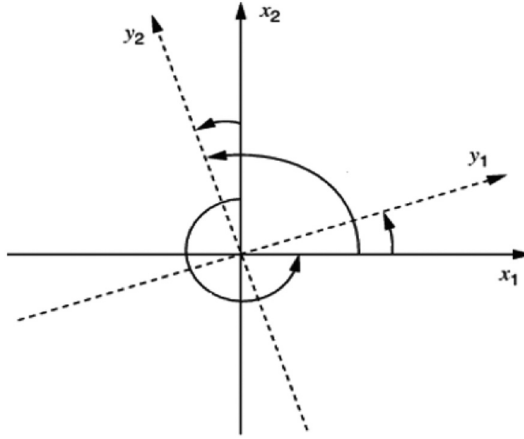


Figure 12.1 Concept of a factor rotation. The new axes ( $y_1$  and  $y_2$ ) are rotated from the old ( $x_1$  and  $x_2$ ).

A *clockwise* rotation features multiplying the factor loadings in  $\Lambda$  by the matrix  $\mathbf{O}_1$ ,

$$\mathbf{O}_1 = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix}$$

whereas a *counterclockwise* rotation of  $\Lambda$  features multiplication by the matrix  $\mathbf{O}_2$ ,

$$\mathbf{O}_2 = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}$$

The new rotated loadings we now denote by  $\Lambda_R$  and it can be shown that these new rotated loadings *provide the same fit to the covariance matrix as the unrotated loadings*. Recall our original deconstruction of the covariance matrix  $\Sigma$  presented earlier:

$$\Sigma = \Lambda \Lambda' + \Psi$$

It can be shown that with the new loadings  $\Lambda_R$ , the same covariance matrix can be reproduced:

$$\Sigma = \Lambda_R \Lambda_R' + \Psi \quad (12.10)$$

It is for this reason that loadings in the orthogonal factor model are not unique. We can rotate the loadings quite freely and still generate the same covariance (or correlation) matrix. There are numerous ways of rotating factors in factor analysis. In what follows we survey a couple of the more popular methods in use.

### 12.5.1 Varimax Rotation

The varimax rotation is given by

$$V = \sum_{i=1}^p (l_{ij}^2 - \bar{l}_j^2)^2 \quad (12.11)$$

where  $l_{ij}^2$  are the squared loadings  $i$  to  $p$  for a given factor, and  $\bar{l}_j^2$  is the average squared loading across the  $p$  variables (i.e., across the *observed variables*, not factors). The name *varimax* derives from the fact that it maximizes these sum of squared loadings for each factor. How is such a maximum achieved? It attains a maximum by driving larger weights to be even larger and smaller weights to be even smaller, creating *heterogeneity* in the estimated weights in a manner that maximizes their squared sum. Varimax rotation has the benefit of clarifying factor structure since the larger weights will, hopefully, be more indicative of what the factor is “made up of” (in some sense) compared to the smaller weights that poorly indicate the factor structure. In his original use of factor analysis, Thurstone called this idea that of establishing *simple structure* (Thurstone, 1954).

### 12.5.2 Quartimax

The quartimax rotation is given by

$$Q = \sum_{j=1}^m (l_{ij}^2 - \bar{l}_j^2)^2 \quad (12.12)$$

where now the sum of squared loadings,  $l_{ij}^2$ , is maximized *across* factors  $j = 1$  through  $m$ , having the effect of maximizing the variance due to observed variables rather than the factor variance as was the case for varimax. That is, maximizing *across* factors instead of *within* factors as varimax does has the effect of maximizing *variable* variance in place of the *factor* variance maximized by varimax.

Regardless of which of the above rotations we perform, the same correlation (or covariance) matrix is reproduced. To reiterate, this point is key to understanding the indeterminacy problem of factor analysis. One could, in theory, rotate the solution until one finds the solution they “prefer” without any consequence on the structure or makeup of the covariance matrix. Clearly, this will lead to the analyst’s subjective preference for a given solution, one that naturally accords with what the analyst was “hoping” to see.

## 12.6 Bartlett’s Test of Sphericity

A factor analysis, not surprisingly, requires there to be correlations greater than 0 in  $\mathbf{R}$ , the correlation matrix. Prior to conducting a factor analysis then, one might wonder whether performing the EFA on the correlation matrix is worthwhile. That is, we may question whether there is *sufficient* correlation in the matrix on which to conduct the factor analysis. Of course, nothing prevents us from trying an EFA on a matrix with even minimal correlation. However, tests exist to evaluate the feasibility of doing so prior to doing the factor analysis and might also help explain why the ensuing factor analysis did not quite work out as the investigator had originally hoped.

*Bartlett’s test of sphericity* is one such test that seeks to evaluate the “appropriateness” of the data as it concerns factor analysis and is given by

$$\chi^2 = - \left[ (n-1) - \frac{(2p+5)}{6} \right] \ln |\mathbf{R}| \quad (12.13)$$

where  $p$  is the number of variables,  $n$  is the number of observations, and  $\ln |\mathbf{R}|$  is the natural logarithm of the determinant of the correlation matrix. For any given problem,

$$-\left[(n-1) - \frac{(2p+5)}{6}\right]$$

will be a constant, and hence what determines the size of  $\chi^2$  will be a function of  $\mathbf{R}$ , more specifically the determinant of  $\mathbf{R}$ ,  $|\mathbf{R}|$ , where the latter can be considered a measure of *generalized variance* (Johnson & Wichern, 2007). Rejecting the null in this case suggests the matrix under analysis is not an identity matrix, which implies there exists sufficient correlation in  $\mathbf{R}$  to carry on with the analysis. A failure to reject the null here does not “prove” the matrix to be an identity matrix, but it may cast doubt on whether performing the factor analysis is worthwhile. As with all significance tests, however, since Bartlett’s will be influenced by sample size, researchers are sometimes encouraged to perform the factor analysis regardless of what is reported by Bartlett’s, especially if sample size is fairly large. On a practical level, performing the factor analysis can’t hurt either way, and at worst, the investigator may simply not find any meaningful factor structure from the analysis.

### 12.6.1 Factor Analysis in Python

We can perform factor analysis in Python using `FactorAnalyzer()`:

```
import pandas as pd
import numpy as np
```

Let’s create some data:

```
data = np.array([[0, 5.90], [0.90, 5.40], [1.80, 4.40], [2.60,
4.60], [3.30, 3.50], [4.40, 3.70], [5.20, 2.80], [6.10, 2.80],
[6.50, 2.40], [7.40, 1.50]])
data
Out[108]:
array([[0. , 5.9],
       [0.9, 5.4],
       [1.8, 4.4],
       [2.6, 4.6],
       [3.3, 3.5],
       [4.4, 3.7],
       [5.2, 2.8],
       [6.1, 2.8],
       [6.5, 2.4],
       [7.4, 1.5]])
```

We will import `factor_analyzer` as well as Bartlett’s test of sphericity and the `chi_square` value for computing Bartlett’s:

```
import factor_analyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
```

Recall that Bartlett’s test evaluates the null hypothesis that the correlation matrix is an identity matrix. As an example, for a three-variable problem,

$$H_0: \mathbf{R} = \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The alternative hypothesis is that  $H_1: \mathbf{R} \neq \mathbf{I}$ . When we run Bartlett’s test of sphericity, we obtain:

```
chi_square_value, p_value=calculate_bartlett_sphericity(data)
chi_square_value, p_value
Out[35]: (23.012894934627763, 1.609184226770749e-06)
```

The  $p$ -value is very small (1.609...-06), suggesting that we can reject the null hypothesis that the correlation matrix is an identity matrix. We infer that there *is* a degree of correlation in the matrix, probably enough to try a factor analysis.

We also wish to import the KMO test, which stands for the *Kaiser-Meyer-Olkin measure of sampling adequacy*, a measure used for evaluating the utility for conducting a factor analysis in the sense of providing an estimate of the proportion of variance in variables that might be “due” (in some sense) to underlying factors (Denis, 2021b):

```
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all, kmo_model=calculate_kmo(data)
kmo_model
Out[36]: 0.5
```

Values of 0.70 or higher are generally preferred, however values as small as 0.5 are sometimes regarded as sufficient. Either way, a result on the KMO test should not discourage you from running a factor analysis even if the result is somewhat less than 0.5; the worst that can happen here is that your factor analysis does not “pan out” as you had hoped.

We now conduct the factor analysis using the ‘minres’ approach, which stands for *minimum residual* (see Comrey (1962) for details on the approach). The number of factors we will request is equal to 1 (`n_factors=1`):

```
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(rotation=None, method = 'minres', n_factors=1)
fa.fit(data)
Out[112]: FactorAnalyzer(n_factors=1, rotation=None, rotation_kwargs={})
fa.loadings_
Out[113]:
array([[ -0.98816761],
       [ 0.98816761]])
fa.get_communalities()
Out[114]: array([0.97647522, 0.97647522])
```

We see that the loadings from the factor analysis are  $-0.988$  and  $0.988$ , with each communality equal to 0.97. Factor analysis can also be performed using Python’s *sklearn*.

### 12.6.2 Factor Analysis in R

Factor analysis is best performed in R instead of Python as more packages and options are devoted to the EFA model than in Python. We adapt a simple example of factor analysis on the *Holzinger.9* data, which consists of numerous intelligence measurements on children such as *visual perception*, *paragraph comprehension*, *addition*, etc. We list some of the data below:



```
> library(psych)
> Holzinger.9
```

|           | vis_perc | cubes    | lozenges | par_comp | sen_comp | wordmean | addition |
|-----------|----------|----------|----------|----------|----------|----------|----------|
| vis_perc  | 1.00000  | 0.325800 | 0.448640 | 0.34163  | 0.30910  | 0.31713  | 0.104190 |
| cubes     | 0.32580  | 1.000000 | 0.417010 | 0.22800  | 0.15948  | 0.19465  | 0.066362 |
| lozenges  | 0.44864  | 0.417010 | 1.000000 | 0.32795  | 0.28685  | 0.34727  | 0.074638 |
| par_comp  | 0.34163  | 0.228000 | 0.327950 | 1.00000  | 0.71861  | 0.71447  | 0.208850 |
| sen_comp  | 0.30910  | 0.159480 | 0.286850 | 0.71861  | 1.00000  | 0.68528  | 0.253860 |
| wordmean  | 0.31713  | 0.194650 | 0.347270 | 0.71447  | 0.68528  | 1.00000  | 0.178660 |
| addition  | 0.10419  | 0.066362 | 0.074638 | 0.20885  | 0.25386  | 0.17866  | 1.000000 |
| count_dot | 0.30760  | 0.167960 | 0.238570 | 0.10381  | 0.19784  | 0.12114  | 0.587060 |
| s_c_caps  | 0.48683  | 0.247860 | 0.372580 | 0.31444  | 0.35560  | 0.27177  | 0.418310 |

|           | count_dot | s_c_caps |
|-----------|-----------|----------|
| vis_perc  | 0.30760   | 0.48683  |
| cubes     | 0.16796   | 0.24786  |
| lozenges  | 0.23857   | 0.37258  |
| par_comp  | 0.10381   | 0.31444  |
| sen_comp  | 0.19784   | 0.35560  |
| wordmean  | 0.12114   | 0.27177  |
| addition  | 0.58706   | 0.41831  |
| count_dot | 1.00000   | 0.52835  |
| s_c_caps  | 0.52835   | 1.00000  |

For this factor analysis, we use the *psych* package (Revelle, 2024) and implement a *varimax* rotation:

```
> fa <- factanal(covmat = Holzinger.9, factors=2, n.obs = 145, rotation = "varimax")
> fa

Call:
factanal(factors = 2, covmat = Holzinger.9, n.obs = 145, rotation = "varimax")

Uniquenesses:
vis_perc  cubes  lozenges  par_comp  sen_comp  wordmean  addition  count_dot
  0.733   0.899    0.781    0.237    0.327    0.323    0.595    0.253
s_c_caps
  0.514
```

The uniqueness values represent the errors, that variance not accounted for by the loadings, that is, the “left over” from the communalities for each observed variable. For example, for *vis\_perc* below, the sum of squared loadings is computed as

$$(0.354)^2 + (0.376)^2 = 0.125316 + 0.141376 = 0.266692$$

The uniqueness value for *vis\_perc* is thus equal to  $1 - 0.266692 = 0.733$ .

Loadings:

|           | Factor1 | Factor2 |
|-----------|---------|---------|
| vis_perc  | 0.354   | 0.376   |
| cubes     | 0.232   | 0.219   |
| lozenges  | 0.364   | 0.293   |
| par_comp  | 0.866   | 0.112   |
| sen_comp  | 0.794   | 0.205   |
| wordmean  | 0.815   | 0.114   |
| addition  | 0.126   | 0.624   |
| count_dot | 0.864   |         |
| s_c_caps  | 0.288   | 0.635   |

|                | Factor1 | Factor2 |
|----------------|---------|---------|
| SS loadings    | 2.455   | 1.882   |
| Proportion Var | 0.273   | 0.209   |
| Cumulative Var | 0.273   | 0.482   |

Test of the hypothesis that two factors are sufficient.  
 The chi-square statistic is 61.7 on 19 degrees of freedom.  
 The  $p$ -value is 2.08e-06

We can produce a *parallel analysis* scree plot (Horn, 1965), showing the EFA (named FA in the plot) along with simulated data (Figure 12.2). Parallel analysis generates a simulation of eigenvalues as a function of component number using a Monte Carlo technique which is parallel to the data set. The procedure will produce a function in the plot next to the scree plot based on the factor analysis. Where the line crosses the scree plot is approximately where the recommended number of factors to keep lies given they are roughly at or just above the line, at least according to the parallel analysis:

```
> fa.parallel(Holzinger.9)
```

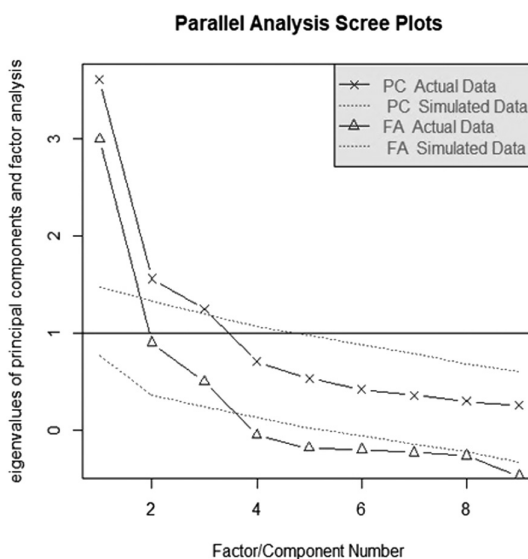


Figure 12.2 Horn's parallel analysis plot for the factor analysis performed on the Holzinger data.

We see in Figure 12.2 that Horn’s function crosses the FA scree function between approximately 3–4 factors, hence it would be reasonable, according to Horn, to keep probably three factors or so in this case. Our solution earlier tested two factors, which may be sufficient so long as we can “make sense” of the factors and associated loadings. Cokluk and Kocak (2016) provide a good overview of Horn’s parallel analysis. The package *hornpa* (Huang, 2022) can also be used for parallel analysis.

## 12.7 Independent Factor Analysis

Recall in our more general discussion of principal components earlier in the book, we surveyed the method of independent components analysis, which required components to not merely be uncorrelated but rather *independent*. ICA was featured as one method in the wider blind-source separation and manifold learning problem. Most applications of ICA, however, did not feature an error term and are regarded as *noiseless ICA*. As Attias (1999, pp. 2–3) notes, however, not including a noise term limits the ability of ICA to recover components:

As the noise level increases, the performance of ICA algorithms deteriorates and the separation quality decreases, as manifested by cross-talk and noisy outputs. More importantly, many situations have a relatively small number of sensors but many sources, and one would like to lump the low-intensity sources together and regard them as effective noise, while the separation sources focuses on the high-intensity ones. There is no way to accomplish this using ICA methods ... Our IFA [independent factor analysis] algorithm reduces to ordinary FA when the model sources become Gaussian, and performs principal component analysis (PCA) when used in the zero-noise limit ... IFA therefore generalizes and unifies ordinary FA, PCA and ICA and provides a new method for modeling multi-variable data in terms of a small set of independent hidden variables ... IFA amounts to fitting those data to a mixture model of co-adaptive Gaussians.

Hence, the classic factor analysis model, principal components, and independent principal components can all be conceived in a sense as special cases of the wider independent factor analysis model. The IFA model will look as it did in the classic version of the EFA model,

$$\mathbf{x} = \boldsymbol{\mu} + \boldsymbol{\Lambda}\mathbf{f} + \boldsymbol{\varepsilon} \quad (12.14)$$

but with the important caveat that factors will now be constrained to be *independent*, not only *uncorrelated* as in the common factor analysis model. The independent latent variables have flexible distributions modeled by mixtures of Gaussians (Bishop, 2006) and hence while common exploratory factor analysis can be thought of as a low-rank version assuming a Gaussian distribution, independent factor analysis distinguishes itself from this as it features non-Gaussian distributions with Gaussian densities as essentially a “special case” (Izenman, 2008).

The loading matrix  $\boldsymbol{\Lambda}$  in IFA is regarded as the *mixing* matrix and as was true of ICA, is responsible for blending in of the various sources. The EM algorithm is typically used to maximize the log-likelihood of the observed data with respect to the parameters of the generative model that account for the data (Attias, 1999). As is the case with common factor analysis, in independent factor analysis the number of latent variables and observed variables can differ (Bishop, 2006).

The independent factor analysis model has not seen much development as a unique method since its inception nor does it appear extensively in the literature. This may be due to the ICA model dominating applications (Izenman, 2008).

## 12.8 Nonlinear Factor Analysis and Autoencoders

We have emphasized that the classic factor analysis model is linear in the parameters. One way to relax this assumption is to allow the mapping of variables to unobserved variables to be a *nonlinear model*. We remind ourselves at this point that classic PCA can be conceptualized as an *autoencoder*, where recall an autoencoder is a kind of *transformer* that encodes input into output but does not necessarily reduce the dimensionality of the input. Factor analysis can also be conceived as such, and hence conceptualizing nonlinear factor analysis is best done via an autoencoder, more specifically what is called a *variational encoder*, which is best thought of as a probabilistic version of a deterministic autoencoder (Murphy, 2022). Factor analysis is a variational encoder due to its inclusion of an error term,  $\epsilon$ . It is also a *generative model*, which implies that we can create new data (e.g., factor scores) from it, whereas the traditional classic autoencoder is deterministic (Murphy, 2022).

### 12.8.1 Unpacking the Autoencoder

Recall that an autoencoder features two components, the *encoder* is  $x \rightarrow z$  while the *decoder* is  $z \rightarrow x$ . In the current context, observed variables  $x$  are encoded into one or more latent variables, while latent variables can be decoded back into observed variables. Unpacked, encoding  $x$  into  $z$  with a weighting matrix  $\mathbf{W}$  corresponds to  $z = \mathbf{W}_1^T x$ , while decoding  $z$  corresponds to  $\hat{x} = \mathbf{W}_2^T z$ , where the distance  $x - \hat{x}$  is minimized (Murphy, 2022). In a classical PCA, the encoding and decoding stages will be linear maps and  $\mathbf{W}$  will be constrained to be orthogonal. In a nonlinear factor analysis or a (nonlinear) neural network, the encoding and decoding will be nonlinear and depending on whether we wish factors to be uncorrelated or not we will place constraints on  $\mathbf{W}$ . Eigenvectors of the covariance matrix in  $\mathbf{W}$  are chosen such that the above reconstruction error is minimized.

The key to nonlinear factor analysis then is that the mapping from  $x \rightarrow z$  contains nonlinear functions, which may include such things as *interaction terms* or *polynomials*, or virtually any other nonlinear function. Note that this may sound somewhat like multiple regression, but recall that in multiple regression, there is no real “encoding,” unless we think of predictors in  $x$  being “encoded” into a fitted vector,  $\hat{y}$ , but such is not traditionally how autoencoders are usually conceived or conceptualized.

### 12.8.2 Factor Analysis as a Neural Network

Because an autoencoder can be conceived as a special type of neural network, the nonlinear maps from  $x \rightarrow z$  and  $z \rightarrow x$  applied to factor analysis can also be thought of as a *neural network* in which the mapping is some nonlinear *activation function*. Hence, we see once more the generality of specific approaches such as factor analysis as special cases of a wider neural network framework. Not surprisingly, as is the case with other nonlinear methods surveyed in this book, imposing a nonlinear encoder can detect patterns and shapes in data that a classic linear factor analytic model or principal components cannot (Japkowicz & Gluck, 2000). According to Japkowicz et al. (2000, p. 1):

While linear autoassociators emulate PCA and thus exhibit a flat or unimodal reconstruction error surface, autoassociators with nonlinearities in their hidden layer learn domains

by building error reconstruction surfaces that, depending on the task, contain multiple local valleys.

Nonlinear factor analysis can also be conceived as a *deep latent Gaussian model* (Rezende et al., 2014). The origins of nonlinear factor analysis go back to Bartlett (1953), though at the time it could not be conceived as an autoencoder or neural network, since these conceptualizations are relatively recent. For a thorough review of nonlinear factor analysis, see Amemiya and Yalcin (2001).

## 12.9 Probabilistic “Sensible” PCA (Again)

We have already surveyed *probabilistic PCA* in the context of PCA. We now survey it in the context of the factor analytic model. Probabilistic PCA, or *sensible PCA* as it is sometimes known (Roweis, 1998), contains noise along the main diagonal of the correlation matrix whereas classical PCA on the correlation matrix contains values of unity along the main diagonal. As explained by Roweis (1997, p. 5):

If we require  $\mathbf{R}$  [correlation matrix] to be a multiple  $\epsilon \mathbf{I}$  of the identity matrix (in other words the covariance ellipsoid of  $\mathbf{v}$  is spherical) but do not take the limit as  $\epsilon \rightarrow 0$  then we have a model which I shall call sensible principal components analysis or SPCA ... the scalar value  $\epsilon$  on the diagonal of  $\mathbf{R}$  the global noise level.

The key to the above is not taking  $\epsilon \rightarrow 0$  in the limit as one would in a classical orthogonal PCA model. Allowing  $\epsilon \neq 0$  permits at least *some* error along the main diagonal of  $\Psi$  yielding the following probabilistic PCA reconstruction of  $\Sigma$  which looks a lot like factor analysis:

$$\Sigma = \Lambda \Lambda' + \Psi \quad (12.15)$$

This is why PCA is sometimes cast as the limiting case of a linear-Gaussian model, equivalent to the classical factor analysis model as the error or “noise” becomes extremely small and in the limit goes to zero.

So, what is the difference between probabilistic PCA and factor analysis then? It would seem they are the same since they both contain error terms. They are slightly different, however. The key difference lies in the restrictions placed on the error term. In the factor analysis model, recall that  $\Psi$  was constrained to have  $\sigma^2 \mathbf{I}$  along its main diagonal. That is,  $\sigma^2 \mathbf{I}$  translates to values of noise  $\epsilon$  along the main diagonal of  $\mathbf{R}$ . However,  $\sigma^2$  *did not need to be constant (i.e., the same number) along the main diagonal*. In probabilistic PCA, however,  $\Psi$  is constrained to have  $\sigma^2 \mathbf{I}$  along its main diagonal and it is required for  $\Psi$  to be *isotropic* or *spherical*, which means it must be *proportional* to the identity matrix (Bishop, 2006, p. 584). Since the identity matrix has values of one along the main diagonal, this implies that  $\Psi$  *must have errors that are equal to one another*. This is why probabilistic PCA is considered a special case of factor analysis, where factor analysis allows, as Lawrence (2005, p. 1784) puts it, a “richer error term” which simply means in this context that the error term need not be *constant*. That is, the covariance matrix need not be spherical in classic factor analysis. In probabilistic PCA, it is required to be so.

## 12.10 Mixtures of Factor Analysis: Modeling Local Linearity

Instead of assuming a linear mapping to unobservable latent variables, as is true of other methods, we can relax this assumption and model things more *locally*. That is, we can assume that locally the model is linear, and then combine all these local model segments to approximate a curved

manifold. Of course, this concept is not new, we have come across this idea of *local approximation* many times throughout this book (such as in locally linear embedding). In the current context, when one sums together local linear approximations, one gets a better picture of the overall curved manifold, somewhat analogous in concept to the idea of fitting many tangent lines to get the “fit” of a curve in calculus. In essence, a mixture of factor analyzers devotes a factor analysis to each of individual groups or clusters. It is perhaps best described by Ghahramani and Hinton (1997, p. 1):

we present an EM learning algorithm for a method which combines one of the basic forms of dimensionality reduction – factor analysis – with a basic method for clustering – the Gaussian mixture model. What results is a statistical method which concurrently performs clustering and, within each cluster, local dimensionality reduction.

First, we recall the factor analysis model, omitting the mean,  $\boldsymbol{\mu}$ ,

$$\mathbf{x} = \Lambda \mathbf{f} + \boldsymbol{\varepsilon}$$

For the mixture model, where now  $\mathbf{f} = \mathbf{z}$ , the generative model is the following,

$$P(\mathbf{x}) = \sum_{j=1}^m \int P(\mathbf{x} | \mathbf{z}, w_j) P(\mathbf{z} | w_j) P(w_j) d\mathbf{z} \quad (12.16)$$

and where  $w_j$  are the different mixing distributions such that

$$p(\mathbf{z} | w_j) = p(\mathbf{z}) = N(0, I)$$

which reads that the conditional probability of  $\mathbf{z}$  given  $w_j$  is equal to the unconditional probability of  $\mathbf{z}$ , of which both are normally distributed. Following Ghahramani and Hinton (1997, p. 4), each factor for each mixing distribution will also have its own mean,

$$p(\mathbf{x} | \mathbf{z}, w_j) = N(\boldsymbol{\mu}_j + \Lambda_j \mathbf{z}, \boldsymbol{\Psi}) \quad (12.17)$$

which is unlike in traditional factor analysis, where the mean was constant across all factors (i.e.,  $\mathbf{x} = \boldsymbol{\mu} + \Lambda \mathbf{f} + \boldsymbol{\varepsilon}$ ).

The mixture of factor analyzers is estimated using the EM algorithm. For details, see Ghahramani and Hinton (1997). If the error structure happened to follow that of the probabilistic PCA model just surveyed, then the model becomes a *mixture of probabilistic PCA models*. Indeed, probabilistic PCA is a special case of the above model when, as we saw, the error matrix is spherical (Murphy, 2022). The covariance matrix for the mixture model will have  $mkp + p$  parameters whereas the classic nonmixture factor analysis had  $mp(p + 1)/2$ , where  $m$  is the number of factor analysis models to use in the mixture model,  $k$  is the number of factors to extract in each of those factor analyses, and  $p$  is the number of variables in  $\mathbf{x}$ .

### 12.11 Item Factor Analysis

Item factor analysis, or IFA, is the factor analysis model adapted to analyzing multivariate *categorical* data, which proves especially useful for modeling item-level response data on surveys and questionnaires. These data are often *binary* or *polytomous* or otherwise *categorical* in nature and thus the classic maximum-likelihood factor analysis model, which assumes continuous variables

can cause issues (Everitt & Dunn, 2001). Indeed, performing such a traditional factor analysis on data that is not continuous can result in very disparate findings when compared to performing it with an alternative estimation method. See Everitt and Dunn (2001, p. 286) for a comparison of output.

Muthén and Kaplan (1985) compared estimators for factor analysis of non-normal skewed variables, which included binary and categorical variables, and which included *maximum likelihood*, *generalized least-squares* (GLS), and *asymptotically distribution-free* (ADF) *generalized least-squares*, as well as a categorical variable estimator. They found that ML and GLS standard errors were typically biased downward, but that for ADF no bias was found and the CVM (*categorical variable estimator*) worked well for skewed variables that had been dichotomized. For further details, see Muthén and Kaplan (1985).

Urban and Bauer (2021) proposed a deep artificial neural network for exploratory item factor analysis, who also showed that other factor analytic models such as nonlinear factor analysis may be subsumed under their novel algorithm. Software-wise, *Mplus* (Muthén & Muthén, 2017) has especially good capabilities for fitting factor analysis models featuring binary or categorical indicators. Item factor analysis can also be fit using the *OpenMx* package in R (Boker et al., 2024).

## 12.12 Sparse Factor Analysis

Recall that a *sparse* model is one in which there is a purposeful effort to simplify the model by driving coefficients *to* or *toward* values of 0 in the same spirit as estimators such as ridge and lasso regression surveyed earlier in the book. In one sense, we have already surveyed a sparse-like factor analysis model when we introduced the *varimax rotation*, whose objective, recall, was to drive large coefficients to be larger and small coefficients to be smaller as to approximate *simple structure* for each factor. The sparse factor analysis likewise aims to increase the sparsity of the factor model by introducing penalization or regularization methods that drive coefficients to or toward 0, as to simplify and clarify the factor structure.

Drton et al. (2024) detail algebraically sparse factor analysis models as models that set or hypothesize some or many of the coefficients (i.e., *factor loadings*) equal to 0. Other approaches do not necessarily set weights equal to zero, but instead “encourage” (Murphy, 2022) them to equal 0 through regularization, such as an  $\ell_1$  penalty (Choi et al., 2010). Penalties of course have a Bayesian flavor in a sense because we are influencing the size of weights *a priori* and thus having a corresponding influence on prior probabilities. Frühwirth-Schnatter et al. (2024) detail *sparse Bayesian factor analysis* for the case where the number of factors is unknown with what are known as *spike-and-slab priors* on the factor loading matrix. The use of such priors drives some factor loadings to 0, which allows the model to be identified. Setting weights to 0 to help identify a model is also a common analytical choice in the structural equation models of the following chapter.

## 12.13 Chapter Review Exercises

1. Compare the *exploratory factor analysis* model to the traditional linear regression model. Note as many similarities and differences as you can.
2. Unpack the following factor analysis assumptions to demonstrate you know what they mean,  $E(\mathbf{ff}') = \mathbf{0}$  and  $E(\mathbf{ff}') = \mathbf{I}$ .
3. What does it mean to say factor analysis *implies a structure to the covariance matrix*?
4. What is the chief problem associated with exploratory factor analysis that invites its critics? Is this problem insurmountable? Can it be resolved?
5. How are *communalities* estimated in the *principal factor estimation* method of factor analysis?

6. Compare and contrast *varimax rotation* to that of *quartimax*. How are they similar? Different?
7. Show that the  $p$ -value resulting from *Bartlett's test of sphericity* can be largely a function of sample size.
8. Perform an exploratory factor analysis in Python on contrived data on five variables.
9. Duplicate the factor analysis in #8 in R and obtain a *parallel analysis plot*. Interpret the plot. How many factors are suggested by the plot?
10. Compare and contrast *exploratory factor analysis* to *independent factor analysis*.
11. How can factor analysis be conceived as an *autoencoder*, specifically, a *variational autoencoder*?
12. Compare and contrast a *mixture model* of factor analysis to classical exploratory factor analysis.



# 13 Confirmatory Factor Analysis, Path Analysis, and Structural Equation Modeling

In the previous chapter, we surveyed the exploratory factor analysis (EFA) model and learned that in the classic factor analysis model, observed variables were being hypothesized as linear functions of unobserved latent variables. We also surveyed some of the very many derivatives and extensions of the classic EFA model, which included nonlinear factor analysis and other possibilities, which ranged from models emanating from statistics and social science to more modern advances in machine learning.

In the current chapter, we develop the full structural equation model, which can be said to consist of elements common to both factor analysis and regression. First, we consider the confirmatory version of factor analysis, appropriately called *confirmatory factor analysis* in which the researcher has a specific factor pattern or hypothesis they would like to test instead of conducting a purely exploratory search as in EFA. What a confirmatory test reduces to, on a technical level, is a more deliberate *fixing and freeing of model parameters* than in its exploratory counterpart. However, as we will see, distinctions between what makes a model exploratory vs. confirmatory can be quite blurry. We then introduce the structural equation model, which can be conceptualized as a series of (typically) linear equations, however nonlinear structural equation modeling is also a possibility and we briefly survey those as well.

The origins of structural equation modeling are twofold. The measurement model of SEM can be said to have begun with Charles Spearman (1904), while the structural portion of SEM has its roots primarily with Sewall Wright (1918) who developed the method of *path analysis*. Path analysis features networks among observed variables and hence is largely seen as a precursor, historically, to the arrival of structural equation models. Combining the measurement model with that of the structural components of path analysis gives rise to the full structural equation model. Jöreskog (1970, 1973) is usually credited with the first true SEM model (Tarka, 2018) although researchers, especially in sociometrics, were running structural equation modeling in the 1950s and 60s. It was Jöreskog, however, who is primarily responsible for pioneering cutting-edge estimation methods for SEM.

## 13.1 What Makes a Model “Exploratory” vs. “Confirmatory”?

Exploratory factor analysis models are named as such because the user does not typically have any idea in advance regarding how many factors may result from the analysis. This is the *exploratory* nature of the model; the user is diving into the data without any preconceived *theory* as to the number of factors that may result. At first thought, this may seem like an unrealistic proposition, since researchers usually have at least *some* idea of what may come out of their factor analyses before conducting them. That is, the Bayesian prior is typically not *uniform*, even in exploratory analyses.

The way to improve on the above exploratory idea of a factor analysis model is to have a *pre-conceived structure* of what to expect from the factor analysis. In this case, the researcher has a pre-established theory in mind and wishes to evaluate that specific hypothesized structure. *Confirmatory factor analysis* in this case is a viable option, where the model allows the researcher to virtually control and specify almost every parameter of the model. That is, the user can *fix*, *free*, or even *equate* estimated parameters with other model parameters and obtain measures of model fit assessing how well or how poorly her model stood up to the data.

A step above the confirmatory factor analysis (CFA) model is the full-blown structural equation model, which in addition to hypothesizing specific latent structures, also hypothesizes relationships among latent structures. For example, the SEM model allows a researcher to hypothesize one latent structure as a predictor of another, and many more possibilities. Many discussions of CFA in the literature are “housed” within the wider SEM framework, and hence it is for this reason that we focus primarily on SEM in this chapter. Confirmatory factor analysis is best understood as a special case of the wider SEM framework.

## 13.2 Why “Causal Modeling” Is Not Causal at All

Hume once defined causality as

We may define a cause to be an object followed by another, ... , where, if the first object had not been, the second never had existed.

(Hume, 1748, Section VIII)

Since the time of Hume, philosophical work on causality has filled volumes, books, and academic careers. The study of causality is a full-time job and truth be told, nobody is still quite sure what causality even means. There are several different versions and interpretations of the meaning of causality (e.g., see Bollen, 1989). Exploring the nature of causality alone in this book would take up at least a chapter if not more, and hence, we do not go there. It will be enough, however, to reject the idea of structural equation models having anything *intrinsically* and *necessarily* to do with causality.

### 13.2.1 Misguided History

If modern causality goes back to Hume, the study of causality itself can be said to go back to the ancient Greeks. The development of statistical regression took place primarily in the late 1880s and early 1890s with the original work of Francis Galton, followed by Karl Pearson and his student, George Udny Yule. Did these pioneers refer to regression models as being causal? Not explicitly, but they did at times use the word *causal* when describing the relationships among the response and predictor variables. However, nowhere did these pioneers ever give regression the power or “license,” *unequivocally*, to make causal claims or draw causal conclusions from non-experimental designs.

Fast-forward to 1917 with the work of Sewall Wright on path analysis, well documented elsewhere (e.g., see Denis & Legerski, 2006). Wright was interested in the generational transmission of color in the guinea pig. From the very inception of Wright’s path models, for which he named the coefficients from such models *causal coefficients*, opposition to such claims of causation surfaced almost immediately. As argued in Denis and Legerski (2006), path models, and their eventual offspring, structural equation models, have no more philosophical authority to claims of causation than the regression models invented in the 1890s. *Causality has nothing to do with the model; it has everything to do with the objects and variables that are being modeled.*

To illustrate the above point, consider a figure (Figure 13.1) from Wright's original guinea pig diagram (left) where he traced what he called the *causal* paths from parent to offspring.

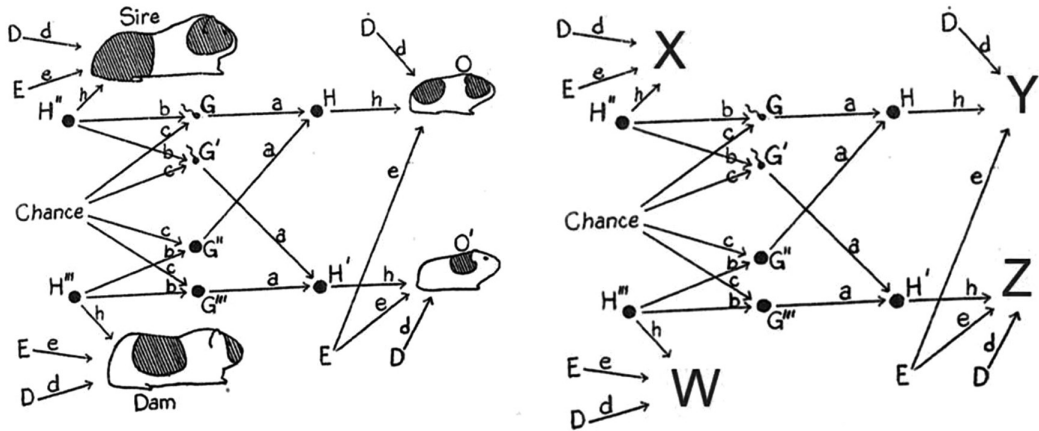


Figure 13.1 Early path diagram by Wright (left). Replacing guinea pig names with arbitrary variable names (right).

Obviously, in the context of parents giving birth to guinea pigs, color transmission was a heritable, biological reality. That is, the fact that offspring had a similar color as their parents could be explained by what was known at the time of heritable influences. Naming such paths “causal” did no harm, as the genetic transmission from parent to offspring agrees more or less with the idea of causality. However, suppose we now replaced the guinea pigs in the picture with simply abstract variables (right):  $X$ ,  $Y$ ,  $W$ , and  $Z$ . Could we now make an analogous causal claim? Does it make any sense to claim that  $W$  causes  $Z$ , for instance? Surely not! After all, we do not even know what these variables even *are*. They are entirely *abstract*.

From this simple example and comparison, it is clear to see that the model alone could never infer causality. Ascribing causal “power” to the statistical *model* instead of to the *variables* on which the model is applied is nothing less than faulty reasoning. *There is nothing inherently causal about a path model or a structural equation model or any other statistical model.* The only conceivable possible exception to this is if all variables in the entire *universe* and all residuals (and their relationships) were able to be included in the model such that the model was fully (and unreasonably on a pragmatic level) specified. Then, it *may* make more sense to speak of the model as causal regardless of the substantive natures of variables used in the model. But of course, such is a fanciful reality that will never be achieved. It is simply unattainable and unrealistic. Hence, identifying or associating causality with the path or structural equation model, or with *any* model for that matter, is simply illogical. Causality is a *scientific* issue, not a *statistical* one.

### 13.2.2 Baron and Kenny (1986)

We fast-forward for a moment to Baron and Kenny's seminal and highly cited 1986 publication, *The Moderator-Mediator Variable Distinction in Social Psychological Research: Conceptual, Strategic, and Statistical Considerations*. In it, the authors wished to emphasize the difference between moderators and mediators, which is a noteworthy contribution. They offer the following definition of a moderator:

In general terms, a moderator is a qualitative (e.g., sex, race, class) or quantitative (e.g., level of reward) variable that affects the direction and/or strength of the relation between an independent or predictor variable and a dependent or criterion variable (p. 1174).

Of course, the above definition, taken *literally*, is somewhat problematic. Statistical moderators have no license whatsoever to *affect* the direction or strength of anything. The word “affect” usually implies “to have an influence on” or similar. As discussed when we surveyed moderation earlier in the book, statistical moderators are simply unable to do such a thing; *substantive moderators*, perhaps, but not *statistical ones*. Calling a variable a *statistical moderator* does not by itself imply that it moderates anything.

Why bring up Baron and Kenny at all here in a discussion of path analysis and structural equation models? We do so for the reason that mediation and moderation models can easily be cast as structural equation models, and hence addressing the causality issue here in relation to SEM models is mandatory. To be fair to Baron and Kenny, they do later in their seminal paper discuss such things as experimental controls and manipulation where causality may possibly be inferred, but the point we wish to make here is that defining moderation or mediation with any kind of *action words* encourages the student (and even researcher!) to falsely and illogically ascribe powers to statistical techniques that they simply do not possess. Some sciences, such as psychology, have a long history of inferring and concluding far “too much” from what a statistical paradigm is philosophically able to provide. We believe that this is in part because in some subfields, carrying out *experimental designs*, where claims of causality may be more reasonable, can be tremendously challenging or impossible due to ethical and other constraints. One cannot randomly assign age, for instance, and hence, many designs in these fields and areas are necessarily *correlational* in nature. Even given these interpretational constraints, however, this may not be enough to stop researchers’ “Freudian id” from stepping in and concluding more from the statistical analysis than what may be warranted.

### 13.3 Is the Variable Measurable? The Observed vs. Unobserved Distinction

Structural equation modeling distinguishes between *observed* or *manifest* vs. *unobserved* variables. With a little thought, the distinction here, like many terms in statistics, gets fuzzy and blurry very quickly, and once more we end up studying the philosophy of these distinctions rather than simply defining them. Still, we must do our best to attempt some definitions here so we can move on and not get stuck in philosophical quicksand.

Essentially, an *observed variable* is any variable for which we can, at least in theory, obtain a *direct measurement*. For instance, height would be regarded as an observed variable. We can easily measure it and have well-defined units to describe it (e.g., inches, feet, etc.). If we are unable to obtain such a *direct* measurement of a variable and need or desire to infer its existence based on other measurable variables, then that variable is usually regarded as *unobserved*. The most classic example of a latent variable is that of intelligence (IQ). Intelligence is a *construct* for which it is difficult or even impossible to measure without inferring it based on other criteria, such as IQ tests, clinical interviews, etc., and hence, to measure something called “IQ,” one must *indicate* it via other more observable variables. Consequently, these observed variables are often referred to as *indicators* for the reason that they *indicate* the unobserved variable.

With this distinction in mind, and again with a little critical thought, it does not take long to realize that all variables are, at least philosophically, *unobservable*, though some variables are less unobservable than others. As an example, consider height once more. Is height observable or unobservable? If we measure your height to be 70 inches, we have, psychometrically, not measured your *true* height. We have merely a *measurement* of your height as an *indicator* of what your

true height might be. Height is thus an unobservable variable but one might argue it is much less unobservable than a variable such as IQ. Likewise, though one's weight is regarded as observable, in the end, weight is also an unobservable construct for which the measurement of weight is but an attempt to “tap into” the construct. This is how, when we push these ideas and distinctions to their philosophical limits, we come to realize that all variables are essentially *unobservable*, but some less so than others. However, for the purpose of building and analyzing path and structural equation models, we will consider the less unobservable variables to be observable and have the capacity of being directly measured. Variables such as height, weight, distance, temperature, biometric pressure, and so on, we will consider to be observable and easily measured variables. Variables such as IQ, self-esteem, happiness, depression, self-actualization, and so on, we will consider to be unobservable or *latent* variables that must be indicated by more observable ones.

### 13.4 Path Analysis: Extending Regression and Previewing SEM

Thus far, we have discussed path analysis informally. We now present the model a bit more formally. Path analysis is essentially a structural equation model that features only observed variables without the *direct* inclusion of unobservable or latent variables. The path model does include residual or *error* variables, however, and these are, by their nature, *latent*. They are considered unobservable because we cannot literally *see* or easily measure what might be “inside” such residual variables, or know precisely what it is composed of, implying it must be indicated by more observable variables. This meets the definition of a latent variable. An equivalent interpretation of a path model is as an extension of the classic regression models surveyed earlier in the book.

Figure 13.2 is a very simple example of a path model (adapted from Denis, 2021a) where IQ from one generation is hypothesized to predict IQ of the next.

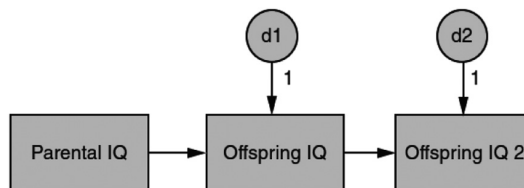


Figure 13.2 Basic path diagram featuring parental IQ predicting offspring IQ. Used with permission of John Wiley & Sons - Books, from *Applied Univariate, Bivariate, and Multivariate Statistics* by Daniel J. Denis, © 2021; permission conveyed through Copyright Clearance Center, Inc.

We see in Figure 13.2 that parent IQ is a predictor of its offspring's IQ, which is then a predictor of the next generation's IQ. In the figure, *d1* and *d2* are *residual* or *error* variables and contain variance unexplained by the predictor preceding it. These typically will have path coefficients set to 1, while other parameters of the model (e.g., path coefficients along arrows) will need to be estimated.

Path models can be useful to researchers and scientists because they allow more complex hypotheses to be stated that are otherwise difficult or even impossible to operationalize and test in classic regression models. As emphasized already, however, they do not allow for concluding causality any more than the typical regression model. In the model of IQ transmission, it probably would not be wrong to conclude that parental IQ is at least a *partial cause* of IQ in offspring, but that causal statement is made as a result of heritable evidence of IQ transmission, not because we are using a path model. *The causal statement is supported by the variables used in the model, not the fact that we are using a path analysis.* This distinction is very important.

### 13.4.1 Exogenous vs. Endogenous Variables

An *exogenous* variable is one emanating from outside of the system and has no arrows pointed toward it. In Figure 13.2, parental IQ is one such exogenous variable. An *endogenous* variable has at least one arrow pointing toward it, such as offspring IQ in the path diagram. Figure 13.3 is another example of a path model, this one featuring hypothesized relationships between parental intelligence, environment, and child's IQ. The only endogenous variable in this model is that of child's IQ, as both parental intelligence and environment are exogenous.

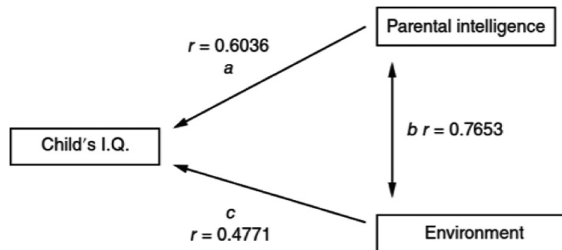


Figure 13.3 Example of a path model, featuring child's IQ as a function of parental intelligence and environment.

In the diagram are correlations ( $r = 0.6036$ ,  $r = 0.4771$ ,  $r = 0.7653$ ), however the theorized relationships include parental intelligence and environment *predicting* child's IQ. The two-sided arrow between parental intelligence and environment is what is generally referred to as an *unanalyzed* relationship, which simply means only the correlation is being modeled and not the estimation of a directional path of one variable onto another. The path from parental intelligence ( $a$ ), for instance, is regarded as an *analyzed* relationship because it is not two-sided but instead hypothesizes a directional relationship, that of parental intelligence predicting child's IQ.

## 13.5 Confirmatory Factor Analysis Model

Having briefly surveyed the path analysis model, we now provide an overview of the confirmatory factor analysis model. Recall once more that the common factor analysis model of the previous chapter was an *exploratory* one, meaning that the user typically entered a number of variables into the model without presumably any hypothesis as to how many factors might be present in the data or what the nature of those factors may be. On a technical level, this translates to there being fewer constraints imposed on the model and any that were included were implicit and part of the model's "default" parameterization. The user did not typically fix, free, or constrain any parameters at will. Of course, a user with some computational experience could alter the exploratory model and re-parametrize it, but the point we are making is that classic factor analysis is generally not amenable to such *precise* model specification. It is an "out-of-the-box" model and we simply run it on data to see what comes out.

In contrast to the exploratory model, the confirmatory factor analysis model is one in which the user typically has, *a priori*, a theory as to how many factors may result from the analysis and which observed variables should load onto which factors. That is, the researcher comes to the model with a *theory* already in mind and is ready to specify the model in great detail in terms of what parameters get estimated, which do not, and other aspects of the model. In this way, the user has much more freedom in modeling because she can essentially fix and free parameters in line with the theory she wishes to evaluate. *Confirmatory factor analysis yields all the power and freedom of modeling to the user.* Of course, one can fix and free parameters in other statistical models as

well and hence, this idea is not unique to CFA models or the wider structural equation modeling framework that generalizes them. For example, in analysis of variance (ANOVA), not testing an interaction term is essentially fixing that parameter estimate to 0, just as not modeling a quadratic term in polynomial regression is implicitly fixing that parameter estimate to 0. Statistical modelers are *implicitly* fixing parameters every day, even if they are only running a simple  $t$ -test! In an SEM model, however, the fixing and freeing of parameters is a prerequisite to even running the model and allows greater specification of the user's theory.

A simple one-factor confirmatory factor analysis model has the same model equation as we saw in the previous chapter for exploratory factor analysis:

$$\mathbf{x} = \Lambda \mathbf{f} + \boldsymbol{\varepsilon} \quad (13.1)$$

where as before,  $\mathbf{x}$  are observed variables,  $\mathbf{f}$  are latent factors,  $\Lambda$  is a loading matrix, and  $\boldsymbol{\varepsilon}$  is a matrix of errors. The number of factors in  $\mathbf{f}$  is prespecified by the researcher. The following is a model with a two-factor structure hypothesizing factors  $f_1$  and  $f_2$ :

$$\begin{array}{c} \mathbf{x} \\ \left[ \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right] \end{array} = \begin{array}{c} \Lambda \\ \left[ \begin{array}{cc} \lambda_{11} & 0 \\ \lambda_{21} & 0 \\ 0 & \lambda_{32} \\ 0 & \lambda_{42} \end{array} \right] \end{array} \begin{array}{c} \mathbf{f} \\ \left[ \begin{array}{c} f_1 \\ f_2 \end{array} \right] \end{array} + \begin{array}{c} \boldsymbol{\varepsilon} \\ \left[ \begin{array}{c} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \end{array} \right] \end{array}$$

which yields the following equations:

$$\begin{aligned} x_1 &= \lambda_{11}f_1 + 0f_2 + \varepsilon_1 \\ x_2 &= \lambda_{21}f_1 + 0f_2 + \varepsilon_2 \\ x_3 &= 0f_1 + \lambda_{32}f_2 + \varepsilon_3 \\ x_4 &= 0f_1 + \lambda_{42}f_2 + \varepsilon_4 \end{aligned}$$

Note that only  $x_1$  and  $x_2$  are hypothesized to “load” on  $f_1$  as reflected by the loading matrix  $\Lambda$  constraining the loading estimates for  $x_3$  and  $x_4$  to 0. The key distinction in the confirmatory model in comparison to the exploratory model of the previous chapter is in precisely specifying the nature of the factor matrix,  $\mathbf{f}$ , and hence attempting to “confirm” a very *specific* hypothesized factor structure through the precise specification of parameters.

### 13.6 Structural Equation Models

Having briefly surveyed path analysis and confirmatory factor analysis, we are now ready to survey the full structural equation model. The full structural equation model is given by

$$\boldsymbol{\eta} = \mathbf{B}\boldsymbol{\eta} + \boldsymbol{\Gamma}\boldsymbol{\xi} + \boldsymbol{\zeta} \quad (13.2)$$

where  $\boldsymbol{\eta}$  is a latent endogenous vector, which we note appears on both sides of the equation. Contrary to classic linear models in which a variable only appears on one side of the equation, in the SEM model,  $\boldsymbol{\eta}$  are allowed to predict one another, which is why we see it both on the left side and right side of the model (in  $\mathbf{B}\boldsymbol{\eta}$ ). This parameterization allows a latent variable to predict another latent variable in an SEM model. The exogenous variables are in the vector  $\boldsymbol{\xi}$ , weighted by

coefficients in  $\Gamma$ , and  $\zeta$  is a vector of latent errors, also known as *disturbances* in the SEM literature. All parameter estimates for the model are contained in matrices  $\mathbf{B}$  and  $\Gamma$ .

Assumptions for the model in (13.2) include  $E(\eta) = 0$ ,  $E(\xi) = 0$ , and  $E(\zeta) = 0$ , as well as latent errors being uncorrelated with latent exogenous variables. For further details on this classic SEM parameterization, see Bollen (1989).

### 13.6.1 Covariance Modeling

Recall that in prior methods surveyed in this book, the goal was essentially to reproduce some function of the data. In regression, we sought to reproduce the response  $y$  by estimating predicted values,  $\hat{y}$ . In SEM and covariance models in general, the goal is also to reproduce some function of the data, but this time, it is the *observed covariance matrix* that we will attempt to reproduce by choosing which parameters to estimate and which to keep fixed. This is not that different from the factor analysis model in which the end goal, mathematically, was to regenerate the covariance or correlation matrix via specification of the model; *the “better” the model, the more precisely we should be able to reproduce the covariance or correlation matrix.*

Covariance modeling comprises an enormous class of models. Bollen (1989) has shown that from the following simple equality, a whole host of models can be deduced as special cases,

$$\Sigma = \Sigma(\theta) \tag{13.3}$$

where the above reads that  $\Sigma$  (the covariance matrix) is a function of model parameters contained in the vector  $\theta$ . That is,  $\Sigma(\theta)$  is the *implied model-based covariance matrix* that regenerates the observed covariance matrix,  $\Sigma$ . Choosing which parameters to *fix* and which to *free*, along with other model specification considerations, will determine how well the model fits the data. Note once more that this is exactly the same thing we do in other models from regression to ANOVA and beyond; the only difference is that such specifications in those models, at least the finer ones, are often not made as *explicit*. The equality in (13.3) is far-reaching, and special cases of it at their most fundamental level simply require decisions made by the user in conjunction with the types of variables and data they are working with in any given context. As Bollen (1989, p. 2) summarizes:

The simplicity of this equation [i.e.,  $\Sigma = \Sigma(\theta)$ ] is only surpassed by its generality. It provides a unified way of including many of the most widely used statistical techniques in the social sciences. Regression analysis, simultaneous equation systems, confirmatory factor analysis, canonical correlations, panel data analysis, ANOVA, analysis of covariance, and multiple linear indicator models are special cases of  $\Sigma = \Sigma(\theta)$ .

The value of  $\Sigma$  is unknown, and hence we estimate  $\Sigma$  via  $\mathbf{S}$ , the sample covariance matrix. We desire the distance between  $\mathbf{S}$  and  $\mathbf{S}(\theta)$  to be as small as possible, that is, to be *minimized*, where  $\mathbf{S}(\theta)$  is the sample covariance matrix *implied* by the estimated model parameters. *Discrepancy functions* are a wide class of functions from which can be selected a specific function to minimize distance between  $\mathbf{S}$  and  $\mathbf{S}(\theta)$ . One such discrepancy function is that of the familiar *ordinary least-squares* that minimizes the distance between observed and fitted values of  $y$ . Other types of models feature the minimizing or maximizing of other discrepancy functions. In covariance modeling using maximum likelihood (ML), for instance, we wish to minimize the following fitting function  $F_{ML}$  (Bollen, 1989, p. 107):



$$F_{ML} = \log |\sum \boldsymbol{\theta}| + \text{tr}(\mathbf{S} \sum^{-1}(\boldsymbol{\theta})) - \log |\mathbf{S}| - (p + q) \quad (13.4)$$

When  $\mathbf{S} = \mathbf{S}(\boldsymbol{\theta})$ ,  $F_{ML} = 0$ . That is, as  $(\mathbf{S} - \mathbf{S}(\boldsymbol{\theta})) \rightarrow 0$ ,  $F_{ML} \rightarrow 0$ , and perfect model fit is achieved, since the difference between the sample covariance matrix  $\mathbf{S}$  and that implied by the model,  $\mathbf{S}(\boldsymbol{\theta})$ , is equal to 0. It turns out that  $(n-1)F_{ML} \sim \chi^2$  (where  $n$  is sample size), which can be used for evaluating the following global null hypothesis,

$$H_0: \sum = \sum(\boldsymbol{\theta}) \quad (13.5)$$

where the above null hypothesis  $H_0$  reads that the population covariance matrix  $\Sigma$  is equal to that population covariance matrix implied by the specified parameters, that is,  $\sum(\boldsymbol{\theta})$ . The above null is evaluated against the alternative hypothesis  $H_1$ ,

$$H_1: \sum \neq \sum(\boldsymbol{\theta}) \quad (13.6)$$

which reads that the population covariance matrix  $\Sigma$  is unequal to that implied by the model parameters in  $\boldsymbol{\theta}$ . Equivalently, under the null hypothesis,  $\sum - \sum(\boldsymbol{\theta}) = 0$ , and hence if the model fits *perfectly*, the expectation for  $\chi^2$  is 0. The extent to which the model does not fit is the extent to which  $\sum - \sum(\boldsymbol{\theta}) \neq 0$ . If you look at the null hypothesis closely, you will see that contrary to traditional null hypothesis testing in which we wish to reject the null, in SEM, we wish to actually *not* reject the null hypothesis. That is, *a null that is supported by the data is a favorable outcome in SEM*, which goes against how traditional null hypothesis testing is classically interpreted. This fact is important, since if you reject the null in SEM, it suggests your estimated model is *not* consistent with the covariance matrix, meaning that it does not reproduce it well, which further implies the fit between the covariance matrix based on the data is not in line with the covariance matrix implied by the model.

### 13.6.2 Evaluating Model Fit

As with most supervised learning models, we desire a measure to assess or evaluate how well the model fits the empirical data. In ordinary least-squares, that measure of model fit is typically  $R^2$ ,  $R_{adj}^2$ , or some other model selection measures such as *AIC* or *BIC*. In SEM, we also require measures of model fit to assess the extent to which our designated model parameters reproduce  $\mathbf{S}$ , the sample covariance matrix. As in regression and other models, there are numerous ways of assessing fit in structural equation models. Measures of model fit in SEM are typically divided into two broad categories, those assessing *overall* or *absolute* model fit, and those evaluating *incremental fit*, sometimes called *relative fit* measures. Overall measures of model fit assess the model in its entirety without reference to a competing model, whereas incremental or relative measures of fit evaluate a model relative to some reference or competing model, usually a null or baseline model, to evaluate how well the hypothesized model performs relative to the baseline model.

Why not just use  $\chi^2$  as a measure of model fit? Recall that  $(n-1)F_{ML} \sim \chi^2$  and the expectation for  $\chi^2$  under  $H_0$  is equal to 0 under a perfectly fitting model. Hence, it might at first glance seem reasonable to simply use  $\chi^2$  as a measure of model fit such that the extent to which  $\chi^2 \rightarrow 0$  is the extent to which the model is increasingly well-fitting and the extent to which  $\chi^2 > 0$  is the extent to which the model fits poorly. That is, the extent to which  $\chi^2$  is statistically significant might be taken as a measure of increasingly poor model fit. While this approach at first glance sounds reasonable, it is very problematic, and overall,  $\chi^2$  alone makes for a poor measure of model fit in

SEM. Recall that as  $n$  increases, achieving  $p < .05$  (or some other arbitrary significance level) for  $\chi^2$  is assured other than under the restriction that  $\chi^2 = 0$ . So long as  $\chi^2 > 0$  is true, and it most often will be, statistical significance of  $\chi^2$  and more importantly, lack thereof, should not be in any real way interpreted as a measure of “effect,” especially for increasing  $n$ .

Of course, this idea is not new and was discussed earlier in the book; *p-values are not effect sizes*. That is not to say a user should not utilize  $\chi^2$  as an initial assessment of fit, only that  $\chi^2$  alone will usually not tell the user what they really want to know, which is a measure of effect that is *independent* of the influence of sample size. In the measures that follow, both overall measures and incremental fit indices are measures of effect size (or of “fit” more generally) that are not influenced by the magnitude of  $n$  in the way that  $\chi^2$  is to a large extent a function of sample size.

### 13.6.3 Overall (Absolute) Measures

Some of the more common measures of overall fit include *root mean square residual* (RMR), its standardized counterpart, the *standardized root mean square residual* (SRMR), and the *root mean square of approximation* (RMSEA). We now briefly survey each of these.

The *root mean square residual* (RMR) is given by

$$\text{RMR} = \left[ 2 \sum_{i=1}^q \sum_{j=1}^i \frac{(s_{ij} - s'_{ij})^2}{q(q+1)} \right]^{1/2} \quad (13.7)$$

where  $s_{ij}$  is an element of the observed covariance matrix  $\mathbf{S}$ ,  $s'_{ij}$  is an element of the model-implied covariance matrix  $\mathbf{S}(\boldsymbol{\theta})$ , and  $q$  is the number of observed variables in the model. Essentially, RMR assesses the extent to which elements of the observed covariance matrix differ from those of the implied by taking their (squared) difference,  $(s_{ij} - s'_{ij})^2$ . The extent to which that difference is larger than not is the extent to which RMR will also be large, which would be increasingly indicative of a poorly fitting model. The best-fitting model would have  $s_{ij} - s'_{ij} = 0$ , which would then imply an RMR of 0. Note that all else in the formulation are simply constants, as well as squaring and taking the square root. That is, the most meaningful part of the formula is contained in  $(s_{ij} - s'_{ij})$ .

One drawback of RMR is that  $s_{ij}$  is computed on *raw scores*, and hence analogous to how the size of a covariance can be misleading due to the extent of variance in one or both variables, RMR can likewise be small or large, not solely due to the extent of effect present, but instead due to magnitudes of variance in the variables. Solving the scale issue with RMR is the *standardized root mean square residual* (SRMR) which first standardizes residuals  $s_{ij} - s'_{ij}$  via the product of standard deviations,  $s_i s_j$ ,

$$(s_{ij} - s'_{ij}) / s_i s_j$$

yielding

$$\text{SRMR} = \left[ 2 \sum_{i=1}^q \sum_{j=1}^i \frac{[(s_{ij} - s'_{ij}) / s_i s_j]^2}{q(q+1)} \right]^{1/2} \quad (13.8)$$

where now the only functional part of the formula continues to be in the numerator, only that now the deviations  $(s_{ij} - s'_{ij})$  have been standardized. As a result of RMR and SRMR being an analysis of residuals from the model, smaller values than not are indicative of better-fitting models. Values

of 0.08 or less usually indicate reasonable to increasingly good fit for SRMR, though of course any model fit measure computed on data is often only as good as what competing research has found in the given area of investigation. As was true for effect sizes in simpler regression and ANOVA models for instance, effects should always be evaluated relative to other findings in the area under investigation. If you achieve a value for SRMR of 0.05 but all of your colleagues in the same area of investigation are achieving SRMR values of 0.0001 with their models, no criteria or “guidelines” can save you from concluding that your model may not be worth keeping.

The root mean square error of approximation (RMSEA) is given by

$$\text{RMSEA} = \sqrt{\frac{1}{n-1} \left( \frac{\chi_m^2 - df_m}{df_m} \right)} \quad (13.9)$$

Recall that  $(n-1)F_{ML} \sim \chi^2$  and thus the statistical significance of an SEM model is based on  $\chi^2$ , where now  $\chi_m^2$  is the chi-square resulting from the model. Recall also that smaller than not values for  $\chi_m^2$ , all else equal, are indicative of an increasingly well-fitting model. Whenever a parameter is estimated, it “costs” a degree of freedom. Ideally, we would wish to achieve a superb model fit without spending too many degrees of freedom. “Spending” a lot of degrees of freedom and estimating everything in the model is suggestive of a poor theory guiding the SEM (and will likely lead to identifiability issues for the model as well, though a discussion of identifiability is beyond the scope of the current text). Estimating everything under the sun, as we know by now, is also an exercise in *overfitting*.

With the above in mind, we can now see how RMSEA works; as the distance  $\chi_m^2 - df_m$  gets larger relative to the value of  $df_m$ , RMSEA grows in magnitude. The only way this happens is if  $\chi_m^2$  is larger than not, which recall, would be suggestive of an increasingly poor fit. A better fit is indicated by increasingly smaller values of  $\chi_m^2$ . Note as well that the fraction  $1/(n-1)$  is inconsequential here and is not the *functional* part of the formulation, it simply serves to “average” the ratio  $(\chi_m^2 - df_m)/df_m$ . Values for RMSEA in the approximate range of 0.01, 0.05, 0.08 are generally indicative of excellent, good, and relatively poor fitting models. However, once more, context matters and one should never rely *absolutely* on *relative* guidelines in any field, including statistics.

### 13.6.4 Incremental Fit Indices

We now briefly survey a few of the more common measures of incremental fit. These measures, recall, evaluate model fit not in absolute or global terms, but instead relative to some baseline or null model.

The *normed-fit index* (NFI) is given by,

$$\Delta_1 = \frac{\chi_b^2 - \chi_m^2}{\chi_b^2} = \frac{F_b - F_m}{F_b} \quad (13.10)$$

where  $\chi_b^2$  is the baseline model and  $\chi_m^2$  is the hypothesized model. Since  $\chi^2 \sim (n-1)F_{ML}$ ,  $F_b - F_m$  is the difference between the corresponding fitting functions,  $F_b$  for the baseline model and  $F_m$  for the model under test. The baseline model has more restrictions (i.e., on parameter estimates) than the fitted model, and so the difference  $F_b - F_m$  is a reflection of how much the fitting function *decreases* due to the proposed model. The better fitting the proposed model, the smaller is the value of  $F_m$  and the difference  $F_b - F_m$  increasingly approximates simply  $F_b$ . Recall that smaller

values than not for  $F_m$  are indicative of an increasingly better model fit. Ideally, if the model fit perfectly, then  $F_m$  would equal 0, and thus  $\Delta_1$  would equal 1, as easily demonstrated,

$$\Delta_1 = \frac{\chi_b^2 - \chi_m^2}{\chi_b^2} = \frac{F_b - F_m}{F_b} = \frac{F_b - 0}{F_b} = \frac{F_b}{F_b} = 1$$

On the opposite end of things, if there is no model improvement, then  $F_b = F_m$  and so

$$\Delta_1 = \frac{\chi_b^2 - \chi_m^2}{\chi_b^2} = \frac{F_b - F_m}{F_b} = \frac{0}{F_b} = 0$$

Values of 0.95 and greater are typically indicative of well-fitting models, not in absolute terms, but relative to a baseline model, which is usually the null model but need not necessarily be so.

A measure of model fit related to  $\Delta_1$  is that of Bollen's  $\rho_1$  (Bollen, 1989), which is essentially the same as the NFI, but with one important difference in that it scales  $\Delta_1$  relative to degrees of freedom:

$$\begin{aligned} \rho_1 &= \frac{(F_b / df_b) - (F_m / df_m)}{(F_b / df_b)} \\ &= \frac{(\chi_b^2 / df_b) - (\chi_m^2 / df_m)}{(\chi_b^2 / df_b)} \end{aligned} \quad (13.11)$$

Under the most ideal of situations assuming perfect fit,  $F_m = 0$  and thus  $F_m / df_m$  will also equal 0 regardless of  $df_m$ . Under the condition that  $F_m > 0$ , which will most often be the case,  $F_m$  is essentially *penalized* by how many degrees of freedom  $df_m$  it required to reduce  $F_b$  to  $F_m$ . *Strong theories require spending less degrees of freedom to achieve increasingly smaller  $F_m$  that do weaker theories.* We can see then that  $\rho_1$  rewards the fitting of models that “spend” a smaller number of degrees of freedom, all else equal, to achieve the same reduction from  $F_b$  to  $F_m$ . That is,  $\rho_1$  rewards models that are *parsimonious* and result from a well-specified theory such that the researcher clearly had a good framework for what parameters to estimate to result in maximal decrease in  $F_m$ . Such will be reflected in a ratio  $F_m / df_m$  smaller than not, which means less and less is being subtracted from the baseline ratio  $F_b / df_b$ . As  $(F_m / df_m) \rightarrow 0$ ,  $\rho_1 \rightarrow 1.0$  indicating perfect fit. Values for  $\rho_1$  generally in the neighborhood of 0.95 or greater are usually regarded as indicating good to excellent fit.

The *Tucker-Lewis index* (NNFI) is given by

$$\rho_2 = \frac{(\chi_b^2 / df_b) - (\chi_m^2 / df_m)}{(\chi_b^2 / df_b) - 1} \quad (13.12)$$

A comparison of  $\rho_1$  to  $\rho_2$  reveals that  $(\chi_b^2 / df_b) - (\chi_m^2 / df_m)$  is no longer being compared to  $(\chi_b^2 / df_b)$  but instead to  $(\chi_b^2 / df_b) - 1$ . Values of 0.95 and greater are suggestive of increasingly good model fit.

Lastly, the *comparative fit index* (CFI) is given by

$$CFI = 1 - \frac{\max(\chi_m^2 - df_m, 0)}{\max(\chi_b^2 - df_b, \chi_m^2 - df_m, 0)} \quad (13.13)$$

The CFI follows a similar logic to that of  $\rho_1$ , but instead of taking ratios as for  $\rho_1$ , we are subtracting  $df$  from  $\chi^2$ , that is,  $\chi^2 - df$  for both baseline and model  $\chi^2$ . CFI values in the range of 0.90 to 0.95 indicate good fit.

The aforementioned measures of model fit, whether absolute or relative, are just a very small sample of model fit measures available in SEM. Specialists in SEM continue to publish new measures it seems almost yearly, all of which offer different ways to assess and evaluate model fit. An understanding of these most common ones, however, opens the door to understanding many variations and gets the proverbial “foot in the door” to the vast area of model fit statistics in SEM.

### 13.7 Structural Equation Modeling with Nonlinear Effects

Most structural equation models, as we have seen thus far with so many models in statistics and machine learning, assume *linearity in the parameters*. They are a complex network of otherwise *linear models*. As we know by now, assuming linearity can provide relatively good generalizability of findings and hence relatively low variance. However, if the true relationship underlying the data is *nonlinear*, then assuming linearity can seriously bias the model as well as bias parameter estimates generated based on such a restrictive assumption. Indeed, entire trends or features in the data can be missed if attempting to fit more complex models is not attempted (Kelava & Brandt, 2014).

For example, a linear SEM model may fail to detect a quadratic effect entirely, which may otherwise prove to be very important to the investigator on a scientific level. And since structural equation models can increase in complexity very quickly, exploratory visualization of such models is much more difficult or even impossible compared to visualizing much simpler models, for example a scatterplot where one might immediately detect nonlinearity simply by observing the trend in the corresponding plot. Visualization of multivariate data, especially in complex models such as SEM or neural networks as we will see later in the book, can prove daunting to say the least. This implies that detecting nonlinearity in such data can be equally challenging.

When structural relationships are, in fact, nonlinear, but nonetheless fit using a linear model, many issues arise, the most serious of which is that latent endogenous variables can no longer be assumed to be normally distributed as they are assumed so in linear SEM models (Klein & Muthén, 2007). This implies that parameter estimates cannot be determined based solely on the covariance structure of the data (Vegelius & Jin, 2021). A failure to include nonlinear effects in a structural equation model when such effects are present, as mentioned, can result in biased estimators (Rdz-Navarro & Yang-Wallentin, 2020).

There has been much work in the development and estimation of nonlinear structural equation models. Klein and Muthén (2007) developed a nonlinear structural equation model using a *quasi-maximum-likelihood* approach for estimation, which when evaluated in simulation studies produced robust standard errors and was relatively computationally inexpensive. Bollen (1996) proposed a two-stage least-squares approach (2SLS), and more recently Holst and Budtz-Jørgensen (2020) have also proposed a two-stage least-squares estimation method. An efficient maximum-likelihood *latent moderated structural equations* approach has also recently been advanced (Klein & Moosbrugger, 2000; Umbach et al., 2017).

#### 13.7.1 Example of a Nonlinear SEM

The following is an example of a nonlinear SEM model, one without an intercept but featuring an interaction term,

$$\eta = \gamma_1 \xi_1 + \gamma_2 \xi_2 + \omega_{12} \xi_1 \xi_2 + \zeta \quad (13.14)$$

where the interaction term is given by  $\omega_{12} \xi_1 \xi_2$ , and hence  $\eta$  is said to be a nonlinear function of  $\xi_1$  and  $\xi_2$ , which implies  $\eta$  has an expectation that will be *non-Gaussian*. One way to deal with this nonlinearity historically has been to treat the product term  $\xi_1 \xi_2$  as a separate latent variable and construct so-called “indicators” based on products of original indicators (Vegelius, 2022). That is, the new variable becomes  $\xi_3 = \xi_1 \xi_2$  (e.g., see Kenny & Judd, 1984). The inclusion of product terms, however, can lead to relatively smaller communalities as well as larger sampling fluctuation and multicollinearity in addition to the aforementioned nonnormal error terms (Klein & Muthén, 2007).

An example of a more complex model, one that not only includes an interaction term but also quadratic terms, is the following (Kelava & Brandt, 2014):

$$\eta = \alpha + \gamma_1 \xi_1 + \gamma_2 \xi_2 + \gamma_3 \xi_1 \xi_2 + \gamma_4 \xi_1^2 + \gamma_5 \xi_2^2 + \zeta \quad (13.15)$$

where now the quadratic terms are given by  $\gamma_4 \xi_1^2$  and  $\gamma_5 \xi_2^2$  in addition to the aforementioned interaction term,  $\xi_1 \xi_2$ . Due to the fact that interaction terms and quadratic terms are often correlated, Klein and Muthén (2007) proposed including quadratic effects to rule out spurious interaction effects, which gave rise to the more general nonlinear SEM model,

$$\eta = \alpha + \mathbf{T} \xi + \xi' \Omega \xi + \zeta \quad (13.16)$$

in which the quadratic term is now indicated by  $\xi' \Omega \xi$  and where the corresponding measurement equations for the latent variables are the following:

$$\begin{aligned} \mathbf{x} &= \tau_{\mathbf{x}} + \Lambda_{\mathbf{x}} \xi + \delta \\ \mathbf{y} &= \tau_{\mathbf{y}} + \Lambda_{\mathbf{y}} \eta + \varepsilon \end{aligned}$$

The model assumes that  $\mathbf{x}$  is multivariate normal,  $\delta$  and  $\varepsilon$  are multivariate normal and mutually independent where  $E(\delta) = 0$  and  $E(\varepsilon) = 0$ . The covariance assumptions include  $Cov(\delta, \xi') = 0$  and  $Cov(\varepsilon, \xi') = 0$ ,  $Cov(\zeta, \xi') = 0$ ,  $Cov(\zeta, \delta') = 0$ , and  $Cov(\zeta, \varepsilon') = 0$ . For further details on the specification of this model, see Klein and Moosbrugger (2000).

Nonlinear models in SEM can be fit using the package *nlsem* (Umbach et al., 2017). Consult Umbach et al. (2017) for an introduction and details about the package.

### 13.7.2 Structural Equation Nonparametric and Semiparametric Mixture Models

We have thus far surveyed the general nonlinear SEM model featuring parametric approaches to its estimation. An alternative and promising approach to estimating nonlinear SEM models is that of using a *semiparametric* approach (SEMM), first proposed by Bauer (2005) featuring mixture modeling. The central idea behind mixture modeling in this context is to represent what is a nonnormal distribution as a combination of typically normal densities (Kelava et al., 2014). The density of the nonnormal distribution is then computed as a weighted sum of the (typically) normal densities. We say “typical” because the SEMM allows for flexible approximations of nonnormal distributions (Umbach et al., 2017). The nonparametric component of these models, as is true of other nonparametric models studied in this book, better allows the data to “speak for themselves”

as it were rather than have an *a priori* functional form imposed on them such as the classic linearity or similar assumption about parametric form.

A mixture density of American adults (males vs. females) is a classic example of a mixture density where each distribution contributes to the mix. Essentially, and most generally, a mixture density is a type of combination, or crudely, an *average* of other normal or nonnormal densities. In traditional non-mixture maximum-likelihood SEM, multivariate normality is typically assumed for the indicators  $\mathbf{x}, \mathbf{y}$  of the model, where each of  $\mathbf{x}, \mathbf{y}$  are vectors corresponding to a number of observed or “manifest” indicator variables. That is,

$$f(\mathbf{x}, \mathbf{y}) = N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  are the implied mean and covariance structures. The mixture model can be considered as a simple extension of the above,

$$f(\mathbf{x}, \mathbf{y}) = \sum_{g=1}^G w_g N(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g) \quad (13.17)$$

where  $g = 1$  through  $G$  are now the number of mixture components,  $w_g$  are mixture probabilities and where each component is distributed multivariate normal,  $N(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$ . The sum of  $w_g$  must equal one to ensure it is a proper density. Each of the  $g = 1$  through  $G$  mixture components will then have their own individual SEM model (Kelava et al., 2014, p. 471),

$$\eta_g = B_g \eta_g + \alpha_g + \Gamma_g \xi_g + \zeta_g$$

where we now note the subscript  $g$  to denote the given model under consideration (e.g.,  $g = 1$  would be the first model, and so on). Most often, at least some of the parameters across the mixture models are constrained to be equal so that identification of the model is possible (Kelava et al., 2014). However, the model is still yet linear in the parameters. To make it a *semiparametric nonlinear model*, which of course is our ultimate goal here, the model becomes

$$\eta_g = B_g \eta_g + \alpha_g + \Gamma_{1g} \xi_g + \Gamma_{2g} h(\xi_g) + \zeta_g \quad (13.18)$$

where the nonlinear terms are included in  $\Gamma_{2g} h(\xi_g)$  and for generality, their exact nature is not specified specifically in this formulation. That is,  $h(\xi_g)$  could include product terms, quadratic terms, and so on, which implies the model has become nonlinear in  $h(\xi_g)$ . For further details on this model, see Kelava et al. (2014), which is an exceptionally clear and well-written summary of mixture models and nonlinearity in SEM.

Other mixture models are seen operating where finite mixtures of linear SEMs are used to approximate unknown nonlinear relationships of the latent variables (Umbach et al., 2017). Since the semiparametric approach does not *a priori* designate the *type* of functional relationship between latent variables, curvilinear relationships can be modeled between latent variables (Umbach et al., 2017). Song et al. (2014) propose nonparametric estimation of multiview latent variable models as special cases of Bayesian networks. In R, the aforementioned package *nlsem* can also be used to fit semiparametric SEM models. Once more, consult Umbach et al. (2017) for details.

SEM models can also accommodate longitudinal data. Such models usually go by the name of *latent curve modeling*, *latent growth curves*, and *latent trajectory modeling*, among others. See Bollen and Curran (2006) for details. Liu (2023) provides a comprehensive review of longitudinal

models in SEM as well as R packages for fitting them. Liao et al. (2020) propose a *generalized structural equation model* that incorporates neural networks in their estimation. Sol and Baras (2022) combine a neural network approach with structural equation modeling in modeling activity trackers.

### 13.8 Caveats Regarding SEM Models

In concluding our discussion of structural equation models, we feel it relevant to clarify a few items of confusion that often arise for those trying to understand these models for the first time.

The first is that SEM models are often marketed as accounting for *measurement error*, presumably because they include unique *error terms* for variables as well as more than a single indicator of a construct, such as is typically the case in simpler models. Including error terms and more indicators than not usually lowers the error with which a construct is measured (or *indicated*). However, if one uses poor indicators, minimizing measurement error is not the chief concern; *the priority instead will be to choose better indicators*. That is, accounting for and minimizing measurement error is only as relevant as the quality of indicators used in reducing said measurement error. For example, measuring heart rate with a stethoscope has relatively little measurement error. Including a second indicator of heart rate or level of cholesterol would not reduce measurement error, but would instead simply make for a relatively poorly defined or indicated latent variable. That is, measurement error can be reduced in SEM but only if indicators are chosen carefully and wisely. *Structural equation models do not unconditionally reduce measurement error. They simply account for it better than many classical statistical models.*

A second caveat is regarding the aforementioned issue of causality. *SEM models have nothing, by themselves, to do with causality!* If the *context* of the model is reasonable, then one *may* be able to infer causality, but by themselves, SEM models are not any more causal than any other statistical or mathematical model or abstract mathematical system. The fact that SEM models often go by the name of *causal models* is not due to their ability to impart or impose causality via their structural relations or parameter estimates. They are named causal models largely for historical reasons and also because researchers often use them on data networks they *believe, a priori*, represent *causal processes*. However, the causality assumption arises from the nature of the *science*, not the *statistical method*. For Wright's pioneering work on path analysis, assuming a causal process in the generational transmission of color in the guinea pig made good sense because it had a scientific biological "organic" basis. *Causality is primarily a design and scientific issue, not a statistical one.*

A third caveat is regarding *context*. As we have discussed throughout this book, all models are *context-dependent*, which means that any relations we observe in a model depend on other variables included or not included in the model. Since perfect specification of a model is rarely if ever possible, virtually all SEM models constructed will be incomplete and hence all parameter estimates will be to some extent, *biased*. All modeling (not just SEM) is but an *approximation* to reality. It could very well be that including an extra few variables in the model results in a vastly different model, so it is essential that when planning an SEM model, or any other model for that matter, the researcher ensures that all instrumental variables are included in the model as to minimize the extent to which it is misspecified.



### 13.9 SEM Models in R

As mentioned in our discussion earlier, the package *nlsem* (Umbach et al., 2017) in R fits nonlinear structural equation models as well as mixture models. It will also fit special cases of these models which are traditional linear SEM models.

The package *modsem* (Slupphaug, 2024) can be used in conjunction with the *lavaan* package (Rosseel, 2012) for testing interactions between latent variables. As an example from the package's documentation, we briefly consider the following three latent variables where we wish to test the corresponding latent interaction term (XZ).

```
> library(modsem)
> m1 <- `
+ X =~ x1 + x2 + x3
+ Y =~ y1 + y2 + y3
+ Z =~ z1 + z2 + z3
+ Y ~ X + Z + X:Z`
```

Above we have created the three latent variables, where X is indicated by x1, x2, and x3, Y by y1, y2, and y3, and Z by z1, z2, and z3 (Slupphaug, 2024). Below we fit the model and request a summary:

```
> est1 <- modsem(m1, oneInt)
> summary(est1)
```

ModSEM:

Method = dblcent  
lavaan 0.6-18 ended normally after 159 iterations

|                                  |            |
|----------------------------------|------------|
| Estimator                        | ML         |
| Optimization method              | NLMINB     |
| Number of model parameters       | 60         |
| Number of observations           | 2000       |
| Model Test User Model:           |            |
| Test statistic                   | 122.924    |
| Degrees of freedom               | 111        |
| P-value (Chi-square)             | 0.207      |
| Parameter Estimates:             |            |
| Standard errors                  | Standard   |
| Information                      | Expected   |
| Information saturated (h1) model | Structured |

We see from the above that the model was fit by maximum likelihood. The value of the chi-square statistic for the model is equal to 122.924 on 111 degrees of freedom. The *p*-value for the chi-square statistic is equal to 0.207, which recall in SEM a non-statistically significant result is considered a “good” result.

Below are the paths from the hypothesized latent variables to the observed or “manifest” variables:

Latent Variables:

|       | Estimate | Std.Err | z-value | P(> z ) |
|-------|----------|---------|---------|---------|
| X =~  |          |         |         |         |
| x1    | 1.000    |         |         |         |
| x2    | 0.778    | 0.012   | 64.933  | 0.000   |
| x3    | 0.888    | 0.014   | 65.681  | 0.000   |
| Y =~  |          |         |         |         |
| y1    | 1.000    |         |         |         |
| y2    | 0.801    | 0.004   | 188.597 | 0.000   |
| y3    | 0.905    | 0.004   | 201.940 | 0.000   |
| Z =~  |          |         |         |         |
| z1    | 1.000    |         |         |         |
| z2    | 0.807    | 0.041   | 19.909  | 0.000   |
| x3    | -0.002   | 0.012   | -0.133  | 0.894   |
| XZ =~ |          |         |         |         |
| x1z1  | 1.000    |         |         |         |
| x2z1  | 0.778    | 0.013   | 59.190  | 0.000   |
| x3z1  | 0.884    | 0.014   | 61.242  | 0.000   |
| x1z2  | 0.843    | 0.031   | 27.045  | 0.000   |
| x2z2  | 0.651    | 0.026   | 24.931  | 0.000   |
| x3z2  | 0.739    | 0.029   | 25.269  | 0.000   |
| x1x3  | 0.358    | 0.033   | 10.743  | 0.000   |
| x2x3  | 0.293    | 0.026   | 11.284  | 0.000   |
| x3x3  | 0.321    | 0.032   | 9.889   | 0.000   |

Regressions:

|     | Estimate | Std.Err | z-value | P(> z ) |
|-----|----------|---------|---------|---------|
| Y ~ |          |         |         |         |
| X   | 0.712    | 0.059   | 12.009  | 0.000   |
| Z   | 0.468    | 0.063   | 7.461   | 0.000   |
| XZ  | 0.643    | 0.059   | 10.944  | 0.000   |

We see that all interaction terms are statistically significant as well as most other paths. We can obtain a plot of the interaction as follows (Figure 13.4), where we note the lines are not parallel (for values of  $z = -0.2$  and  $0$  [ $c(-0.2, 0)$ ], we encourage the reader to try other values):

```
> plot_interaction("X", "Z", "Y", "X:Z", -3:3, c(-0.2, 0), est1)
```

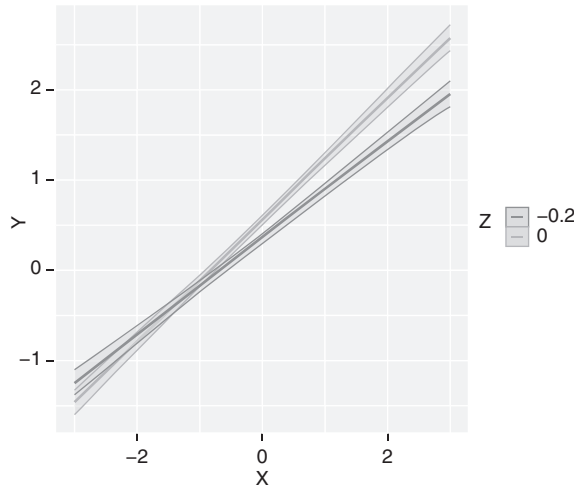


Figure 13.4 Interaction plot generated from SEM data. Lines cross, suggesting a latent variable interaction.

The package *SEMgraph* (Grassi et al., 2024) also has capabilities for fitting factor analysis and structural equation models. The package *semopy* can be used to fit structural equation models in Python. For details, see [semopy.com](http://semopy.com).

### 13.10 Chapter Review Exercises

1. Distinguish between *exploratory* vs. *confirmatory* factor analysis. How are they similar? Different?
2. Explore the idea of *causality* through online searches. Why is causality so difficult to define precisely and poses so many philosophical challenges to philosophers of science?
3. Distinguish between *observable* vs. *unobservable* variables. If you were to classify all variables of one type, which would it be?
4. Is it possible for a system of variables to have no *endogenous variables*? Why or why not?
5. How is the confirmatory factor analysis model a special case of the wider *structural equation model*?
6. What is the *null hypothesis* for a structural equation model? What constitutes a rejection of the null? How is this similar or different from traditional null hypothesis significance testing?
7. What is the influence of  $q$  in the *root mean square residual*? As  $q$  increases or decreases, does this have any effect on the value of RMR?
8. Compare the *Tucker-Lewis index* to that of *Bollen's*  $\rho_1$ . Which would you consider to be a better measure of model fit? Why?
9. Compare the *nonlinear* structural equation model to that of the *semiparametric* structural equation model. How are they similar? Different?
10. Building on the example in the chapter, run the same SEM model but where  $Z$  is indicated by only  $z1$  and  $z2$ . How do your results compare to the results in the chapter? Discuss.

# 14 Cluster Analysis and Data Segmentation

Cluster analysis, also known as *data segmentation*, is a general name for a variety of methods and analyses designed to discover, *in some sense*, groups of cases that, again, *in some sense*, “go together.” Perhaps the most significant challenge to cluster analysis is in defining what exactly *is* a cluster. Indeed, unpacking the phrase “in some sense” is very much the goal of this chapter. As noted by Izenman (2008), determining the exact nature of a cluster “is a difficult question to answer mainly because there is no universally accepted definition of exactly what constitutes a cluster. As a result, the various clustering methods usually do not produce identical or even similar solutions” (p. 408). Hence, as we will see, there is no universal single cluster solution when it comes to cluster analysis. Rather, there are a variety of algorithms that will potentially generate disparate results depending on the algorithm used in performing the clustering and the data on which that algorithm is interacting.

A good clustering algorithm, in general, should intuitively be one for which pairwise distances between cases are smaller for cases *within* the same cluster than for cases *between* clusters. More formally, cluster analysis attempts to discover multiple convex regions (i.e., convex meaning *circular* or *sphere-like*) of the variable space that essentially contain modes of the probability distribution of that variable space (Hastie et al., 2009). One can also seek to “cluster the clusters” such that the clusters that are most alike are classified into a larger “super cluster.” At first glance, the establishment of clusters may seem to resemble very much techniques such as PCA. However, in PCA the number of generated dimensions could never be more than the number of original dimensions (i.e., variables) subjected to the PCA. And while in cluster analysis one cannot typically generate more clusters than there are cases, the dimensionality of the new “clustered” space can actually be more numerous than in the original variables (Alpaydin, 2020, p. 178).

## 14.1 Cluster Paradigms and Classifications

There are a variety of cluster classification paradigms that attempt to literally “cluster” the types of clustering algorithms. One rudimentary distinction often made is that between *top-down* vs. *bottom-up* approaches. In the top-down approach, one might designate the number of clusters one wishes to “extract” from the data or allow an algorithm to find smaller clusters from a much larger initial one. In the bottom-up approach, the algorithm begins with individual cases and builds up clusters by fusing case memberships, usually in a hierarchical manner. As we will see, the classic *k*-means clustering algorithm is one such example of a top-down approach (Hastie et al., 2009, p. 503) while agglomerative clustering is an example of a bottom-up method to clustering.

Following Hastie et al. (2009), we can also more generally classify clustering algorithms into one of the following three types:

*Combinatorial algorithms.* These are applied directly to the data without reference to an underlying probability model. These are sometimes known as *hard* clustering algorithms, because cases are deterministically allocated to cluster membership. That is, there is no probability associated with a case being allocated to a given cluster, or, if we prefer to be all-probability-inclusive about it, cases are allocated with probability equal to 1.0; that is, the case either belongs in the cluster or it does not. This is what we generally mean by *hard clustering*. Combinatorial algorithms can also be interpreted as *encoders* in which the encoder is the  $k$ th cluster in an analysis, to which any given observation in the data is assigned (Hastie et al., 2009).

*Mixture modeling* assumes an underlying probability model and is thus part of the family of *probability-based clustering*. Mixture models have a similar goal as combinatorial algorithms but use association rules to construct simple descriptions (also known as *conjunctive rules*) that describe regions of high density. Mixture modeling is an example of *soft clustering*, where a case is not deterministically guaranteed to belong to a given cluster. Rather, it belongs to a given cluster with an associated *probability*. Mixture models are especially useful when working with binary data. More specifically, in mixture models we assume observed data arose or was originated from a mixture of distributions, that is, the data were “generated” based on those distributions. For this reason, not surprisingly, mixture models are often regarded as *generative*. The choice of distribution is often Gaussian, but is essentially variable, and allows the user to incorporate prior knowledge of the cluster distributions. In this sense, a *prior probability* idea is utilized in clustering. In such probabilistic mixture models, as opposed to non-probabilistic models, the log-likelihood can be used as a metric to determine the success of clustering (Murray, 2022), though this approach will not necessarily be effective in entirely evaluating the clustering that is discovered by the model.

*Mode seekers* or “bump hunters” are nonparametric approaches to clustering and seek to estimate distinct modes of a probability density function. Classification of cases to respective clusters is made by assignment of cases to the cluster for which the observation is the closest. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a popular method of density-based clustering, one which we will survey in this chapter.

The above are not the sole categorizations possible for clustering algorithms. For instance, Xu and Tian (2015) provide *nine* different categories by which clustering algorithms can be differentiated and distinguished. For instance, traditional *k-means* and *k-medoids* are generally referred to as algorithms based on *partition* due to their way of segregating data. This is in contrast to algorithms such as DBSCAN which is in its own clustering category of those based on *densities* (i.e., the category of *density-based clustering algorithms*). For our purposes in this chapter, we stop short of classifying algorithms too finely and are generally content with the above aforementioned classification.

This chapter features a survey and discussion of the more popular methods of clustering, which include *k-means*, *hierarchical clustering*, *density-based clustering* (DBSCAN), and *cluster mixture models*.

## 14.2 Are Clusters Meaningful?

Cluster analysis, whatever the algorithm, will typically always “find” clusters since they are designed to do just that. However, whether those clusters are meaningful in any sense other than *mathematically* is often left to the user to ultimately determine. Unlike supervised methods that typically feature a loss function to quantify the extent to which predictions are in error, unsupervised methods, of which cluster analysis is one, do not generally feature such “hard” measures to assess how “far off” one is from an ideal solution. This is especially so if natural groupings are

unknown before carrying out the cluster analysis, which is in most cases why we are employing cluster analysis in the first place. In evaluating cluster solutions, as Hastie et al. (2009, p. 487) remark,

One must resort to heuristic arguments not only for motivating the algorithms, as is often the case in supervised learning as well, but also for judgments as to the quality of the results. This uncomfortable situation has led to heavy proliferation of proposed results since *effectiveness* [emphasis added] is a matter of opinion and cannot be verified directly.

Indeed, Venables and Ripley (2002, p. 316) issue a rather dire warning on relying on cluster analysis exclusively in discovering groupings:

Do not assume that “clustering” methods are the best way to discover interesting groupings in the data; in our experience the visualization methods are often far more effective. There are many different clustering methods, often giving different answers, and so the danger of over-interpretation is high.

Perhaps the take-away message from the above warnings is that, as is the case with so many statistical models, to be aware that *what you find is often a function of what tool you use to do the finding*. There is no such thing as the definitive “existence” of clusters. Rather, as was the case with previously studied dimension reduction techniques in this book, the “existence” of a dimension or cluster solution will be intimately tied to the algorithm that “discovered” the solution. Naturally, this reality has huge philosophical implications and raises many questions. For instance, that one cannot make immediate sense of a cluster solution should not necessarily imply the solution is not sensible. All we can really conclude is that it currently is not sensible to the user *at this point in history, given the context and current state of the science in which he or she practices*. Indeed, this dilemma makes for an interesting philosophical debate regarding whether the *existence* of cluster solutions is primarily *mathematical* or whether it is primarily *substantive*. It is not a debate we will have here but is an important one in the *philosophy of statistics and mathematics* and is something the user should keep in mind as we survey a variety of cluster algorithms.

### 14.3 Dissimilarity Metrics: The Basis of Clustering Algorithms

Most clustering centers around the idea of a *dissimilarity metric*. For some types of clustering, dissimilarity or similarity is defined explicitly, while for other clustering approaches such as mixture modeling, a distance metric is usually implied from the way the clustering is performed. If the data are *qualitative* rather than *quantitative*, then mathematical distance cannot be computed and some measure of similarity will be required in its place (Xu & Tian, 2015).

In mathematics, a dissimilarity coefficient  $d$  may be symmetric, meaning  $d(A, B) = d(B, A)$ , as well as being non-negative, that is,  $d(A, A) = 0$ . We can also easily deduce from a dissimilarity metric that for points  $A, B, C$ , where  $B$  lies between  $A$  and  $C$ ,

$$d(A, C) \leq d(A, B) + d(B, C) \quad (14.1)$$

We can also express  $d(A, B)$  as an *ultrametric*,

$$d(A, B) \leq \max(d(A, C), d(B, C)) \quad (14.2)$$

where  $A$ ,  $B$ ,  $C$  can also be conceived as clusters with points existing within each cluster. The ultrametric holds that the distance between clusters  $A$  and  $B$  is less than or equal to the maximum distance between points in clusters  $A$  and  $C$  or between points in clusters  $B$  and  $C$ . Venables and Ripley (2002) note that hierarchical approaches to cluster analysis can be largely interpreted as ultimately approximating an ultrametric dissimilarity. Indeed, Simovici and Hua (2019) have found that a dissimilarity space (which empirically, is a set of data) is more clusterable the closer such space is to an ultrametric.

#### 14.4 Association Rules: Market Basket Analysis

Association rules are a relatively primitive way of clustering. The idea behind *association rule* analysis, also commonly referred to as *market basket analysis* (since it is commonly used in marketing analysis of products) is to, not surprisingly, find associations between variables, typically binary variables (Hastie et al., 2009). Much of association rule analysis reduces down to the familiar idea of *conditional probabilities*, that is, the probability  $p(y|x)$  where  $y$  is what is being purchased but conditional on product  $x$ . For example, we may want to know the probability of a customer purchasing a computer monitor given that that same customer has just purchased a desktop computer or the probability of purchasing a television stand given one has just purchased a new television. Many of the algorithms for association analysis reduce to finding modes or what are called “bumps” in the data, whereby collections of prototype  $x$  values are associated with probability densities (i.e., the “bumps”).

More formally, association analysis seeks to find *intersections* of subsets of variables on  $x$ , that is, overlapping spaces of the kind

$$P\left[\bigcap_{j=1}^p (x_j \in s_j)\right]$$

where the intersection of such subsets is known as a *conjunctive rule* (Hastie et al., 2009). These subsets naturally occur for qualitative variables, where for quantitative variables the subsets  $s_j$  consist of intervals along some continuum.

Clustered association rules have also been proposed that seek to essentially group association rules themselves into categories instead of operating by several individual rules (Lent et al., 1997). Association rules have their modern origin with Agrawal et al. (1993), who provide a highly readable and highly recommended introduction to the topic. The package *arules* (Hahsler et al., 2024) in R can be used for mining association rules. For further details on market analysis, see Aggarwal (2015).

#### 14.5 Why Not Consider All Groups?

The first question that arises in the study of cluster analysis *algorithms* is that of why not look at *all groups* of cases, and then decide which is optimal by some metric? For example, when considering the possible clusters of two objects from three available objects, it would be foolish to not simply consider all possibilities of the simple combinatorial problem *3 choose 2*, equal to 3. Indeed, it would at first glance seem that using a combinatorial algorithm from the combinatorial paradigm of clustering is the obvious choice, that is, simply enumerate all possibilities and choose the solution that maximizes or minimizes some objective function. That is, choose the “best” cluster in this sense, and be done with the task.

While the above solution is highly recommended for very small data sets, it unfortunately will be ineffective or impractical for any data set of even moderate size. The reason for this is that the number of ways of partitioning  $n$  cases into  $g$  clusters is literally gargantuan in number, and can be shown to be equal to

$$N(n, g) = \frac{1}{g!} \sum_{k=1}^g \binom{g}{k} (-1)^{g-k} k^n \quad (14.3)$$

More succinctly, the number can also be approximated rather well by  $g^n / g!$ . This approach is sometimes referred to as the *sledgehammer* approach. Not only is the approach impractical from a substantive point of view, it is absolutely prohibitive from a computational vantage point as well since to evaluate this vast number of combinations would require a huge amount of computational time and capacity. While smaller data sets will never experience these upper limits of capability, increasingly large data sets will, and given the vast amounts of data that can be clustered today in the era of big data, computational capacity becomes a determining and influential factor when deciding on how to cluster. A good algorithm isn't only one that clusters well; it must also be *computationally efficient*.

Hence, what is needed are *ways* to group cases together in some sense that is efficient without searching through every single possible combinatorial pattern. That is, we need to consider or hypothesize a much smaller set of *enumeration possibilities*. This is precisely why cluster analysis algorithms exist. Algorithms are designed to partition cases such that mathematically meaningful groups emerge which may or may not simultaneously be substantively meaningful (i.e., the user and research community can make “sense” of what the clusters might represent). Yet good algorithms do not consider all possible groupings; they only consider the *best* ones, in the sense of the most *efficient*, *creative*, and *greedy* ways to generate such groupings, where “greedy” implies that many algorithms consider what is best “here and now” and not two or three steps down the road. By way of analogy, the carrot cake with enough sugar to make you dizzy sounds *great* right now without any thought of how you will feel in a couple of hours.

#### 14.5.1 What Makes a “Good” Clustering Algorithm?

Some have suggested criteria for a good clustering algorithm. For instance, some have proposed that clustering algorithms should require minimal requirements of domain knowledge to determine input parameters, as appropriate values are typically not known in advance when working with large databases. A good cluster algorithm should also not automatically assume clusters will be *spherical* in shape, as empirical evidence has shown that many empirical clusters are *drawn out*, *linear*, or *elongated*, etc., which are shapes that a technique such as traditional clustering algorithms like  $k$ -means may fail to detect. This last point is very important in cluster analysis, as an entire literature in statistics and machine learning exists with the goal of discovering *non-spherical* cluster shapes. A good clustering algorithm, as already emphasized, should also provide reasonable *efficiency* on large databases. This of course is coupled with computational concerns and the limitations of software and hardware to carry out such clustering.

Whatever the clustering algorithm adopted, many clustering procedures, as is the case with most statistical and machine learning methods, essentially come down to *maximizing* or *minimizing* some quantity, that is, some objective function of the data. That is, good clustering, from a mathematical point of view, becomes simply a problem of *optimization*. What makes for a good optimization strategy for clustering? It seems reasonable to require that *within-cluster variation* be minimized. That is, an ideal goal to strive for might be to minimize the sum of *within-cluster variation* across each cluster,



$$\text{MIN} \sum_{k=1}^K \text{Within}(C_k)$$

where  $C_k$  is a given cluster  $k$ . The above reads the sum from  $k = 1$  to  $K$  of within-cluster variation is minimized, that is, as small as it can be through successive iterations. This amounts to the clusters being as *homogeneous* as possible.

More specifically, the within-cluster distance is usually specified by an average squared distance function:

$$\text{Within}(C) = \sum_{k=1}^K N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2 \quad (14.4)$$

where  $\bar{x}_k$  is the mean for the  $k^{\text{th}}$  cluster, and  $\sum_{k=1}^K N_k$  simply represents the sum across all clusters  $k$  to  $K$ . That is, we are summing across all clusters  $k$  the squared distances from respective cluster means,  $\bar{x}_k$ . The objective is to minimize this average *within-cluster* distance. Once this objective is minimized, the clustering algorithm is said to have *converged* on a solution. As we will discuss later in this chapter, the  $k$ -means clustering algorithm seeks to accomplish just this and provides a *local optimum* (James et al., 2021).

*Cluster analysis algorithms do not typically consider all partitions of the data since such are too numerous and would be far too labor intensive. Combinatorially, there are simply too many ways of generating groups. Instead, they seek to partition cases in a way that is mathematically meaningful which usually translates to them optimizing some “greedy” function of the data.*

## 14.6 Distance and Proximity Metrics

As already mentioned, virtually all of cluster analysis techniques available, despite their individual differences, are based on the notions of *similarity* or *distance* between objects. Indeed, as discussed at the outset of this book, one can even say that most, if not all statistical and learning machine methods eventually boil down in some capacity or another to notions of *distance* when inspected at the most foundational level (Johnson & Wichern, 2007). Distances will generally be organized into a *distance matrix* that contains non-negative entries. That is, it is assumed that distances are a *non-negative integer*. To measure proximity of cases, we require a measure of distance, where the distance  $d_j$  between  $x_{ij}$  and  $x_{i',j}$  is, most generally,

$$d_j(x_{ij}, x_{i',j}) = (x_{ij} - x_{i',j})$$

and squared distances are thus given by

$$d_j^2(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$$

There are several such distance measures available. The most common is that of *Euclidean distance*. The Euclidean distance between two  $p$ -dimensional objects,  $\mathbf{x} = [x_1, x_2, \dots, x_p]$  and  $\mathbf{y} = [y_1, y_2, \dots, y_p]$  is given by,

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2} \\ &= \sqrt{(\mathbf{x} - \mathbf{y})'(\mathbf{x} - \mathbf{y})} \end{aligned} \quad (14.5)$$

A *statistical distance* can also be used, where distances  $\mathbf{x} - \mathbf{y}$  are scaled by the covariance matrix,  $\mathbf{S}$ ,

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})' \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})} \quad (14.6)$$

Other metrics are also available. For instance, the *Minkowski metric* is given by

$$d(\mathbf{x}, \mathbf{y}) = \left[ \sum_{i=1}^p |x_i - y_i|^m \right]^{1/m} \quad (14.7)$$

for which when  $m = 2$ , the Minkowski metric reduces to the Euclidean distance,

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= \left[ \sum_{i=1}^p |x_i - y_i|^m \right]^{1/m} \\ &= \left[ \sum_{i=1}^p |x_i - y_i|^2 \right]^{1/2} \\ &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2} \end{aligned}$$

Yet another distance metric is that of *city-block*, also known as *Manhattan distance*,

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^p |x_i - y_i| \quad (14.8)$$

One place where the Manhattan distance is used is in the  $k$ -medians approach to cluster analysis (Aggarwal, 2015) which we will feature shortly when we survey algorithms from the  $k$ -representatives family.

The importance of choosing an appropriate distance metric cannot be overstated. As emphasized by Hastie et al. (2009, p. 506):

Specifying an appropriate dissimilarity measure is far more important in obtaining success with clustering than choice of clustering algorithm. This aspect of the problem is emphasized less in the clustering literature than the algorithms themselves, since it depends on domain knowledge specifics and is less amenable to general research.

A good summary of the more popular distance measures used in clustering algorithms is provided in Xu and Tian (2015).

### 14.7 Is the Data Clusterable?

Before engaging in clustering, we might ask the question of how “clusterable” is a data set to begin with? A first answer to this might be that it does not matter and to just give clustering a “try” and see what comes out. However, this strategy can be somewhat problematic. Cluster analysis in most cases *will* find a solution, the algorithms are designed specifically to do just that! Hence, a test for clusterability *before* engaging in cluster analysis, if that test is a good one, should give us an idea of whether the data should be clustered in the first place and whether any clusters we do observe will be more a function of the algorithm than the actual empirical existence of clusters. As remarked by Adolfsson et al. (2019, p. 1), “perhaps an even more fundamental issue than algorithm selection is when clustering should, or should not, be applied.”

The *Hopkins statistic* is one popular measure that can be used to assess the clusterability of data. It can be interpreted as a measure of *clustering tendency* of a data set (Aggarwal, 2015; Wright, 2022). In this approach,  $p_r$  values are randomly generated as well as  $p_d$  points randomly sampled from the data. For both sets of points, distances to their nearest neighbors in the data are computed. That is, if we let  $x_i$  be a nearest neighbor, then what we wish to compare are the distances  $p_r$  to  $x_i$ , which we will call  $D_{random}$  vs.  $p_d$  to  $x_i$ , which we will call  $D_{data}$ . That is, we are interested in knowing how the distances to nearest neighbors compare based on a *random data set* vs. *actual data points* in the data set.

The Hopkins statistic is thus defined as

$$H = \frac{\sum_{i=1}^r D_{i(random)}}{\sum_{i=1}^r (D_{i(data)} + D_{i(random)})} \quad (14.9)$$

For uniform randomly generated data, the expectation for Hopkins is in the neighborhood of 0.5. This is as a result of  $D_{i(data)}$  and  $D_{i(random)}$  being similar. If the data are clusterable,  $D_{i(data)}$  is typically less than  $D_{i(random)}$  and hence  $H$  will increase. Values of  $H$  greater than 0.75 have been found to equate to clustering “confidence” at 90% (Lawson & Jurs, 1990; Wright, 2022). Adolfsson et al. (2019) provide an extensive review of so-called clusterability methods (many other techniques exist over and above Hopkins), and have found that the Hopkins statistic may be a somewhat limited measure when clusters touch or overlap.

The package *hopkins* (Wright, 2023) is available in R for computing the Hopkins statistic. As an example from the package, we compute a Hopkins statistic on the iris data (sepal length through petal width, excluding species), where we compute Hopkins on  $m = 15$  observations (or sampled rows of the data, the default is 1/10th of the rows if not set specifically):

```
> library(hopkins)
> hopkins(iris[, 1:4], m=15)
[1] 0.9986529
```

We can also obtain a  $p$ -value for the computed Hopkins statistic where  $x$  is the value of the computed Hopkins and  $n$  is the number of observations sampled:

```
> hopkins.pval(x = 0.9986529, n = 15)
[1] 0
```

As a result of the value for Hopkins being so large, the  $p$ -value is extremely small (it is rounded to 0,  $p$ -values *never* equal 0 *exactly*). Suppose instead we obtained a Hopkins equal to 0.5, the  $p$ -value would thus be

```
> hopkins.pval(x = 0.5, n = 15)
[1] 1
```

The value is equal to 1 since recall the expectation of Hopkins for a uniform distribution is equal to 0.5. The further away from 0.5, the lower the  $p$ -value, as demonstrated once more,

```
> hopkins.pval(0.1, 15)
[1] 3.925226e-08
> hopkins.pval(0.9, 15)
[1] 3.925226e-08
```

We see that for both values of 0.1 and 0.9, identical  $p$ -values are obtained. As usual with a significance test, as  $n$  increases, all else equal,  $p$ -values drop. Using our previous example but increasing  $n$  to 100 not surprisingly yields

```
> hopkins.pval(0.9, 100)
[1] 0
```

## 14.8 Algorithms for Cluster Analysis

In this section we survey some of the more popular clustering algorithms, which include  $k$ -means and associated  $k$ -representatives, hierarchical or agglomerative clustering, density-based clustering (specifically DBSCAN), and clustering via mixture models.

### 14.8.1 K-Means Clustering and K-Representatives

The  $k$ -means clustering algorithm is probably the simplest and well-known, most used form of cluster analysis.  $K$ -means is a special case of the wider group of *representative-based* algorithms for clustering that rely on the concept of *distance* between observations as the basis for clustering. These approaches are also sometimes called *partitional* (Madhulatha, 2012) or *unnested* methods since they seek to partition the data into groups or clusters by assigning cases nearest cluster centroids whereas hierarchical clustering is considered a form of *nested* clustering that builds clusters by essentially fusing smaller subordinate clusters. The centroids in this case are the so-called “representatives” of the given clusters. Other representative-based algorithms include  $k$ -medians,  $k$ -medoids, and *kernel-based  $k$ -means*. Briefly, the  $k$ -medians approach uses the median instead of the mean and is thus more robust to extreme and outlying cases than is the  $k$ -means approach. In the  $k$ -medoids approach, initial starting points to initiate the algorithm are drawn from the original database, and in kernel-based  $k$ -means, a kernel function via the familiar *kernel trick* is applied in generating clusters.

As mentioned, representative-based algorithms are usually contrasted to *hierarchical approaches* which take an either *bottom-up* or *top-down* approach to building clusters and generally require the storage of a distance matrix. Nonhierarchical clustering methods are generally

more efficient than hierarchical methods since they do not require the storing of a distance matrix. The jargon can differ slightly depending on the source one reads, so it behooves the reader investigating clustering algorithms to be sure of how terms are being used by the author.

#### 14.8.2 How K-Means Works

$K$ -means clustering attempts to partition cases into distinct clusters where the number of clusters is requested by the user beforehand. In  $k$ -means, the sum of squared distances to their closest representatives is optimized and an observation belonging to one cluster cannot simultaneously belong to another. That is, cluster membership is *mutually exclusive*. This will typically not be the case in some other clustering methods such as density-based or mixture modeling clustering. So-called *fuzzy clustering* algorithms also allow for a case to be clustered into more than a single cluster.

The  $k$ -means algorithm also clusters every object, implying that there will not be any observations left out of the clustering. In this sense,  $k$ -means is regarded as a *complete clustering* algorithm such that it seeks to cluster *every* case. This is in contrast to *partial clustering* algorithms that allow some cases to go unclustered. For example, DBSCAN, a density-based clustering algorithm which we will survey shortly, typically does not produce complete clustering since it defines some cases as simply *noise* points rather than classifying them to clusters. These “noise” points are those that do not seem to “fit” in any of the clusters being generated. And while  $k$ -means is often regarded as an “algorithm” today, there is nothing special about the word “algorithm” as it applies to the procedure. Indeed, in the past,  $k$ -means was often called a data-analytic *procedure* or *method*. Like many statistical and machine learning approaches today, it is often referred to as an “algorithm” largely, though not *exclusively*, due to the zeitgeist that we currently find ourselves in. Otherwise excellent works on cluster analysis may not once even mention the word “algorithm” and may simply describe the process by which clusters are derived.

*K-means is a special case of the wider group of representative-based clustering algorithms. Hierarchical algorithms on the other hand can be divided into bottom-up and top-down approaches.*

#### 14.8.3 Defining Proximity for K-Means

Recall that there are a host of distance metrics and proximities available for cluster analysis, including that for  $k$ -means. The most typical choice of distance metric in  $k$ -means is that of the *squared Euclidean metric*,

$$\begin{aligned} d(x_i, x_i') &= \sum_{j=1}^p (x_{ij} - x_{i',j})^2 \\ &= \|x_i - x_i'\|^2 \end{aligned} \tag{14.10}$$

such that the quantity being minimized is that of the average within-case or within-point scatter variation discussed earlier:

$$Within(C) = \sum_{k=1}^K N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2$$

We follow James et al. (2021) quite closely in detailing precisely how the  $k$ -means algorithm goes about clustering:

1. Each observation in the data receives a random number 1 to  $K$ . These are the *initial* cluster assignments.
2. For each of the  $K$  clusters, the cluster *centroid* is computed, where “centroid” is another word for *arithmetic mean* in this case. More generally, a centroid in mathematics is simply a *center point*, and thus it is, at least in theory, open and flexible regarding how we wish to define it. For example, the *median* can also be regarded as a centroid in cluster analysis. In classical  $k$ -means, the arithmetic mean is featured as the centroid.
3. Assign each observation to the cluster whose centroid is closest. The measure of “closeness” is defined by the squared Euclidean metric or some other distance function we have chosen *a priori*. There is nothing preventing the user from trying out different distance metrics on the same clustering problem and study the patterns that result.
4. Iterate on steps 2 and 3 until the minimization criterion is achieved, which as mentioned earlier is typically  $Within(C)$ , the within-cluster sums of squares.

The algorithm is guaranteed to decrease the value of the objective function at each step and optimizes a *non-convex* function (Murphy, 2022), which recall implies that achieving a global minimum is more difficult in  $k$ -means since numerous local minima exist in its place, and hence the algorithm will typically converge to local minima. As a result of the non-convexity, how the algorithm is initialized can make a big difference in the final cluster solution. For this reason, choosing several *random starts* is often advised to learn if the various cluster solutions more or less converge on a single solution (Murphy, 2022) or whether any of the single cluster solutions obtained are favorable to the user. Another approach is successively selecting points that are more distant from cluster centroids thus ensuring that initial seeds are well separated. This should be done with caution, however, as to not exclusively sample outliers.

#### 14.8.4 *Setting $k$ in K-Means*

A key decision in  $k$ -means clustering is that of deciding how to set  $k$ . One could set it by trial and error to learn what kinds of clusters emerge and decide whether they are substantively meaningful, but it would be advantageous to have some statistical “objective” criteria to help us determine an optimal value for  $k$ . We could at first attempt to set it such that it minimizes the reconstruction error. The problem with this approach is that reconstruction error will naturally decrease with increasing  $k$ , and hence setting this as a criterion to minimize will be meaningless as we could always minimize the error further by increasing the number of clusters desired, somewhat akin to forever increasing  $R^2$  due to increasingly adding more predictors to the model. Since we know that  $R^2$  must virtually always increase with increasing numbers of predictors, an inflating  $R^2$  is a poor measure of anything, which recall is why  $R^2_{adj}$  or some other penalized statistic (e.g., AIC) is typically recommended over, or at least in conjunction with,  $R^2$ . This situation occurs also in probabilistic mixture models that we will survey later in which we can compute the log marginal likelihood of the solution as an aid to determining an “ideal” number of clusters such that whichever cluster solution yields the largest likelihood may be considered “best.” In this context, however, it will be

entirely possible for a solution with fewer clusters to generate a more favorable substantive finding than a solution with a greater number of clusters.

#### 14.8.5 Weakness of K-Means

K-means comes with it a host of weaknesses. As noted by Müller and Guido (2017):

Even if you know the “right” number of clusters for a given dataset, k-means might not always be able to recover them. Each cluster is defined solely by its center, which means that each cluster is a convex shape. As a result of this, k-means can only capture relatively simple shapes. K-means also assumes that all clusters have the same “diameter” in some sense; it always draws the boundary between clusters to be exactly in the middle between the cluster centers. (p. 175)

This implies that *k*-means clustering essentially comes with it a prior idea of what the clusters *should* look like to some extent, and if the empirical realities do not conform to that idea, then recovering them can be quite difficult or even impossible. *K*-means implicitly also assumes a *spherical shape* for each cluster, and as we will see when it comes to mixture modeling, the EM algorithm is capable of discovering a broader class of *elliptical clusters*. Once again, this is a general problem when it comes to clustering algorithms. To reiterate, *what you find often depends on what tool you use to do the searching*. As warned by Meila and Zhang (2023, p. 26) with regard to manifold learning more generally, “Are the features observed really in the data or artifacts of the algorithm?” Add to this difficulty that in most applications of cluster analysis there is typically no *ground truth*, as it is sometimes called, to the data. That is, we do not usually know the labels before we do the clustering. Only in toy problems do we typically have this benefit (e.g., species membership on the iris data).

#### 14.8.6 K-Means vs. ANOVA vs. Discriminant Analysis

We said above that the goal of *k*-means was to minimize within-cluster difference. Does this criterion sound familiar? This minimization of such an objective function may have caught the eye of the reader and reminded them of what occurs in the analysis of variance or discriminant analysis. Consider the following conceptual comparison of approaches:

*ANOVA* – goal is to maximize between-group differences and minimize within-group differences.

*Discriminant analysis* – goal is essentially the same as in ANOVA, except in reverse roles of the independent and dependent variables such as to maximize group separation.

*K-means cluster analysis* – goal is to minimize within-group differences and maximize separation between clusters.

To be clear, ANOVA, discriminant analysis, and *k*-means clustering are *not* the same procedures and have their own individual respective roles in data science. However, they often seek to minimize a similar objective function and can be, to some extent, likened to one another. Indeed, one way to think about *k*-means clustering, where there is no *a priori* ground truth, is such that we seek to *discover* groups. In classic ANOVA or discriminant analysis, the investigator comes to the analysis already with a theory in mind in the form of either a factor with various levels in ANOVA or a response variable in discriminant analysis featuring a binary or categorical grouping. That is, in ANOVA and discriminant analysis the user advances the hypothesized grouping structure. Is it the *best* grouping structure in some sense? Not necessarily, but it is the one hypothesized by the



researcher. In cluster analysis, we typically do not yet have a hypothesized grouping structure in mind and seek to “find one” via the algorithm. This is a powerful conceptual way to relate these approaches, even if their technical details will differ substantially.

#### 14.8.7 *Making K-Means Probabilistic via K-Means ++*

We have seen that  $k$ -means assigns cases that are closest to cluster centers but have noted that initial cluster seeds can have a significant impact and influence on the determination of clusters. Given this impact, we have said that it is usually recommended to try a variety of initial seeds and not simply rely on a single “run” of seeding. But how exactly should this be done? We could simply rerun the  $k$ -means algorithm manually a few times and settle with that approach to determining initial seeds. While this is not a terrible way of doing things, such an approach is not very efficient and could take a long time to execute.

A much more efficient approach to the above problem is to extend the traditional  $k$ -means algorithm and adopt a method in which the initial seeds are selected *at random* as in classical  $k$ -means, but then subsequent points are selected subject to being probabilistically *proportional* to the square distance to the point’s nearest cluster center (Murray, 2022, p. 719). The approach here is to try to, essentially, “cover” the data as much as possible. This approach is called *farthest point clustering*, or  $k$ -means ++.

#### 14.8.8 *Using the Data: K-Medoids Clustering*

We have seen that  $k$ -means clustering assigns cases to the cluster for which it is closest to the cluster centroid. In  $k$ -means, that “representative” centroid is that of the *arithmetic mean* while in  $k$ -medians clustering it is the *median*. Note that both of these approaches use a representative that is a *function* of the data and not the data itself. That is, the mean is a summary statistic of the data; it is not an *actual* data point in the data. In this sense, the mean does not “exist” in the data, it is simply an abstract measure we apply to the data set. Even if the mean happens to equal a particular data point, it is still not being drawn from the data itself, and likewise with the median; the median may or may not exist in the data. The bottom line is that whether we are using the mean or median, we are nonetheless using a *function of the data* and not the data itself.

A variant of such approaches is that of  $k$ -medoids. In  $k$ -medoids clustering each cluster center is estimated not through computing a *function* of the data, but instead by choosing an *actual* empirical data value from the actual data set that is “optimal” in some sense. How should we choose this value? One strategy is to select it such that *its average dissimilarity to all other points in the data set is a minimum*. The empirical value, or *representative object*, that is, the *medoid*, may also be chosen randomly. The key advantage of  $k$ -medoids, in addition to being less sensitive to outliers than  $k$ -means, is that it can be applied to spaces that are non-Euclidean which do not necessarily permit the computation of an average (Leskovec et al., 2014). In such cases, a representative can be calculated called a *clustroid*, of which there are many ways of choosing the relevant clustroid for a given problem (see Leskovec et al., 2014 for details).

The algorithm typically used in  $k$ -medoids is called the *partitioning around medoids* (or *PAM*) algorithm. Essentially, the algorithm exchanges medoids for non-medoids in an iterative fashion recomputing the cost each time to learn whether the exchange was advantageous in minimizing the cost. If the exchange was worthwhile, it is kept. An allied method to PAM is the *Voronoi iteration*. See Murphy (2022) for details.



### 14.8.9 K-Means in Python

We conduct a simple *k-means* cluster analysis in Python on the *iris* data using *seaborn*:

```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
iris = sns.load_dataset("iris")
print(iris.head())
```

|   | sepal_length | sepal_width | petal_length | petal_width | species |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |

```
df = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
df.head(3)
Out[132]:
```

|   | sepal_length | sepal_width | petal_length | petal_width |
|---|--------------|-------------|--------------|-------------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         |

The function `df.head(3)` serves to list the first three cases. By default, `df.head()` will yield the first five cases.

We now perform the cluster analysis, requesting three clusters:

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(df)
kmeans.predict(df)
Out[25]:
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

The above are the results of the classification for each case. We can obtain cluster centroids as follows:

```
centroids = kmeans.cluster_centers_
centroids
Out [33]:
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

We now obtain predicted species classification (Figure 14.1):

```
kpredict = kmeans.predict(df)
kpredict

Out [56]:
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])

plt.scatter(iris['petal_length'], iris['petal_width'], c =
kpredict, cmap = 'cool')

plt.scatter(iris['sepal_length'], iris['sepal_width'], c =
kpredict, cmap = 'cool')
```

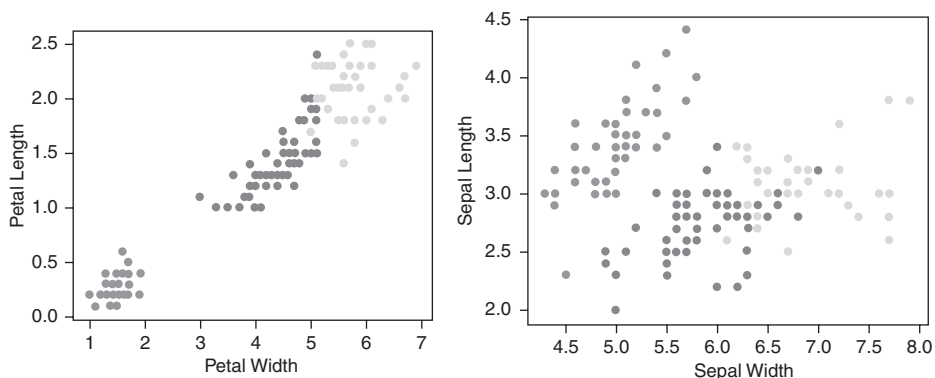


Figure 14.1 Predicted species classification on the iris data for petal length by petal width (left) and sepal length by sepal width (right).

We can produce cluster summaries by cluster number:

```
cluster = kmeans.labels_
cluster1 = iris.loc[cluster == 0]
cluster2 = iris.loc[cluster == 1]
cluster3 = iris.loc[cluster == 2]
```

We demonstrate only for cluster 1 (we print only a few cases):

```
cluster1
Out [66]:
```

|     | sepal_length | sepal_width | petal_length | petal_width | species    |
|-----|--------------|-------------|--------------|-------------|------------|
| 52  | 6.9          | 3.1         | 4.9          | 1.5         | versicolor |
| 77  | 6.7          | 3.0         | 5.0          | 1.7         | versicolor |
| 100 | 6.3          | 3.3         | 6.0          | 2.5         | virginica  |
| 102 | 7.1          | 3.0         | 5.9          | 2.1         | virginica  |
| 103 | 6.3          | 2.9         | 5.6          | 1.8         | virginica  |
| 104 | 6.5          | 3.0         | 5.8          | 2.2         | virginica  |
| 105 | 7.6          | 3.0         | 6.6          | 2.1         | virginica  |

```
cluster1.describe()
Out [65]:
```

|       | sepal_length | sepal_width | petal_length | petal_width |
|-------|--------------|-------------|--------------|-------------|
| count | 38.000000    | 38.000000   | 38.000000    | 38.000000   |
| mean  | 6.850000     | 3.073684    | 5.742105     | 2.071053    |
| std   | 0.494155     | 0.290092    | 0.488590     | 0.279872    |
| min   | 6.100000     | 2.500000    | 4.900000     | 1.400000    |
| 25%   | 6.425000     | 2.925000    | 5.425000     | 1.825000    |
| 50%   | 6.700000     | 3.000000    | 5.650000     | 2.100000    |
| 75%   | 7.200000     | 3.200000    | 6.000000     | 2.300000    |
| max   | 7.900000     | 3.800000    | 6.900000     | 2.500000    |

*Sklearn* can also be used to perform basic to more advanced cluster analyses. For details, see *scikit-learn.org*.

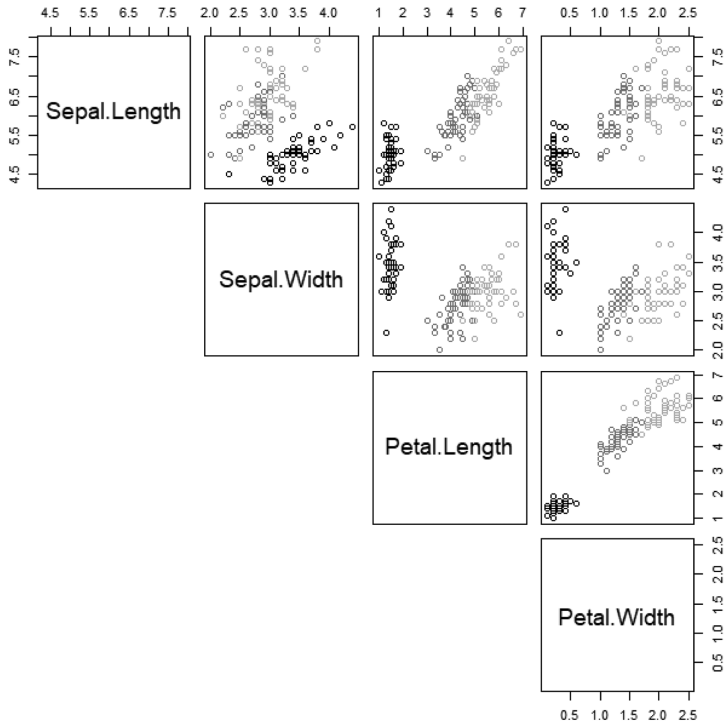
#### 14.8.10 K-Means in R

The function `kmeans` can be used for  $k$ -means clustering in R. As an example, we once more cluster the iris data:

```
> library(car)
> attach(iris)
> iris.data <- data.frame(Sepal.Length, Sepal.Width, Petal.Length, Petal.
Width)
> some(iris.data)
```

|     | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|-----|--------------|-------------|--------------|-------------|
| 6   | 5.4          | 3.9         | 1.7          | 0.4         |
| 34  | 5.5          | 4.2         | 1.4          | 0.2         |
| 41  | 5.0          | 3.5         | 1.3          | 0.3         |
| 48  | 4.6          | 3.2         | 1.4          | 0.2         |
| 50  | 5.0          | 3.3         | 1.4          | 0.2         |
| 98  | 6.2          | 2.9         | 4.3          | 1.3         |
| 116 | 6.4          | 3.2         | 5.3          | 2.3         |
| 122 | 5.6          | 2.8         | 4.9          | 2.0         |
| 133 | 6.4          | 2.8         | 5.6          | 2.2         |
| 146 | 6.7          | 3.0         | 5.2          | 2.3         |

```
> iris.2 <- iris[,-5]
> species <- iris[, 5]
> pairs(iris.data, col = species,
+ lower.panel = NULL)
```



```
> set.seed(20)
> k.means.fit <- kmeans(iris.data, 3, nstart=20)
> k.means.fit
K-means clustering with three clusters of sizes 50, 62, 38
Cluster means:
```

|   | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|---|--------------|-------------|--------------|-------------|
| 1 | 5.006000     | 3.428000    | 1.462000     | 0.246000    |
| 2 | 5.901613     | 2.748387    | 4.393548     | 1.433871    |
| 3 | 6.850000     | 3.073684    | 5.742105     | 2.071053    |

The clustering vector below represents the clustering assignments. For instance, the first case in the data was assigned to the first cluster. By simply viewing the clustering vector, we can see that the first 50 cases are homogeneous, such that they are assigned to the first cluster. Contrast that to the third cluster in which we see a mix of “2” and “3” cluster assignments:

```
Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[31] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2
[61] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2
[91] 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 2 3 3 3 3 2 2 3 3 3 2
[121] 3 2 3 2 3 3 2 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2 3 3
Within-cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
( between SS / total SS = 88.4 %)
```

The above *within-cluster sum of squares by cluster* can be used as a potentially useful measure of cluster quality, such that smaller values suggest the given cluster is more “condensed” than do larger values. Likewise, *between\_SS/total\_SS* can be used to help validate the cluster solution, where larger values are preferable to smaller values. We discuss more ways of validating cluster solutions shortly.

What follows is a classification table, so-called *confusion matrix*, based on the cluster solution, where we can see, for example, that all 50 cases were correctly assigned (*setosa*), 48 cases of *versicolor* were assigned to cluster 2 with two cases being assigned to cluster 3. For those cases *virginica*, 14 were assigned to the second cluster while 36 were assigned to the third:

```
> table(species, k.means.fit$cluster)
Species      1   2   3
Setosa       50   0   0
Versicolor   0  48   2
Virginica    0  14  36
```

*Cluster purity* is a measure of how well the algorithm classifies relative to the ground truth, if available (Sikhakolli & Sikhakolli, 2023). If the algorithm performed perfectly, then out of 150 cases total, 150 should have been clustered correctly. If we let  $p_j$  be the proportion representing the maximum number of points clustered correctly into the given class, and  $\frac{N_i}{N}$  be the total fraction of cases eligible to be clustered for the given class from the total number of cases  $N$ , then cluster purity is given by

$$\text{cluster purity} = \sum_i \frac{N_i}{N} p_j \quad (14.11)$$

For the iris data, if all cases were clustered correctly, the cluster purity would equal  $(50 + 50 + 50)/150 = 1.0$  as a result of there being 50 cases per species of iris. For our actual data, the cluster purity is computed to be  $(50 + 48 + 36)/150 = 0.893$ . This is a shorthand calculation since  $N_i / N = 50/150$  for each class of species. To be more precise, the full computation is the following:

```
> purity <- (50/150)*(50/50) + (50/150)*(48/50) + (50/150)*(36/50)
> purity
[1] 0.8933333
```

The package *funtimes* (Lyubchich et al., 2023) computes cluster purity as well as reproducing results initially observed in the aforementioned confusion matrix:

```
> library(funtimes)
> purity(species, k.means.fit$cluster)
$pur
[1] 0.8933333
$out
```

|   | ClassLabels | ClusterLabels | ClusterSize |
|---|-------------|---------------|-------------|
| 1 | setosa      | 1             | 50          |
| 2 | versicolor  | 2             | 48          |
| 3 | virginica   | 3             | 36          |

We see that the measure computed above (0.89) matches our manual computation. If the ground truth is known before the clustering, then cluster purity in this example is simple to conceptualize, it is essentially assessing the total proportion of cases that were clustered correctly into their ground truth classifications. For instance, if *setosa* had 50 cases, and all of those cases were clustered into “*setosa*,” then such would constitute perfect cluster purity for *setosa*. The cluster purity for *versicolor* is not quite as good because out of the 50 cases that *should* have gone into *versicolor*, only 48 ended up classified into that class. *Virginica* had the worst cluster purity at only 36 out of 50. The overall cluster purity is effectively summing up these class-specific purity ratios.

We can also plot the within sums of squares for the optimal number of clusters (code from Boehmke & Greenwell, 2020, p. 409) as shown in Figure 14.2. The function `fviz_nbclust()` helps visualize and determine the optimal number of clusters, where `get_dist` computes a distance function, in this case based on the correlation method of spearman (other options include `pearson` and `kendall`, `minkowski`, etc.); see the package *factoextra* (Kassambara & Mundt, 2017, for details):

```
> library(factoextra)
> fviz_nbclust(iris.data,
+ kmeans,
+ k.max = 3,
+ method = "wss",
+ diss = get_dist(iris.data, method = "spearman"))
```

We can see from the plot that the ideal number of clusters is 2 or 3, with 3 generating the lowest within sums of squares (`method = "wss"`):

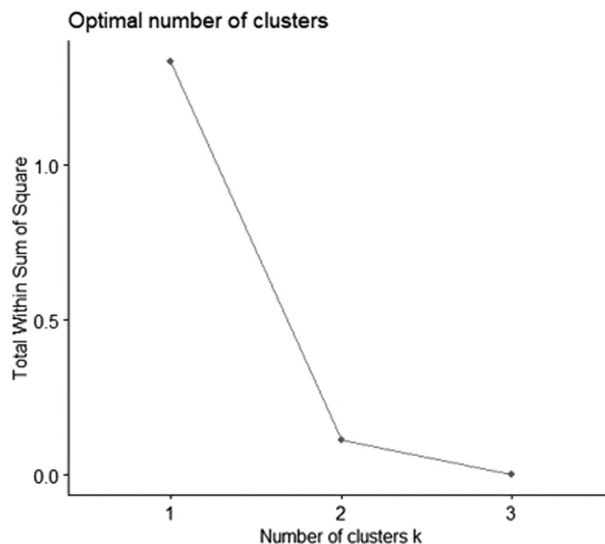


Figure 14.2 Within sums of squares as a function of number of clusters in helping determine the optimal number of clusters to keep.

## 14.9 Sparse and Longitudinal *K*-Means

There are numerous variations of *k*-means clustering. For instance, in *sparse cluster analysis*, permutations of the data can be obtained such that the features in the permuted data set are uncorrelated

with each other. In R, sparse cluster analysis can be performed with `KMeansSparseCluster()`. Longitudinal  $k$ -means is also a possibility where clusters can be tracked over successive time points. The *kml* package in R (Genolini et al., 2024) can perform  $k$ -means on longitudinal data. The package *longitudinalData* (Genolini et al., 2024) is also useful, as *kml* relies on it.

### 14.10 Hierarchical Clustering

As we have seen, in the  $k$ -family or  $k$ -representatives of clustering the number of clusters is specified *a priori*. Methods such as  $k$ -means and their variants rely on distance matrices to perform their clustering. An alternative approach to clustering is *hierarchical clustering*, in which the number of clusters is not requested *a priori*, but instead the algorithm will build a tree, a *dendrogram*, that displays the historical linkages made as the algorithm progresses from single cases to fused conglomerates of cases subject to some minimization criterion. A variety of linkages can be used that define case similarity, which we will survey shortly. Hierarchical clustering can either be agglomerative or divisive. *Agglomerative clustering* begins with single cases and fuses cases together building up the dendrogram. *Divisive clustering* begins with one big cluster and separates into smaller clusters at each step. We only survey the agglomerative case here as it is the most common.

Figure 14.3 is an example of a simple dendrogram where we can easily visualize the history of where cases were joined from the bottom up in the dendrogram. We see that cases 2 and 3 were first fused, followed by the fusing of those cases to case 4. Lastly, this cluster made up of cases 2, 3, and 4 was fused to case 1 to form the largest cluster.

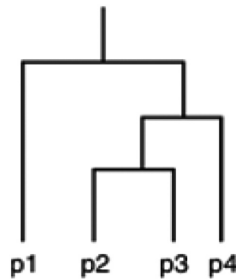


Figure 14.3 Example of a cluster analysis dendrogram, revealing the history of linkages from the bottom up.

As mentioned, there are competing linkages one can use in fusing cases and clusters, including *complete*, *single*, *average*, and *centroid* methods, each having its strengths and weaknesses. In single linkage (also known in the literature as *MIN*), the minimum distance between cases is used as a criterion for fusing, while in complete linkage (also known in the literature as *MAX*) that of the maximal furthest distance between cases is employed. Average linkage, not surprisingly, takes an average of distances between groups. Dendrograms in single and complete linkage are invariant under monotone transformations of pairwise dissimilarities, however this is not true for average linkage (Izenman, 2008). These are not the only linkages possible but are the most common ones.

For a good discussion of linkages used in hierarchical clustering, see Johnson and Wichern (2007). The key point for now is to recognize that employing different linkages could lead to different solutions.

Hierarchical clustering can be expensive computationally because it operates on what are potentially quite large dissimilarity indexes. The slower speed of hierarchical clustering can be considered a drawback of this approach and of agglomerative approaches more generally as they

can be quite expensive since they require entire distance matrices as inputs. Another drawback to hierarchical approaches is that contrary to  $k$ -representative clustering, there is usually no clear single optimization criteria or objective that is optimized in the final solution (Murphy, 2022). Further, once cases are fused, they cannot be *unfused*. Rather, the user typically has to mostly rely on substantive theory in determining whether a cluster solution is successful.

#### 14.10.1 Agglomerative Clustering in Python

We demonstrate a simple agglomerative clustering routine in Python on the iris data:

```
import seaborn as sns
iris = sns.load_dataset("iris")

df = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
iris.drop(['species'], axis=1, inplace=True)
iris
```

We produce a series of scatterplots between variables (Figure 14.4).

```
import seaborn as sns
sns.pairplot(iris)
```

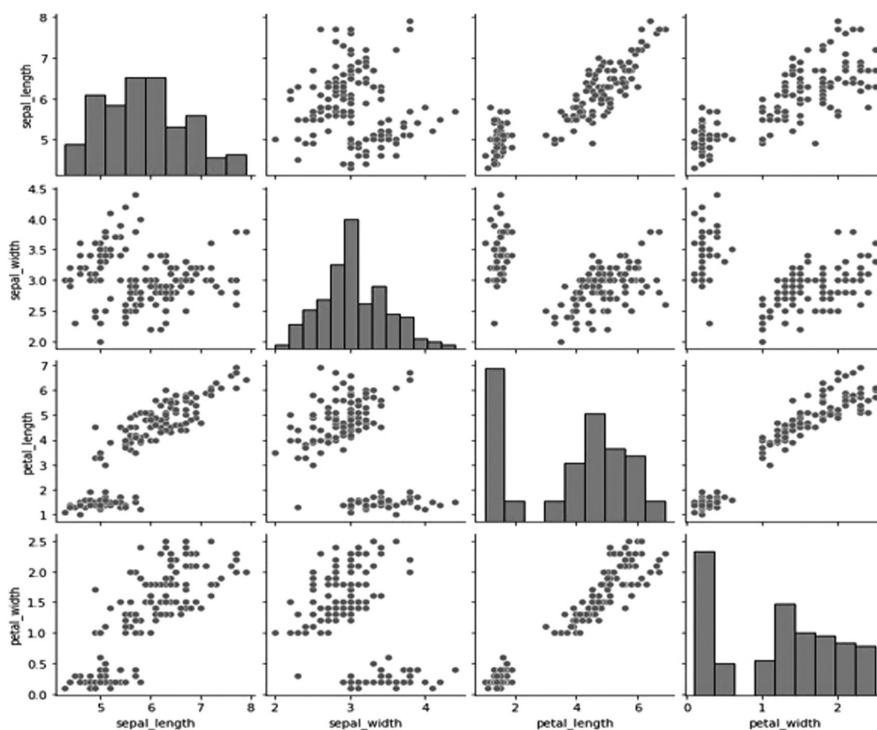


*Figure 14.4* Bivariate scatterplots on iris data features. Such a plot helps to visualize tentative bivariate groupings in the data, and while not a cluster analysis in itself, may still be useful on a descriptive level in anticipating cluster separation. Given *a priori* knowledge of species, it becomes readily clear that *setosa* already separates itself well from *virginica* and *versicolor*.

We now perform a single-linkage cluster analysis, using Euclidean distances:





```
d <- dist(iris.data, method = "euclidean")
clust.single <- hclust(d, method = "single")
clust.average <- hclust(d, method = "average")
plot(clust.single)
plot(clust.average)
```

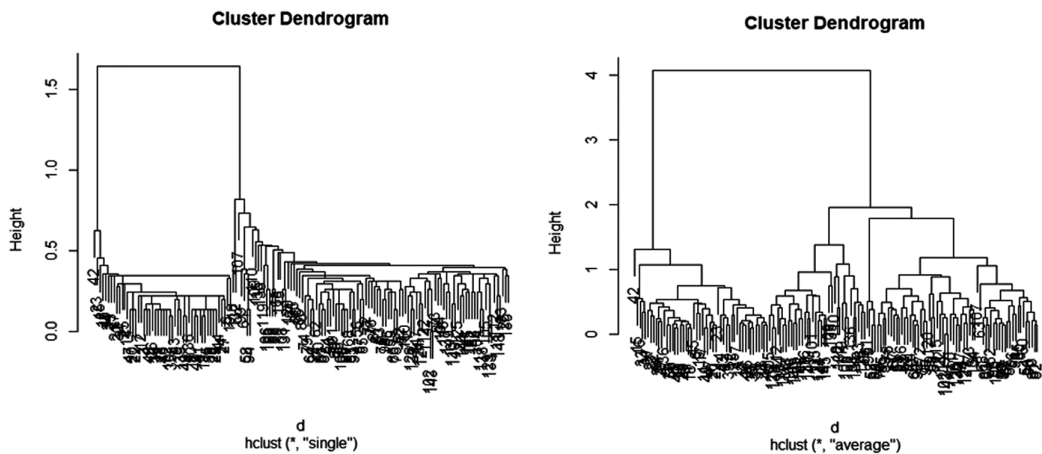


Figure 14.6 Dendrograms based on single (left) and average (right) linkage.

We can also produce a variety of plots in addition to the dendrograms in Figure 14.6. For example, *triangle*, *unrooted*, and *fan plots* are possibilities (Figure 14.7) using the *ape* package (Paradis & Schliep, 2019).

```
> library(ape)
> dendro <- as.dendrogram(clust.single)
> plot(dendro, type = "triangle", ylab = "Height")
> plot(as.phylo(clust.single), type = "unrooted", cex = 0.6, no.margin = TRUE)
> plot(as.phylo(clust.single), type = "fan")
```

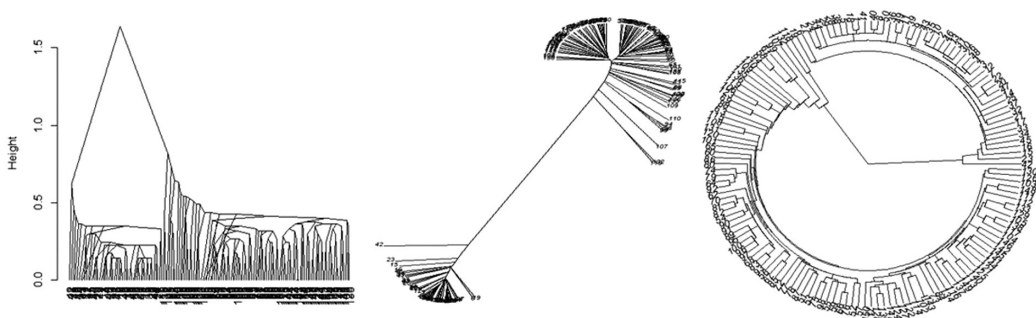


Figure 14.7 Triangle (left), unrooted (middle), and fan plot (right) on cluster data.

Of course, the plots in Figure 14.7 are challenging to visualize due to the size of the data, but they are enough to get a sense of the general idea that there are two larger clusters that come out of the data. To produce a colored dendrogram, consult the package *sparcl* (Witten & Tibshirani, 2018).

In helping determine the number of clusters to keep, we can produce what is called a *silhouette plot* as well as compute the *gap statistic* (Figure 14.8), where larger values of these statistics are indicative of a preferred solution:

```
> library(factoextra)
> p1 <- fviz_nbclust(iris.data, FUN = hcut, method = "silhouette", k.max = 10) +
+ ggtitle("(B) Silhouette method")
> p1
```

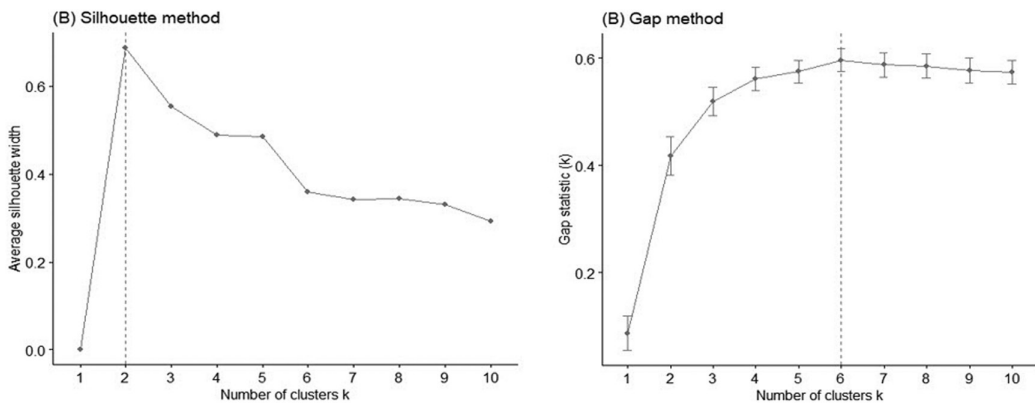


Figure 14.8 Silhouette plot (left) and gap statistic plot (right) suggesting the ideal number of clusters to be between approximately two (silhouette) and six (gap).

We can see that the ideal number of clusters is equal to 2 for the silhouette method, which more or less agrees with the dendrogram on this data. Silhouette values are computed for each data point and then averaged per cluster. A greater than not value for average silhouette values indicates that data points are essentially “matched” well for the given cluster, and hence cluster solutions with a higher than not silhouette average value are preferred.

```
> p1 <- fviz_nbclust(iris.data, FUN = hcut, method = "gap", k.max = 10) +
+ ggtitle("(B) Gap method")
```

The number of clusters suggested by the gap statistic is equal to 6 (however, gap values are relatively strong for anything greater than three or so clusters). Gap values correspond to the difference between within-cluster dispersion relative to a reference or null distribution. For details on how the gap statistic is calculated, see Tibshirani et al. (2001). For further details on the silhouette method, see Rousseeuw (1987).

### 14.11 Density-Based Clustering: DBSCAN

We have thus far in this chapter surveyed  $k$ -means and  $k$ -representative clustering approaches as well as hierarchical and agglomerative paradigms. These are perhaps the most traditional and common ways of clustering across the sciences.

A third general approach to clustering which is especially useful for detecting clusters of arbitrary shape goes by the name of *density-based clustering*. One such approach is the popular DBSCAN algorithm, which is an acronym for *density-based spatial clustering of applications with noise*. DBSCAN is especially efficient for large databases. The method is similar to what are referred to as *grid-based methods* (Aggarwal, 2015), but uses circular regions as building blocks for the clusters. To understand DBSCAN, we must first understand the notions of *dense points* and *crowded regions*.

#### 14.11.1 Dense Points and Crowded Regions

DBSCAN operates differently than  $k$ -representative clustering or hierarchical clustering. Instead of building up clusters by establishing random seeds and then computing distance functions, DBSCAN identifies points that are in so-called *crowded regions* of the feature space, that is, where many points are in some sense already “close together.” These regions are sometimes referred to as *dense regions*. Hence, we already see a key difference between this approach and that of  $k$ -means. In  $k$ -means, the algorithm did not seek out dense spaces of any kind but instead simply selected random points to begin the clustering process. In DBSCAN clustering, we actually seek out what appear to be clusters *first* as defined by the density of those regions. But how exactly to define density? For this we need to implement a distance function.

The density of a data point in DBSCAN is defined by how many points lie within a given *radius*  $\varepsilon$  of the point (Aggarwal, 2015). All points that lie within a radius of  $\varepsilon$  are considered to be in the same cluster as the *core point* so long as they are equal to or less than *MinPts*, another parameter set by the user specifying the minimum number of points (or “neighbors”) required to form the so-called “dense region” within radius  $\varepsilon$ . Points that are within a dense region are called *core samples* or *core points*. Other points are either *border* or *noise points*. Border points are those that are contained within the radius  $\varepsilon$  but have insufficient points around them by the criterion *MinPts* to consider themselves a core point. That is, border points fall within the neighborhood of core points but are not significant enough to be core points themselves because they do not have *MinPts* points around them. *Noise* points are those that are neither core points nor border points.

The algorithm essentially connects core points that are within  $\varepsilon$  (or “*eps*”) of one another, which is a flexible parameter that can be freely adjusted by the user. A choice for  $\varepsilon$  too large and what will result is a cluster with density equal to the number of cases. On the other hand, for  $\varepsilon$  set too small will have virtually all points essentially being their own cluster. Border points are then assigned to the cluster for which they have the highest *connectivity*, meaning that it is not sufficient on its own to be a core point (due to not meeting the *MinPts* criterion), so is classified to the cluster for which it is closest to a core point. The groups that result are the clusters and the remainder of the points are noise points. The precise steps of the algorithm are detailed in Schubert et al. (2017, p. 3).

The DBSCAN algorithm can actually be viewed as a more robust generalization of the single-linkage method where instead of only detecting spherical clusters as in  $k$ -means, it can find arbitrarily shaped clusters (Aggarwal, 2015). DBSCAN can also be considered a modification of *single-linkage agglomerative clustering* by treating border and noise points with special consideration relative to other methods where they are not given any special treatment. That is, in classic

single-linkage clustering, for instance, border and noise points have no meaning in such an algorithm, and hence are just more points around which to compute linkages. As noted by Aggarwal (2015), the benefit of DBSCAN over single linkage is that with the consideration of border and noise points, the sensitivity of outliers is diminished in addition to the algorithm being able to detect clusters of arbitrary shape rather than only spherical ones such as in  $k$ -means approaches. Recall that we said from the outset that one of the chief difficulties with clustering algorithms is that they often make prior assumptions about the shape of clusters (e.g., spherical in the case of  $k$ -means). DBSCAN is useful because it is not limited to detecting spherical or other pre-defined shapes. DBSCAN has also been found to have better cluster classification when used on smaller data sets.

#### 14.11.2 DBSCAN in R

We feature an example of DBSCAN in R from the library *dbscan* (Hahsler et al., 2024) on the iris data:

```
> library(dbscan)
> data(iris)
> x <- as.matrix(iris[, 1:4])
> x
```

|       | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|-------|--------------|-------------|--------------|-------------|
| [1,]  | 5.1          | 3.5         | 1.4          | 0.2         |
| [2,]  | 4.9          | 3.0         | 1.4          | 0.2         |
| [3,]  | 4.7          | 3.2         | 1.3          | 0.2         |
| [4,]  | 4.6          | 3.1         | 1.5          | 0.2         |
| [5,]  | 5.0          | 3.6         | 1.4          | 0.2         |
| [6,]  | 5.4          | 3.9         | 1.7          | 0.4         |
| [7,]  | 4.6          | 3.4         | 1.4          | 0.3         |
| [8,]  | 5.0          | 3.4         | 1.5          | 0.2         |
| [9,]  | 4.4          | 2.9         | 1.4          | 0.2         |
| [10,] | 4.9          | 3.1         | 1.5          | 0.1         |

For this analysis, we will choose a value for `eps` of 0.4 and `minPts` of 4. We plot the cluster solution below:

```
> db <- dbscan(x, eps = 0.4, minPts = 4)
> db
DBSCAN clustering for 150 objects.
Parameters: eps = 0.4, minPts = 4
Using euclidean distances and borderpoints = TRUE
The clustering contains four cluster(s) and 25 noise points.
```

```
0  1  2  3  4
25 47 38 36 4
```

Available fields: `cluster`, `eps`, `minPts`, `dist`, `borderPoints`

The numbers of points per cluster are reported above as 47 in cluster 1, 38 in cluster 2, 36 in cluster 3, and 4 in cluster 4. There are 25 noise points (cluster 0). We plot the result in Figure 14.9, which reveals the clusters, where black points (may not be visible in text) are noise points (Hahsler et al., 2024, p. 30):

```
> pairs(x, col = db$cluster + 1L)
```

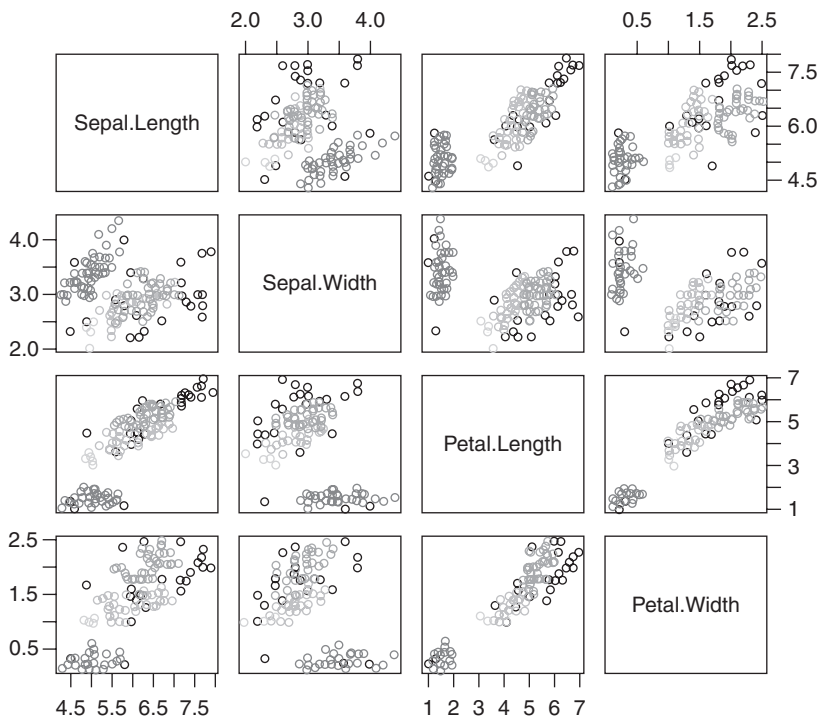


Figure 14.9 Pairwise scatterplots of iris features after DBSCAN clustering revealing cluster separation.

How best to set *eps*? We can produce a *distance plot* to help evaluate the suitability of *eps* for a fixed value of *minPts*:

```
> iris <- as.matrix(iris[, 1:4])
> kNNDistplot(iris, minPts = 5)
```

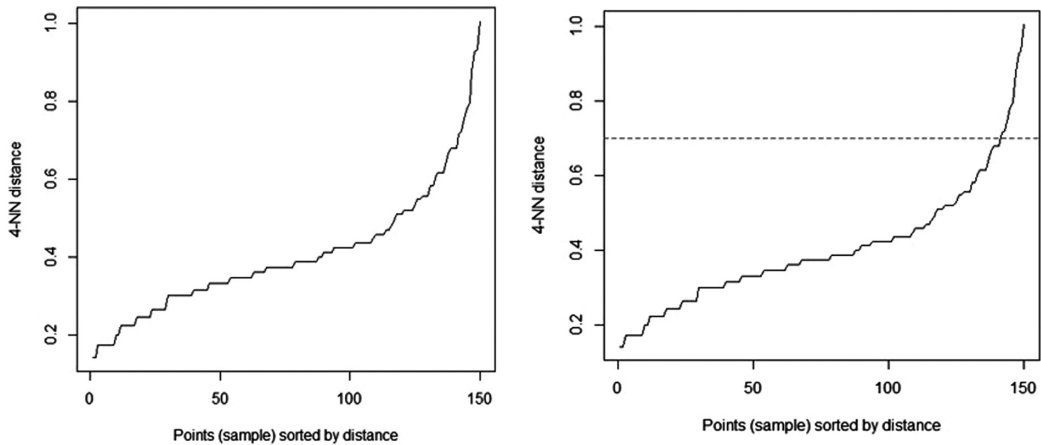


Figure 14.10 Nearest-neighbor distance plot (left), revealing distance as a function of points sorted by distance, where 4-NN ( $k$ -NN, where  $k = \text{minPts} - 1$ ) is the distance from a point to its  $k$ -nearest neighbor. In the right plot is a horizontal line at approximately the “knee” in the plot where points to the right may be considered outliers or “noise” points. Hence,  $\text{eps} = 0.7$  may be suitable.

The *nearest-neighbor distance plot* sorts all data points by their corresponding  $k$ -nearest neighbor distances, where here  $k$  is equal to 4 (Figure 14.10, left). A sudden increase in the distance, a so-called *knee* in the plot suggests that most points to the right are likely outliers, and hence choosing a value for  $\text{eps}$  at the location of such a knee is recommended (Hahsler et al., 2024, p. 6). According to Hahsler et al., points to the right of approximately 0.7 seem to suggest the location of the knee and thus points to the right may be primarily regarded as noise. Hence, a suitable value for  $\text{eps}$  would be 0.7. A horizontal line can be drawn at 0.7 by coding (Figure 14.10, right):

```
> abline(h=.7, col = "red", lty = 2)
```

## 14.12 Clustering via Mixture Models

The clustering algorithms we have surveyed thus far, those of  $k$ -means (and  $k$ -representatives), hierarchical clustering as well as density-based approaches such as DBSCAN, have made no assumptions about where the data have been sampled. That is, they have not made any *parametric assumptions* about the data or *prior assumptions* about the nature of the clusters (aside from assuming they must be spherical in shape in  $k$ -means or in dense regions as in DBSCAN), nor have they invoked anything to do with probability. They have simply relied on the empirical data to essentially tell the story of whether or not clusters might exist and have proceeded as such through hard-rule algorithmic processes of assigning cases to clusters.

An alternative approach to these clustering approaches is to assume the data arise from a *mixture* of (typically) Gaussian distributions or other parametric distributions (e.g., Poisson for discrete data, etc.). These distributions, both kind and number, are often chosen by the user based on prior substantive experience in the area and prior knowledge or hypotheses about the mixture components, though in most cases where specific knowledge is not known, Gaussian distributions are usually the default. The number of components, however, can be regarded as a hyperparameter

that the user must specify (Alpaydin, 2020). If the number of clusters is known *a priori* and hence the “ground truth” of the number of clusters exists, specifying the number of components is easy. However, and as usual, cluster analysis is performed when knowledge of the number of clusters is not known *a priori*, in which case prior knowledge of what the user might *expect* to see may be useful in designating the number of components.

A *mixture model*, as the name suggests and as we have seen already in other contexts, is a mix of probability density distributions which for cluster analysis has the general form,

$$f(\mathbf{x}; \theta) = \sum_{g=1}^G \pi_g f_g(\mathbf{x}; \theta_g) \quad (14.12)$$

where there are  $g = 1$  to  $G$  clusters,  $\pi_g$  is the probability of an observation arising from the  $g$ th cluster ( $\pi_g$  represents the *mixing proportions*),  $f_g(\mathbf{x}; \theta_g)$  is the probability density function in the  $g$ th cluster, and  $\theta_g$  is a vector of parameters that requires estimation. The usual form to assume for  $f_g(\mathbf{x}; \theta_g)$  is that of multivariate normality, implying that  $\theta_g$  will therefore consist of a mean vector and covariance matrix (Jolliffe, 2002).

Once the mixture model is fit, posterior probabilities  $p(y/\theta)$  are obtained. The most popular mixture model, as mentioned, is that of a *Gaussian mixture model* (Murray, 2022) with density  $N(y|\mu_g, \Sigma_g)$  where now  $N(y|\mu_g, \Sigma_g)$  denotes a multivariate normal distribution with mean vector  $\mu_g$  and covariance matrix  $\Sigma_g$ . A more robust version here would utilize the *t*-distribution instead of the normal (McLachlan & Chang, 2004).

The statistical objective in a *mixture model* is to estimate the parameters of the model by maximizing the *likelihood function* of the data to generate posterior probabilities of cluster membership with respect to the data, given knowledge of the model parameters. A problem can occur when maximizing the likelihood in a mixture of Gaussians, however, which is the problem of potential *singularities*. A singularity can occur if one of the components of the mixture model has a mean equal to that of a single data point. As noted by Bishop (2006), under this condition, a single data point contributes a single term to the likelihood function,

$$N(x_i | x_i, \sigma_j^2 \mathbf{I}) = \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j} \quad (14.13)$$

where  $\sigma_j^2$  is the variance of the  $j^{\text{th}}$  component and  $x_i$  is a given data point, which, as a consequence of the singularity, is equal to the mean of the given component, that of  $\mu_j$ . The problem occurs as  $\sigma_j$  gets closer and closer to 0, and in the limit,

$$\lim_{\sigma_j \rightarrow 0} \left( \frac{1}{(2\pi)^{1/2}} \frac{1}{\sigma_j} \right) = \infty$$

and hence the log-likelihood will go to infinity due to  $1/\sigma_j = 1/0$ . In cases where this does occur, a random value for the mean can be “reset” to help avoid the “collapsing” of a point on the likelihood in this way. For a more thorough description of this problem, see Bishop (2006).

As mentioned, the user can select from different choices of model distributions and parameters which have the effect of inducing *flexibility* with regard to the probability distribution for each component. The advantage of a mixture modeling approach is that resulting clusters will tend to



have high peaks resulting from relatively tightly and distinct cluster centers and hence the mixture model can account for cluster data relatively well in many cases.

Mixture models are part of the wider family of probability-based clustering algorithms which feature the idea of *soft clustering*, where a case is assigned to a cluster not *deterministically* but instead, *probabilistically*. These models are often referred to as *generative models* such that they assume observations were *generated* from one or more probability distributions. The reader may recognize “Bayesian” tones in this idea. More technically, the underlying idea behind a generative model is that one can generate synthetic data by drawing values from a marginal distribution (Bishop, 2006).

A popular method for the estimation of these probability-based models is that of maximum likelihood (ML) using the well-known EM algorithm. Recall that the EM algorithm is a general optimization routine typically employed for the treatment of *incomplete data*, but has been found to be especially valuable for fitting mixture models in cluster analysis, consisting of the *E-step* and *M-step* in its iteration. The *E-step* (*expectation*) and *M-step* (*maximization*) are iteratively executed to convergence to determine optimal parameters, where in the end posterior probabilities are estimated and a probabilistic model is obtained with *soft assignments* in place of *hard assignments* of cases to clusters. It turns out that special cases of the EM algorithm are equivalent to a soft version of the *k*-means algorithm, where the representative distances are used to define soft EM assignment probabilities (Aggarwal, 2015).

#### 14.12.1 Model Selection for Clustering Solutions

At first glance, model selection in a cluster analysis mixture model environment might be performed by simply maximizing the marginal likelihood, or more specifically, its logarithm, and choosing the cluster solution for which the log-likelihood is the greatest for  $g_1, g_2, \dots, G$  solutions. However, formal tests for this are difficult to come by (McLachlan & Chang, 2004). A formal test of  $H_0 : g = g_0$  vs.  $H_1 : g > g_0$  (where  $g_0$  is the number of clusters under a null and alternative, respectively) is possible via the bootstrap, where bootstrap samples are drawn from a given fitted mixture model for a specific  $H_0$  of  $g = g_0$  components and the value of  $-2 \log \lambda$  is computed both for  $H_0$  and for  $H_1 : g > g_0$ . A *p*-value can then be obtained to learn whether adding another component is, in a statistical sense, worthwhile.

The likelihood we seek can also be approximated by the following BIC statistic,

$$BIC(g) = \log p(\mathbf{x} | g, \hat{\theta}_g) - \frac{v_g}{2} \log(n) \quad (14.14)$$

where  $\hat{\theta}_g$  is the maximum-likelihood estimate of  $\theta_g$ ,  $v_g$  is the number of estimated parameters associated with the model with  $g$  components (Baudry et al., 2010), and  $n$  is the sample size. Note that in this case, *greater values for BIC* are preferred over smaller values as the goal is to *maximize* the first part of the equation,  $\log p(\mathbf{x} | g, \hat{\theta}_g)$ . The penalty or correction factor is in the second term,  $\frac{v_g}{2} \log(n)$ , which includes the number of parameters  $v_g$  required to obtain the given maximum for given  $g$  components. To the extent that more parameters  $v_g$  are required over less,  $\frac{v_g}{2} \log(n)$  will be increasingly large and hence we are subtracting an increasingly larger number from  $\log p(\mathbf{x} | g, \hat{\theta}_g)$ . Hence, as usual with *BIC*, more parsimonious models are typically preferred. Cut-off criteria here are generally discouraged and what denotes a “good” clustering solution will depend on the research area and user judgment. Note as well that there are different versions of BIC and related model selection statistics for mixed models in cluster analysis. What we have presented here is

only one possibility. It is good practice to always verify how the given author or software is defining the particular model selection statistic featured before interpreting it.

The package *mclust* (Scrucca et al., 2023) can perform mixture modeling in R.

### 14.13 Cluster Validation

Having surveyed the most essential algorithms and approaches to cluster analysis which have included *k-means*, *hierarchical*, *density-based*, and *mixture models*, we close out this chapter with a brief discussion and recap of the *cluster validation problem* and revisit criteria for choosing the number of clusters.

In general, there are two types of validation one may perform on a cluster solution: 1) *internal validation criteria*, and 2) *external validation criteria*. The most preferred of these is, of course, external validation, since externally validating a cluster solution puts us back in touch with the empirical world akin to cross-validation of a model on test data helps us know how the model or algorithm will perform on data it has not yet seen. Internal validation approaches can be useful however when external validation criteria are not available or otherwise difficult to come by.

Internal methods of validation include *sums of square distances to centroids*, which is, as the name suggests, a measure of variability of cases from their corresponding centroids. A better than not cluster solution generally has a smaller sums of squares distance than a poorer solution. This measure can be used with *k-means* but not with density-based clustering such as DBSCAN. A ratio of the *intracluster* distance (distance *within*) to the *intercluster* distance (distance *between*) can also be computed (Aggarwal, 2015, p. 196) where smaller values than not are suggestive of a better solution. Other measures of internal validity include the *silhouette coefficient* as well as probabilistic measures. See Aggarwal (2015, pp. 196–197) for details. Tibshirani, Walther and Hastie (2001) have also proposed the aforementioned “gap” statistic which in some cases can be useful as well. The gap essentially compares total within-cluster variation for a given cluster solution (e.g., *k*-cluster solution in *k-means*) to what would be expected under a null distribution. An increasingly large gap is thus indicative of a more refined cluster solution.

For mixture models, as we have seen, log-likelihood statistics can be computed as well as BIC for helping determine the number of components to the mixture model. It should be noted that mixture models are of the few clustering methodologies that generally allow for significance tests to be performed since they are probabilistic. A device called the *clustergram* has also been proposed (see Fleischmann, 2023 for details).

External methods of cluster validation are most useful when knowledge of the actual clusters, sometimes called the *ground truth*, is known *a priori* or beforehand to performing the cluster analysis (such as in the case of clustering the iris data). The benefit of using external methods of validation over internal ones is that they are not biased by the particular algorithm doing the clustering (Aggarwal, 2015). To externally validate a cluster solution, something as simple as a *confusion matrix*, as the reader will recall from discriminant analysis, can be generated to learn how well the clustering solution “mimics” reality, or ground truth. While simple visualization of smaller confusion matrices may be sufficient, for especially larger confusion matrices measures of *cluster purity* can be computed as we did earlier when we clustered the iris data. As might be suspected, while these and other measures are useful, final decisions on the number of clusters usually come down to substantive interpretation and should be highly influenced by how pragmatic the cluster solution proves to be given the research or scientific context.

### 14.14 Cluster Analysis and Beyond

In this chapter we surveyed the most common and foundational techniques of cluster analysis and data segmentation. These have included techniques such as  $k$ -means and other  $k$ -representatives, hierarchical and agglomerative approaches, density-based methods such as DBSCAN and mixture models. The reader should be aware that the current chapter has literally only scratched the surface when it comes to the literature on cluster analysis. New clustering techniques are continuously being developed in the statistical and machine learning literatures. However, as is the case with other methods surveyed in this book, if the reader is strong on the foundations of the techniques surveyed here, they will be in a reasonable position to generalize that knowledge to future techniques as they come across them.

Where to go from here? The extensions of the methods surveyed are almost limitless. The  $k$ -means clustering approach in this chapter can be extended to the kernel  $k$ -means algorithm. The DENCLUE algorithm is based on kernel-density estimation (Aggarwal, 2015). Ensemble methods can also be used in choosing a cluster solution, since different methods may provide conflicting solutions but combining them may paint a clearer picture.  $K$ -means is used in vector quantization related to data compression, often used in the analysis of images (see Bishop, 2006, p. 428 for details).

Cluster analysis can also be applied to categorical data, but with more difficulty;  $k$ -modes clustering is one option for categorical analysis. A hierarchical algorithm that has been found useful for categorical data is that of *ROCK*, which stands for *Robust Clustering Algorithm for Categorical Attributes*. See Aggarwal (2015) for details. Semi-supervised methods of cluster analysis are also available.

Other options for cluster analysis include *spectral clustering* which features an eigen analysis of a similarity matrix (Aggarwal, 2015) as well as biclustering which involves clustering rows or columns of a data matrix. See Murray (2022, p. 731) for details.

### 14.15 Chapter Review Exercises

1. Define what is meant by a *combinatorial algorithm* vs. *mixture modeling* vs. *mode seekers* as they pertain to cluster analysis algorithms.
2. Leaning on the quote by Venables and Ripley in the chapter, argue *against* cluster analysis as a potential means for discovering groupings in data.
3. Explain in words what is meant by an *ultrametric* as it concerns distance.
4. Why might so-called “greedy” approaches to cluster analysis be sometimes preferred over non-greedy approaches? What is generally meant by a “greedy” approach?
5. Explain the basis for the *Hopkins statistic* and why clustering algorithms alone may be insufficient in determining whether cluster analysis can find (in some sense) clusters in data.
6. Explain the logic of  $k$ -means clustering.
7. Sketch on a piece of paper how  $k$ -means clustering works and why the choice of *initial seeds* can play a fundamental role in determining the final results of clusters. Is there any way to address this influence?
8. Perform a  $k$ -means cluster analysis in R on the iris data using all features except for *sepal length*. How does your solution compare to the solution in the chapter that utilized all features?
9. Define *cluster purity* for a cluster solution and give an example using fictional data.

10. Compare and contrast *k-means* or *k-representatives* clustering to that of *hierarchical* clustering.
11. Describe how *DBSCAN* goes about generating clusters and contrast this approach to that of *k-means*.
12. Perform *DBSCAN* on the iris data in Python on all features except *sepal length*, and compare your solution to that obtained in the chapter.
13. Compare *cluster mixture models* to other mixture models surveyed elsewhere in the book. What is the general “theme” behind a mixture model?

# 15 Artificial Neural Networks and Deep Learning

Throughout much of this book, we have surveyed and previewed neural networks while in the midst of discussing other techniques. This was done intentionally to illustrate and emphasize the extent to which other models could be understood as *related* or even as *special cases* of the more general neural network design. Indeed, this was one of the major objectives of this book, to “demystify” neural networks and other models by relating them to similar methods. As with any new statistical or machine learning model the reader may come across, it is helpful, in fact virtually *essential* in understanding them to relate them to other techniques. This is one reason why introductory books and courses on statistics and machine learning should emphasize *foundations* at the expense of surveying every method or algorithm under the sun. If students properly learn foundations, they are then in a good position to generalize and extend on that knowledge as needed later in their careers. Without a grounding in foundations, however, one risks not only misunderstanding additional methods, one also risks misusing and misinterpreting those same methods. *Foundations are always the key to a good education.*

In this chapter, we finally arrive at the full neural network and unpack it in its full generality. In our treatment, our hope is that the reader will appreciate more fully how neural networks are related to other statistical procedures but also why they have essentially revolutionized data analysis in the last 20–30 years. We begin the chapter with a brief survey of the motivation behind the rise of neural networks. We then introduce what a neural network actually *is*, emphasizing that even a simple linear regression can be regarded as a simple single-layer feedforward neural network. Based on this foundation, we then inspect more closely the nature of *activation functions*, a concept that is central to neural networks, surveying different types of activation functions and differentiating between *saturating* vs. *non-saturating* functions. We then generalize the feedforward network to the multilayer perceptron, which is the most common neural network in use today. Incidentally, in our discussion of neural networks we do not strongly distinguish between what is a *neural network* vs. what is a *perceptron*, and at times use the terms interchangeably. However, due to their history, the term *perceptron* is usually reserved for neural networks having discontinuous step-function nonlinearities as hidden units (Bishop, 2006) while classic neural networks are typically understood to have continuous nonlinearities in the hidden units. So long as this distinction is appreciated, there is no harm in using the terms interchangeably as authors sometimes do.

## 15.1 The Rise of Neural Networks: Original Motivation

Artificial neural networks are often what first comes to mind when we think of modern machine learning. The origins of machine learning as it pertains to artificial neural networks dates back to at least psychologist Rosenblatt and his work on the human nervous system, developing a machine for recognizing the letters of the alphabet. This “machine,” he called a *perceptron*. The

McCulloch-Pitts neuron was featured in the seminal publication “*A Logical Calculus of the Ideas Immanent in Nervous Activity*” in 1943, which is often considered to be one of the initial “seeds” of the artificial intelligence movement that is in full bloom today. Abraham (2002) details the historical origins of this important publication as well as the contributors to it. Beginning in the 1930s, the mathematical description of the brain was becoming an important topic and these pioneers were of the earliest to apply mathematics in developing such networks.

Drawing on Turing’s work at the time, which was itself groundbreaking, McCulloch and Pitts reasoned that the task of computation could be described *logically* in a finite number of steps, and that Turing’s “machine” could be perceived as a “logic machine” (Abraham, 2002). Of course, from today’s perspective, these developments are obvious as key contributions to the modern algorithm and discipline of machine learning more widely, but at the time these realizations were quite novel. As well summarized by Abraham (2002):

What distinguished the McCulloch-Pitts neural network from previous concepts of networks was its basis in axioms and definitions, and their use of such axioms to construct a logical calculus of the relations between neurons ... McCulloch’s and Pitts’s goal was to represent the functional relationships between neurons in terms of Boolean logic: to embody the mental function of reasoning in the actual physiology of the brain.

(pp. 19–20)

The classic McCulloch-Pitts neuron is drawn in Figure 15.1 where  $x_1 \dots x_p$  are inputs and where each input has a value of 0 indicating “off” or 1 indicating “on”;  $S = \sum_{j=1}^p x_j$  represents the sum of the inputs, a *total excitation* of the neuronal connections.

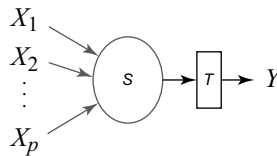


Figure 15.1 Classic McCulloch-Pitts neuron. Used with permission of Springer Nature BV, from *Modern Multivariate Statistical Techniques* by Alan J. Izenman, © 2008; permission conveyed through Copyright Clearance Center, Inc.

The total for  $S$  is then compared to a threshold value,  $T$ . If  $S \geq T$ , the output is 1 and the neuron fires, which means it transmits the signal to  $Y$ . Otherwise,  $Y$  is 0 and the neuron does not fire. This is known as a *threshold logic*, or in neurology, the *all-or-none principle* which was thought to govern how neurons behave. Mathematically, this can be represented using the *heavyside* function, which is nothing more than a step function having values “1” if the neuron fires and “0” otherwise, that is,

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (15.1)$$

which when plotted, looks like that in Figure 15.2.

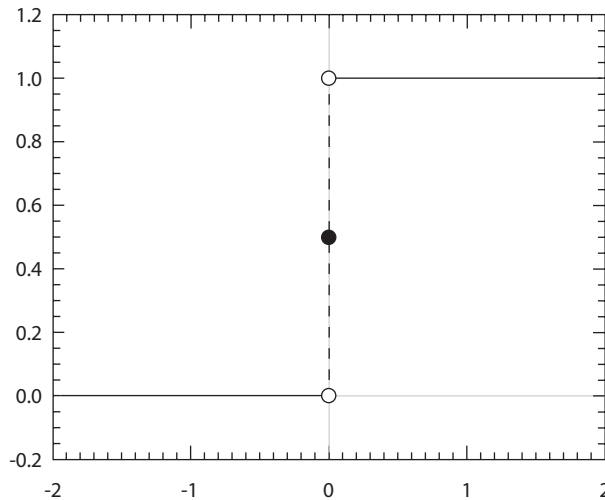


Figure 15.2 Plot of heavyside “step” function.

Contrary to what most believe about McCulloch and Pitts’ early intentions, they themselves did not hold too strongly to this *all-or-none* binary principle and acknowledged that the process could be continuous as well rather than only binary (Abraham, 2002). It is also worth noting that McCulloch and Pitts recognized from the outset that their neuronal model was likely *not* an exact representation of neuronal function. As noted by Abraham:

Their goal in the paper was not to present a factual description of neurons, but rather to design “fictitious nets” composed of neurons whose connections and thresholds are unaltered ... They noted emphatically that they were not arguing that any “formal” equivalence between their nets can be part of a “factual explanation” ... in no way did McCulloch and Pitts claim their model was a true description of known networks.

(pp. 20–21)

In this early depiction, or shall we say “idealization” of the nature of neuronal transmission, there are no adjustable parameters or weights (Izenman, 2008, p. 321). That is, each of  $x_1, x_2, \dots, x_p$  are not weighted by scalars but instead are simply summed, and if that sum exceeds a given threshold, then the signal is activated. To forecast further developments of the neuron and to be even more general, we could say that  $x_1, x_2, \dots, x_p$  are implicitly weighted, but by scalars of *one* for each input. However, these weights are not *adjustable*, and hence as noted by Izenman (2008), they were not a sufficient model for biological systems to “learn.” Forecasting to later developments up to the modern day neural network, it is the *adjustable* nature of these scalars that will constitute *learning* in modern neural networks. Of course, this idea of “adjustable scalars” is nothing new since virtually all statistical models beginning with regression in the 1890s featured adjustable weights and scalars. Adjusting the weights based on learning from the data is what statistical modeling, or statistical *learning*, has always been about since its inception. Optimizing weights is what learning actually *is*, from a statistical point of view.

Hence, though the McCulloch and Pitts neuron was not thought to “mimic” brain activity *exactly*, it was a pretty good start and first stab at mathematically representing the communication between neurons. Computational advances would have to take place before one could seriously think of modeling neuronal activity. In the brain, there are approximately 86 billion neurons, many of which are interconnected and can stimulate other neurons (Cheng & Titterton, 1994). To mimic the parallel processing of neurons, *parallel processing* in computing power would have to arrive in full, which would have to wait until the 1980s. It is the computer that provides the capacity for neural networks and many other statistical and machine learning methods to actually become pragmatic for use. However, the mathematics underlying these and other models is distinct from computation. Mathematical studies of the perceptron continued with Rosenblatt (1959) but it was Novikoff (1963) who became more widely known for providing the conditions for convergence of the perceptron.

*In the early McCulloch-Pitts neuron, though input values were summed and then subjected to a threshold “activation” function, none of the inputs were weights nor were they adjustable. Hence, there was no way, from a modern point of view, for the neuron to actually “learn.” At the time, regression models had long been on the scene since the 1890s and featured adjustable weight parameters.*

## 15.2 Rosenblatt’s Perceptron

Rosenblatt aimed to improve the MC neuron which meant applying weights to the inputs. His ideas were influenced by Donald Hebb’s book in 1949 called *The Organization of Behavior* (Hebb, 2002). The book conceptualized memory traces via sets of connected neurons which help determine future behavior. Such connections were referred to as *Hebb synapses*. As Izenman (2008) notes, Hebb’s book was a significant development in behavioral neuroscience even if its findings have been largely overturned since its writing. Hebb’s ideas motivated a large body of work.

Rosenblatt was not convinced that most neural connections were random and that cell assemblies could self-generate within a purely homogeneous mass of neurons. He believed instead that he could improve upon Hebb’s work and constructed a “minimally constrained” system he called the *perceptron* (Rosenblatt, 1958, 1962). A perceptron is nothing more than a McCulloch-Pitts neuron but *now with adjustable connection weights*. The magnitude of the weights communicates the strength of the connection. The perceptron is hence more flexible than the aforementioned MP neuron.

In Rosenblatt’s perceptron, depicted in Figure 15.3 (right), we see the extension of the MC neuron (left):

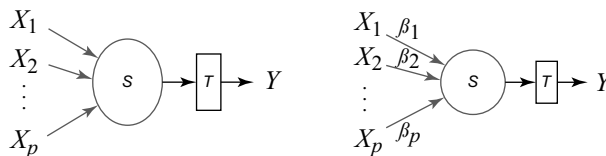


*Figure 15.3* Rosenblatt’s perceptron (right) extending on the classic MC neuron (left) featuring adjustable weight parameters  $\beta_1 \cdots \beta_p$ . Used with permission of Springer Nature BV, from *Modern Multivariate Statistical Techniques* by Alan J. Izenman, © 2008; permission conveyed through Copyright Clearance Center, Inc.



Recall that for the MP neuron, the neuron either fired or did not fire subject to the threshold or heavyside function (e.g., +1, -1). Rosenblatt's perceptron is similar to the MP neuron, but with one crucial difference; the inputs  $x_i$  are now associated with "connection weights,"  $\beta_1 \cdots \beta_p$ , where positive weights  $\beta_p > 0$  are *excitatory* synapses, and negative weights  $\beta_p < 0$  are *inhibitory* synapses. These weights could then be summed to obtain the overall *action* of the neuron or the strength of the connection (Izenman, 2008, p. 322).

Does all this sound familiar? Of course it does. It sounds very much like regression analysis where  $\beta_1 \cdots \beta_p$  are simply partial regression weights. As with the MP neuron, the sum of values in Rosenblatt's perceptron is computed and then subjected to the threshold binary logic. The difference, of course, is that the sum this time consists of weighted values of  $x_i$ , that is,  $S = \sum \beta_j x_j$ . Thus, it is easy to see that the MP neuron is a special case of the perceptron where, as previously mentioned,  $\beta_1 \cdots \beta_p$  are all understood to be unity such that

$$S = \sum_j \beta_j x_j = \sum_j (1) x_j = \sum_j x_j$$

or other fixed constant. A final component to the perceptron is the inclusion of what is known in neural networks as a *bias element*,  $\beta_0$ , which is nothing more than the familiar *intercept* in regression.

### 15.3 Big Picture Overview of Machine Learning and Neural Networks

Having surveyed the McCulloch-Pitts neuron and Rosenblatt's perceptron, we now consider neural networks in greater generality. First and foremost, in its most general and wide interpretation, machine learning can be said to be about *function approximation*. Of course, the problem of approximating functions was not at all "new" at the time machine learning arose since scientists and mathematicians had been working on function approximation in one way or another since at least the Greeks. Even the derivative in calculus is a way of approximating a *nonlinear function* with one that is *linear*, which is the tangent to the curve. The integral can also be said to be an approximation to the area under a curve. The Greeks' method of exhaustion in approximating the integral was just that, an exercise in *approximation theory*. Indeed, approximation in mathematics is an ancient concept and one that in one form or another has been applied countless times over history.

What was new, however, with regard to neural networks, were the algorithms and methods advanced to accomplish such function approximation. The infamous *backpropagation algorithm*, which is central to modern neural networks, actually had its origins in the 1960s, but was only fully recognized in the 1980s and constituted a major step in the advancement of function approximation and hence neural networks as well. *Deep neural networks* are thought to have originated in the mid-1980s (Dechter, 1986). What made these networks "deep" was the idea of *layering* of networks, that of including more than a single layer in the given neural network. Essentially, this meant that one could *approximate the approximation of a function* and do this iteratively through the chain of layers. By the mid-1990s, Cortes and Vapnik (1995) introduced the *support-vector network*, which showed great promise when compared to more traditional methods. This work initiated further work in machine learning.

More modernly, neural networks are increasingly popular and are used in addressing very meaningful and complex applied problems such as self-driving cars, detecting financial fraud, medical diagnoses, among many others. Why is this the case? What makes neural networks so attractive to practitioners, perhaps more so than any other statistical or machine learning technique? Alpaydin (2020) nicely summarizes their appeal:

Deep learning methods are attractive mainly because they need less manual interference. We do not need to craft the right features or suitable basis functions or kernels, or worry about optimizing the exact network architecture. Once we have the data – and these days we have “big” data – and sufficient computation available, we just wait and let the learning algorithm discover by itself all that is necessary.

(p. 315)

Kroese et al. (2020) also do a nice job of contextualizing machine learning algorithms relative to the previously discussed early (and perhaps in some cases, overly ambitious) origins that sought to mimic neuronal activity:

The attempts at building efficient machine learning algorithms by mimicking the functioning of the human brain have been as unsuccessful as the attempts at building flying aircraft by mimicking the flapping of birds’ wings. Instead, many effective machine algorithms have been inspired by age-old mathematical ideas for function approximation.

(Kroese et al., 2020, p. 324)

### 15.3.1 *What Is a Neural Network? Minimizing the Hype*

Neural networks can become very complex, but in understanding just what *is* a neural network, it is helpful, as is true with all statistical methods surveyed in this book, to strip them down to their most *basic elementary level*, akin to explaining the nature of an algorithm with an example as something as simple as long division. The power in understanding, in general, is that of coming across a term or concept that you believe must be very complex, then realizing you already know what it is through deeper examination and reminders of simpler examples and prototypes with which one is already familiar. Indeed, this has been the pedagogical approach adopted throughout this book. As we will discuss with reference to neural networks, in a strong sense, we already know what they are because we have already surveyed models that we will find to be extremely similar to them. Indeed, as remarked by Sarle (1994) regarding the relatively early days of neural networks,

NN [neural network] researchers routinely reinvent methods that have been known in the statistical or mathematical literature for decades or centuries, but they often fail to understand how these methods work.

(p. 1)

Perhaps a bit extreme and exaggerated, Sarle’s argument does have some merit. Neural networks are their own methodology, but they do hold equivalencies (or *near* equivalences) to other methods under specific *special case* conditions. Our goal here is to strip them down to their essential constituents to learn and appreciate what is actually *there* instead of getting lost in the verbiage and hype of their jargon. The jargon and fanciful statements made about neural networks are useful for *marketing* them, but as one who wishes to truly understand them, the reader must truly seek to comprehend what is going on, as it were, *under the hood*.

### 15.3.2 *Neural Networks Are Composite Functions*

First and foremost, a neural network could be as simple as a linear regression model, or as complex as several nonlinear models. A neural network can also be *nonparametric*. One reason why neural networks are often referred to as “deep” learning is simply because they take model *functions of*

*functions*. The “depth” is in the layering of what are referred to as *hidden layers*. *Multiple levels of composition* are what neural networks are all about (Goodfellow et al., 2016).

As a very simple example of so-called “depth,” consider the function  $f(x) = x^2$  which is that of a parabola. The function  $g(f(x)) = \sqrt{x^2}$  is more complex and deeper since it arises as a result of taking a function of the first function. That is, the outside function  $g(x)$  is applied to the inside function  $f(x)$ , which constitutes a *composite function*, in our case, *the square root of x-squared*. Shallow neural networks typically contain only a single hidden layer with only a few nodes. A *deep network* consists of several layers with many nodes, each of which might range in complexity. In a deep network, at each hidden layer we are taking (typically) relatively complex functions of prior layers. As we increasingly take *functions of functions*, the network is learning characteristics of the features at an increasingly more complex level of abstraction, capitalizing on even the smallest of discriminators that will help better predict the output. These often include nonlinear and interacting relationships on the journey from input to output.

Because of their complexity, neural networks have often been called “black boxes” as a result of the numerous nonlinear and interacting elements that are not explicit (Yosinski et al., 2015). That is, neural networks may indeed perform well under many circumstances, but their “black box” nature prevents us from knowing exactly *why*. This is one of the drawbacks to these networks, in contrast to other techniques such as regression where one can obtain a listing of partial regression coefficients in an attempt to evaluate the *relative performance* of predictors in a model. A neural network typically does not provide such precise individual variable performance information, and in some cases, this is understandable and for good reason; the number of parameters modeled in a neural network can be upward of over 50 million and hence attempting to understand how those parameters relate to one another is a gargantuan, and most likely, impossible task. Efforts to better understand the performance of neural networks via sophisticated visualization techniques is an ongoing area of research that may assist in “decoding” the black box of these networks and untangling their interacting, nonlinear components (Yosinski et al., 2015).

We can thus far then sum up the fundamental property of neural networks, which holds that neural networks, at their ultimate core, are *combinations of combinations*, that is, they are *composite functions*. For now, we characterize them as composite functions without any further constraints such as nonlinearity or linearity for the reason that *neural networks need not be nonlinear in nature*. As we will see, they are *often* nonlinear, such that they can transform nonlinear inputs into more linearly separable abstract features (Dubey et al., 2022), but the key point is that *they need not be nonlinear*.

*Neural networks can be attractive approaches to data analysis due to their inherent complexity and nonlinearity possibilities which do not need to be modeled overtly and instead can be relegated to a “black box” approach. By taking composite functions as many times as desired, the network becomes “deep” as it learns more about the variables in the model and their interactions at successively deeper levels of abstraction. While neural networks usually feature the modeling of nonlinearity, at a technical level, a neural network can still be linear.*

To understand the nature of a neural network, it is best to begin with the simplest case of such networks – the feedforward neural network with a single layer, which as we will see, the simple linear regression model can be considered a special case.

### 15.4 Single-Layer Feedforward Neural Network

We begin with the simplest example of a neural network, which is a *single-layer feedforward network*, or sometimes called the *single-unit perceptron*, illustrated in Figure 15.4.

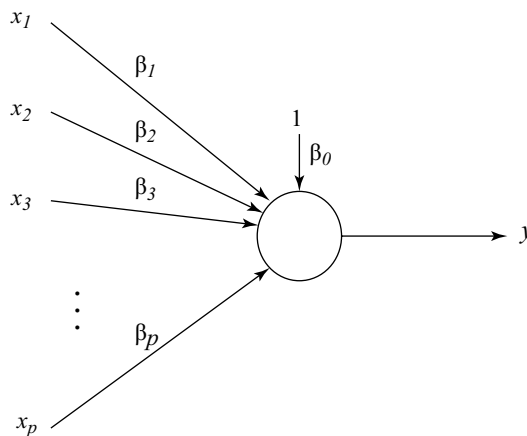


Figure 15.4 Single-layer feedforward neural network. From Warner, B., & Misra, M. (1996). Understanding Neural Networks as Statistical Tools. *The American Statistician*. Copyright © 1996 American Statistical Association, reprinted by permission of Informa UK Limited, trading Taylor & Francis Group, <https://www.tandfonline.com> on behalf of American Statistical Association.

The network is called a feedforward neural network for the reason that information “flows” (in necessarily a mathematical sense *only*) from input to output and there are no reverse feedback loops. When a network contains feedback paths, it is usually referred to as a *recurrent neural network* (Goodfellow, 2016).

In Figure 15.4, we see that all inputs,  $x_1$  through  $x_p$ , are weighted by  $\beta_1$  through  $\beta_p$ , where a *bias term*,  $\beta_0$  is also added, which is the familiar intercept term in classic regression. The inputs  $x_1$  through  $x_p$  are nothing more than observed variables as we would have in a model such as multiple regression (as opposed to “latent” ones as we encountered in structural equation modeling, for instance). In actuality, what we have in Figure 15.4 is in effect the familiar multiple regression model where  $y_i$  is nothing more than a function of an intercept  $\beta_0$  and a set of weighted predictors,  $\beta_i x_i$ :

$$y_i = \sum_{i=1}^p \beta_0 + \beta_i x_i$$

Why call it a neural network if it is simply a multiple regression model? In this most simple of situations, we do not have to. However, the benefit of doing so is the gateway to understanding neural networks in their full generality. The model in Figure 15.4 is a multiple regression only because the implied *activation function* is an *identity function*. A multiple linear regression does not allow us to change the nature of the activation function. This is where neural networks become more general than the classic multiple regression model. *In neural networks, we will be free to choose activation functions more complex than the identity function that is naturally implied or implicit in the multiple regression framework.*

Activation functions are central to neural networks, and as we will see later, can be chosen from a variety of possibilities. However, as we have said, if we choose the identity function in this case, the neural network reduces to that of a multiple regression. Recall an identity function is the following:

$$f(x) = x$$

which reads *what you put into the equation you get out*; there is no transformation at all on the input variable. Had we chosen a different activation function, perhaps a more complex *nonlinear* one, then we would have no longer had a simple or multiple regression and would have had a more complex neural network. For instance, instead of taking the identity function, we could have instead taken the log of the function as one example,

$$\begin{aligned} y_i &= \sum_{i=1}^p \beta_o + \beta_i x_i \\ &= \log\left(\sum_{i=1}^p \beta_o + \beta_i x_i\right) \end{aligned}$$

making the activation function no longer an identity function but instead a logarithmic one. *Activation functions are what give neural networks their life.* For example, a perceptron with a threshold activation function, that is, a “yes” vs. “no” binary outcome,

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

becomes a *binary linear discriminant* analysis and hence the perceptron in this case is imposing a familiar hyperplane. If the output is multi-valued and categorical, it becomes a multiple logistic function featuring a *softmax* activation function:

$$\text{softmax } f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (15.2)$$

Neural networks essentially act as a type of parametric/nonparametric regression model (more on this distinction later), which allows the user to model complex and often unknown nonlinear functional forms based on what the data have to say, rather than assuming a prior parametric form such as in the classic linear model. Be sure to note that this does not forbid us from assuming a functional form, but the point is that it does not *require* us to do so. This is why neural networks are often considered first and foremost, *nonparametric*. The parametric case in which prior assumptions are invoked is simply a special case of the larger nonparametric framework in which such assumptions are not advanced. If the relationship between inputs and outputs is known a priori to be linear, then typically a linear regression will usually outperform a neural network in terms of prediction (King, 1999).

How often do we know the exact functional form of a relationship? *How many scientific phenomena actually follow perfect linear trends?* As we have noted throughout this book, many phenomena are rather complex with unknown parametric forms, which is why so many machine learning techniques are nonparametric in nature allowing us to skip over the assumptions and get right to *what the data have to say*. That is, the algorithm *learns from the data*, not from our prior beliefs (theory) about the data or imposed (*induction biased*) parametric structures. Recall that we had said at the outset of this book that this was one of the distinguishing features made between classical statistics and the machine learning revolution, or perhaps even “rebellion.” Where classical statisticians have historically arrived at a problem with a priori functional forms and assumptions

largely already imposed on their data, machine learning types became more interested in simply wanting to know *what the data have to say*. Classical statistics and machine learning approaches overlap a great deal, of course, but the emphasis and approach to problem-solving can differ somewhat, especially in the age of massively large data sets and tremendous computing power where making distributional assumptions may not seem as relevant as with the smaller samples of yesterday. Of course, there is not simply one or two things that distinguish these fields; they overlap a tremendous amount and each field benefits from the other. Much of machine learning, despite its advancements, has relied on the field of statistics to make such headwinds.

We have said that neural networks are all about activation functions. We now survey such functions in more detail.

### 15.5 What Is an Activation Function?

Activation functions are at the very core of neural networks. An activation function is basically, at its most foundational and rudimentary level, simply a *function*, often better understood in the context of the previously discussed *composite function*. A composite function, as we have seen and as the reader may recall from basic calculus, is a *function of a function*. In theory, we can layer as many of these composites as we like and iteratively take *functions of functions of functions*, and so on. That is,

$$f(x; \theta) = f_L(f_{L-1}(\cdots(f_1(x))\cdots)) \quad (15.3)$$

where in (15.3)  $\theta$  is the parameter vector. The most “inside” function is that of  $f_1(x)$ . From there, we are iteratively taking *functions of functions*, where  $f_L$  is the most “outside” function, followed next by  $f_{L-1}$ . This idea of taking composite functions is precisely the concept of what makes a neural network *deep*. To put this into context again, recall that *linear regression can be conceived, from the most global perspective, as a neural network*. It is simply an elementary one that is linear, and for which the composite function is that of the *identity function*. That is, though we do not operationalize regression this way, we could in the way of composite functions by identifying  $f_1(x)$  as the regression function and  $f_{L-1}$  as the identity function where  $f_{L-1} = I(x)$ . Hence, the trivial neural network would be

$$\begin{aligned} f(x; \theta) &= f_{L-1}(f_1(x)) \\ &= I(f_1(x)) \end{aligned}$$

where in the above,  $f_{L-1} = I$ . When  $f_{L-1}$  is, for instance, a *nonlinear* function, then we have a more traditional representation and application of a neural network. This is the typical rationale given to activation functions in the literature, that they allow *nonlinearity* to be incorporated into the network. However, *an activation function need not necessarily be nonlinear to still make it a proper neural network*. Many writers require a neural network to be nonlinear in order to define it as a proper neural network and the reader should be aware of this.

The use of a composite function in neural networks, as we will see, also accounts for why backpropagation works so well, since it is based on the *chain rule of differentiation* in calculus, which as the reader may recall is all about differentiating composite functions. Hence, complex and as powerful as neural networks can become, they are all based on these very fundamental and elementary foundations.

### 15.5.1 Activation Functions Do Not “Activate” Anything

As we have said throughout this book, many statistical terms unwillingly denote “movement” or “action” of some kind. To one who is knowledgeable of how these terms are being used, no harm is done. However, to the newcomer, the use of such *power* and *action* words can be tremendously misleading. Indeed, recall that we earlier in the book discussed how mediation and moderation in no way necessarily implied movement or action. They are simply names for partitioning variance in different ways. If the *substantive* context of the problem implies movement or action, then that’s an entirely different matter. The point, however, is that statistical modeling, whatever the kind, could never on its own imply such movement no matter how “sexy” of terms used to describe the modeling.

The same can be said of activation functions. The outside function in  $g(f(x))$  is not “activating”  $f(x)$  in any way other than being applied, mathematically, to  $f(x)$ . Yet, the word “activating” seems to suggest a more important role, a physical movement of sorts, a more “exciting” story! Indeed, activation functions are sometimes referred to as *transfer functions*, but this again is *hyped-up jargon*, as activation functions, by themselves, do not “transmit” anything. A mathematical function, on its own, cannot physically transmit a thing! Now, neurologists might claim that neurons in the brain might *transfer* information, but that is a *scientific* matter, not solely a *statistical* one. Neural networks may have originally been influenced by neuronal activity as a paradigm, but in the end they are simply mathematical abstract functions. As was the case with so-called “causal modeling” in our survey of structural equation modeling, if a neural network is applied to model the electrical activity of the brain or some other presumed physical activity, then the word “transfer” may indeed make more sense. But such is not a property of the statistical model; *it is a property of the neural activity*.

Bounded activation functions (e.g., 0 to 1 or  $-1$  to 1) are also often referred to as *squashing functions* (Sarle, 1994). For instance, as we will see, the sigmoid function has this property of “squashing” values into a bounded interval (Lederer, 2021). Again, this is not meant to imply any physical action, but only to communicate that the activation function has the effect of outputting values within a bounded interval. Activation functions need also not be consistent across each output and different activation functions can be used for each.

### 15.5.2 Types of Activation Functions

As we have seen, we are unlikely to simply use identity functions and will prefer to use more complex functions, most of which will be *nonlinear* in order to capture complex nonlinear aspects of the data or processes we are seeking to model. The most popular activation functions in use in neural networks are, generally, the *logistic*, *tanh*, and *ReLU* (Lederer, 2021), however novel functions have recently been proposed, including *softplus*, *elu*, *selu*, and *swish*. We survey some of these activation functions shortly. Lederer (2021) provides an exceptional review of these functions and many more. According to Lederer (2021), *softsign* and *ReLU* should be the preferred activation functions in practice. The *softsign* function has been found to have computational advantages over *tanh* due to its simplicity (Lederer, 2021).

There are some important things to consider in choosing an activation function, however. Indeed, the choice of an activation function can have rather dramatic effects on the performance of a neural network, and hence how to choose a “good” activation function has been ongoing research in the neural network community for years with entire literatures devoted to the subject. As noted by Lederer (2021), “the literature contains an entire zoo of different activation functions, and arguments for and against specific activation functions are often based on heuristics and anecdotal

evidence” (p. 2). Also, predicting which activation function will work best for a given situation can be very difficult to impossible (Goodfellow, 2016).

### 15.5.3 *Saturating vs. Non-Saturating Activation Functions*

Ideally, activation functions should be *differentiable* at most points and generally also increasing. Beyond that, we have the choice to select from a wide range of activation functions, some more preferable than others depending on the context. One very important distinguishing feature of activation functions is whether they are *saturating* or *non-saturating*. Non-saturating activation functions are generally preferred over saturating ones. The requirements for activation functions are not absolute, however, which is to say these requirements are flexible to some extent and not always preferable in all contexts.

A *saturating* activation function implies that for large inputs, the function asymptotes at a constant value, such as 0 or 1. This is a characteristic of the *logistic sigmoid* activation function (Dubey et al., 2022; Murphy, 2022),

$$\text{Logistic Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (15.4)$$

which saturates (or “asymptotes”) at values of 0 and 1 (Dubey et al., 2022). That is,

$$\lim_{x \rightarrow -\infty} \left( \frac{1}{1 + e^{-x}} \right) = 0$$

and

$$\lim_{x \rightarrow +\infty} \left( \frac{1}{1 + e^{-x}} \right) = 1$$

These functions essentially “squeeze” or “squash” the input to be within upper and lower bounds. Unfortunately, as we will soon discuss, saturating activation functions will generate a *vanishing gradient problem*, where the gradient of the objective function becomes very close to 0, and hence updating weights becomes virtually impossible. The *tanh* function,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (15.5)$$

also quite popular, also squashes the inputs, but this time between  $-1$  and  $1$  (Dubey et al., 2022).

Hence, one of the keys to training very deep neural networks is to select activation functions that are *non-saturating* (Murphy, 2022). A non-saturating activation function is one that does not asymptote at a constant value and hence typically goes to infinity. That is,

$$\lim_{x \rightarrow \infty} f(x) = \infty$$

The *rectified linear unit* (or “ReLU” for short) is one such non-saturating activation function,

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (15.6)$$



It is non-saturating because the range of the function is from 0 to positive infinity, that is,  $[0, \infty)$  as we can see from Figure 15.5 featuring a few competing activation functions.

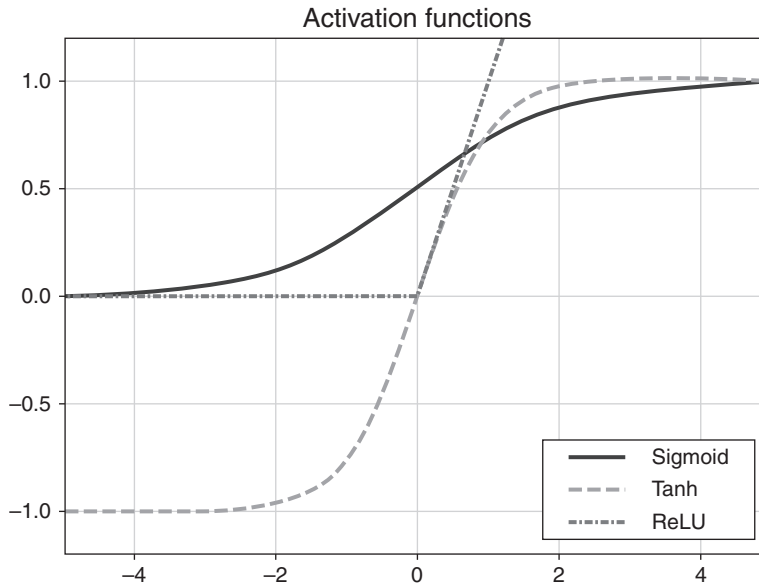


Figure 15.5 Some activation functions commonly used in neural networks. The tanh and sigmoid functions are saturating while the ReLU is a non-saturating activation function.

We see from Figure 15.5 that the *sigmoid function* is saturating, squeezing input values to be within  $[0, 1]$ . The *tanh* function is saturating for similar reasons, this time squeezing inputs to be within  $[-1, 1]$ . The *arctan*, another activation function, is also saturating. It is easy to see that the ReLU function is non-saturating, being constant at 0 for values equal to or less than 0 and increasing linearly for values greater than 0 and hence still allowing for updates to occur even for large values of  $x$  (Alpaydin, 2020). However, for negative inputs, the vanishing gradient problem still rears its ugly head (Dubey et al., 2022).

*A saturating activation function essentially “squeezes” the input values to be within specified bounds (e.g., 0, 1). A non-saturating activation function does not “bind” the input in this way and instead allows the function to typically go to infinity. When it comes to choosing an activation function for a neural network, non-saturating ones are often preferred over saturating ones.*

Incidentally, all of these functions can be easily plotted in software. For instance, a sigmoid activation function in Python (plot not shown) can be generated quite simply as:

```
import matplotlib.pyplot as plt
import numpy as np
import math
```

```

x = np.linspace(-20, 20, 100)
y = 1/(1 + np.exp(-x))
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("Sigmoid(X) Activation Function")
plt.show()

```

### 15.5.3.1 The Problem with ReLU

The ReLU function, also commonly known as the *positive-part function* or *ramp function* due to its shape, is one of the most common activation functions used in neural networks. The positive part of the function is merely the identity function, linear with slope equal to 1. It is this *near linearity* of ReLU that in part, makes ReLU attractive in the sense that it can allow for easy optimization using gradient-based methods as is the case with most linear models (Goodfellow et al., 2016). The negative part of the function is constant at a value of 0. A look at the ReLU function more closely reveals these characteristics (Figure 15.6).

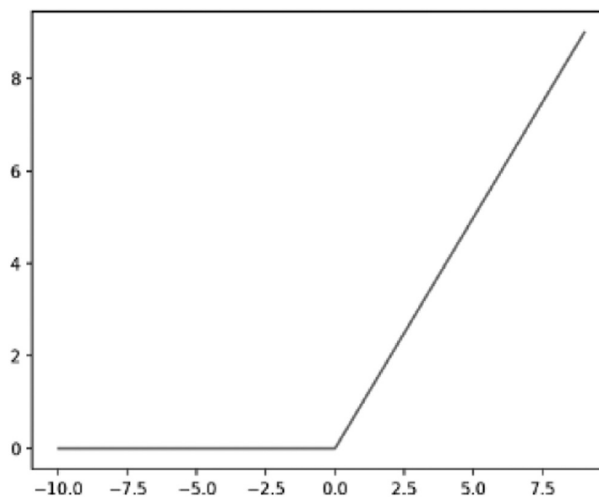


Figure 15.6 The ReLU activation function. For values equal or less than 0, the function is constant at 0, whereas for values greater than 0, the slope of the function is equal to 1.

We can see that the ReLU “cuts out” values less than 0 but activates values greater than 0 in a steady “ramp” fashion. One benefit of the ReLU function is that the function and its derivatives are very easily implemented at a relatively low computational cost (Lederer, 2021).

For all of its advantages, there is nonetheless a weakness with ReLU. The problem is if initialized weights are large and negative such may translate into many ReLU nodes being *inactive* during a big part of the training process, which can make it difficult or impossible for the algorithm to learn increasingly complex models. That is, there is no information going into the network for values less than 0 since they are set to equal 0 by default of the way the function defines negative values. This is known in the literature as the *dead ReLU* problem. It is a version of the *vanishing gradient problem* (Dubey et al., 2022; Lederer, 2021). Possible solutions to the problem include choosing a lower learning rate or replacing ReLU by *leakyReLU*, which we now discuss.

15.5.3.2 *LeakyReLU*

The *leakyReLU* is given by (where weight  $a = 0.01$  is featured)

$$LReLU(x) = \begin{cases} x, & x \geq 0 \\ 0.01(x), & x < 0 \end{cases} \quad (15.7)$$

A comparison of *ReLU* vs. *leakyReLU* appears in Figure 15.7.

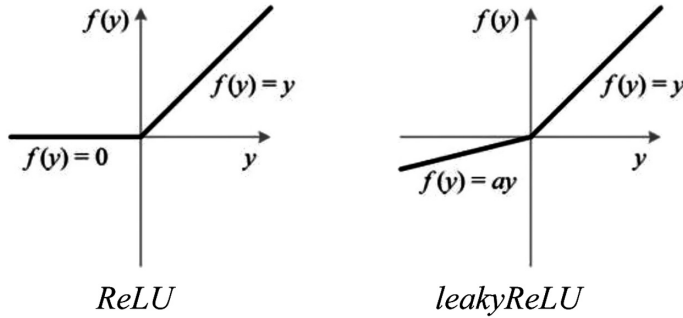


Figure 15.7 ReLU (left) vs. leakyReLU (right). The leakyReLU slopes downward for values less than zero instead of remaining constant at zero as with the ReLU.

We see that instead of the function being equal to 0 for all values less than 0, as was the case for *ReLU*, *leakyReLU* will simply equal some very small negative number instead of the constant zero. This translates to a small negative slope for values less than 0. Consequently, the range of leakyReLU is  $(-\infty, \infty)$  instead of  $[0, \infty)$  as it is for ReLU. This allows for updates for values less than 0 and avoids the vanishing gradient problem (Alpaydin, 2020). As Dubey et al. (2022) note, a problem with leakyReLU is in determining the correct slope for negative inputs, as different slopes may be appropriate or most relevant for different types of problems. The *parametric ReLU* (*PReLU*) attempts to rectify this by considering negative inputs as a training parameter, however this can lead to overfitting. See Dubey et al. (2022) for details.

Another activation function that attempts to improve on the ReLU is the *elu*, or *exponential linear unit*, which is a smooth function and thus has an advantage over ReLU:

$$elu(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases} \quad (15.8)$$

In *elu*,  $\alpha$  is a trainable hyperparameter in  $\alpha(e^x - 1)$ . While *elu* has shown positive results, its theoretical backing is lacking (Basirat & Roth, 2019). A variant of *elu* is *selu*, where the “s” stands for “scaled” and constrains the slope larger than one for positive inputs (Dubey et al., 2022; Lederer, 2021).

We could continue to detail a whole host of activation functions by looking at their algebraic functions, but it is perhaps most convenient to simply view the most common ones graphically as their graphs immediately tell more of their form than interpreting their algebraic counterparts will ever. In Figure 15.8 are a few of the more popular activation functions which include and extend on some we have surveyed thus far.

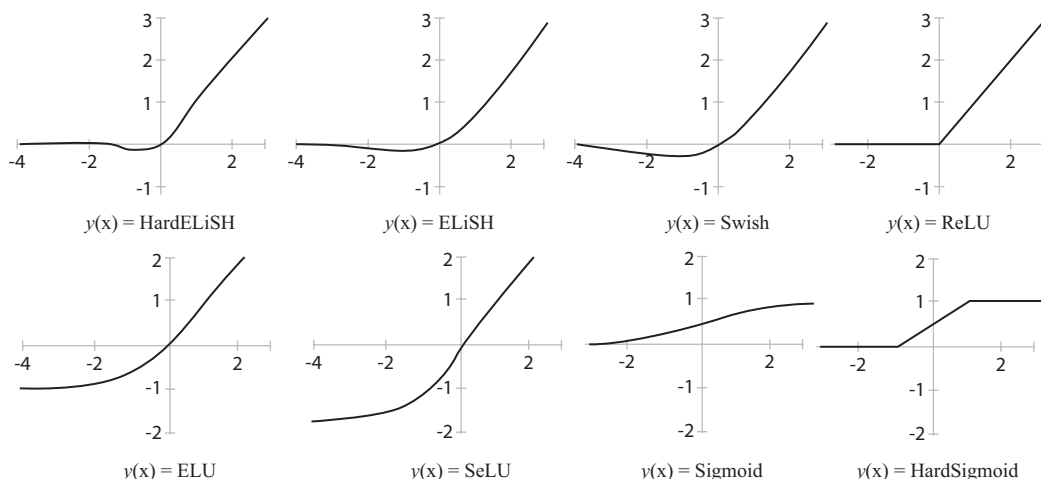


Figure 15.8 Plots of various activation functions used in neural networks.

#### 15.5.4 Which Activation Function to Use?

As discussed at the outset of our review of activation functions, according to Lederer (2021), *softsign* and *ReLU* (or *leakyReLU*) should be the preferred activation functions in practice. According to Basirat and Roth (2019), most deep learning approaches use *ReLU* largely due to its simplicity and reliability. Izenman (2008) notes that algorithms that feature the *tanh* function generally tend to converge more rapidly than those featuring the logistic, however the *tanh* itself has been found to be computationally inefficient (Dubey et al., 2022). As discussed, the *sigmoid* and the *tanh*, originally motivated by the biological neuronal firing of the brain in the early days of neural nets (Dubey et al., 2022), suffer from the *vanishing gradient problem* (Basirat & Roth, 2019) due to the fact that they are squashing functions. Recall this is because the gradient becomes very close to 0, resulting in virtually no update in parameters during training. With *ReLU*, since the derivative of positive inputs is equal to one, the gradient cannot vanish.

There is an entire literature and specialization within machine learning and mathematics devoted to evaluating existing activation functions as well as proposing new ones. Recent work has also introduced the idea of *learning/adaptive activation functions*, which, as the name suggests, contain learnable parameters to exploit more of the data's complexity. These include such activation functions as the *Adaptive Piecewise Linear (APL)*, the *swish*, *e-swish* and more. Consult Dubey et al. (2022) for a thorough and excellent review of the strengths and weaknesses of a variety of non-adaptive and adaptive activation functions and who also give a series of recommendations regarding which activation functions to use under a variety of conditions. While *ReLU* is commonly a default choice, these authors also recommend the *swish*, *mish*, and *PAU (Padé activation unit)* to be tried for most problems. Of course, there is absolutely nothing wrong with trying out a variety of activation functions for any given problem. There is no theoretical “type I error rate” on activation functions that would restrict the user from trying out a handful, so long as the resulting network is ideally subjected to cross-validation.

### 15.6 The Multilayer Perceptron – A Deeper Look at Neural Networks

We have thus far considered a very simple and basic neural network such that when the activation function was an identity function, mimicked a multiple regression. We now consider a more complex neural network, one that is more representative of neural networks actually used in practice

across scientific fields. The neural network we are about to discuss features a layer of hidden units over and above the single layer in the feedforward network surveyed earlier. The addition of one or more layers gives rise to the *multilayer feedforward network*, depicted in Figure 15.9.

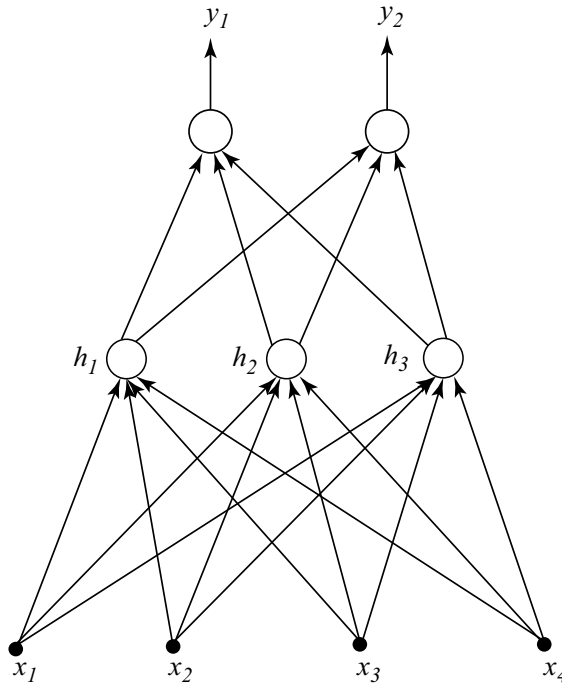


Figure 15.9 Multilayer feedforward network. From Warner, B., & Misra, M. (1996). Understanding Neural Networks as Statistical Tools. *The American Statistician*. Copyright © 1996 American Statistical Association, reprinted by permission of Informa UK Limited, trading Taylor & Francis Group, <https://www.tandfonline.com> on behalf of American Statistical Association.

As we will emphasize later when we compare neural networks to other statistical methods, the layering concept of a neural network, along with its depth, can be thought of as yet another *basis expansion* of the original inputs to the network (Hastie et al., 2009). However, these parameters are not imposed or a priori determined as in the basis expansion methods we surveyed earlier in the book (e.g., regression splines). Instead, they are arrived at by effectively *learning from the data*. This idea of *depth* in these networks was influenced historically by the ideas of *connectionism*, which essentially held that though one neuron may not be able to accomplish much on its own, several neurons acting together and interacting can possibly exhibit intelligent behavior (Goodfellow et al., 2016).

As we did with the single-layer network, we will unpack the multilayer network. In the network in Figure 15.9, a given hidden unit,  $h_i$ , is equal to the sum of products of the inputs  $x_i$  multiplied by the weights  $\beta_i$ :

$$h_i = \sum_{i=1}^p \beta_i x_i$$

The hidden layers in a multilayer feedforward network consist of variables arranged in successive layers, which are traditionally called *nodes*, *neurons*, or *processing units* (Izenman, 2008). Calling them “nodes” is probably the most general, as neurons and “processing units” implies something

that mathematics, by itself, cannot do. We can see in Figure 15.9 that there are two layers at which computation takes place, the *hidden layer* and the *output layer*. There are four input nodes and two output nodes, and three nodes in the hidden layer. This network can be then specified for regression or classification. In a multiple regression, there is only one output variable  $y_i$ , whereas in multivariate multiple regression there are many output variables  $y_i, \dots, y_{n+1}$ . In a binary classification, there is only one output variable  $y_i$  with values 0 or 1.

As was the case in the simpler neural network discussed earlier, an activation function is applied to the given hidden unit. That is, we take a *function of the given hidden unit*, which as we know by now that function is usually one that is *nonlinear* chosen from a set or family of prespecified activation functions. This is where, for instance, we could take the aforementioned logistic sigmoid,

$$\text{Logistic Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

or many other of the activation functions previously discussed. Simplified, as we move our way toward the output, that output is now a function of the hidden layer. That is, the output is a *function of a function of the inputs*:

$$f_i = \sum_{i=1}^N h_i(\beta_i x_i) \quad (15.9)$$

where in (15.9),  $N$  is the number of hidden units, and  $\beta_i$  is the weight from the hidden unit to output. The final step is the outputting of a function of  $f_i$ ,

$$\hat{y}_i = g(f_i) = \frac{1}{1 + e^{-f_i}} \quad (15.10)$$

where we see now  $f_i$ , that is, the inner function, takes the place of  $x$  in the sigmoid function. As we continue to move from layer to layer in the network, the concept of composite functions is very apparent. That is, neural networks continuously take *functions of functions* in arriving at their eventual final output.

## 15.7 Training Neural Networks

We have discussed the architecture of neural networks in some detail, but we have not yet surveyed how the weights in a neural network are optimized or trained. How are the weights in a neural network chosen? As with many statistical methods, they should be chosen in such a way as to *maximize* or *minimize* some function. The selection of the weights thus becomes a problem of *optimization* as has been the case in most other models surveyed in this book. That is, the “training” of the neural network consists of finding optimized weights similar to how we found optimal weights in linear regression using ordinary least-squares or discriminant analysis, for instance. “Training” a statistical or machine learning model means running the model through the data such that weights (or *coefficients*) are optimized by some criteria. In the past days of statistics, we *fit* models to data; today we *train* them in machine learning. Some of this terminology is simply due to a shift in zeitgeist. However, in a wide variety of statistical learning, the “learning part” is usually the updating of scalars in line with obtaining maxima or minima subject to some algorithm. That is, a big part of “learning” is simply *optimization*.

### 15.7.1 Backpropagation and Minimizing Error Sums of Squares

To optimize a neural network, backpropagation (sometimes called *error backpropagation*) is typically employed. It should be emphasized that backpropagation is used to compute the gradient, while a gradient descent algorithm uses this computed gradient to perform the learning (Goodfellow, 2016). Hence, the backpropagation algorithm and gradient descent are used in conjunction with one another, not in place of each other. Backpropagation has also been called the *delta rule* (Hastie et al., 2009; Widrow & Hoff, 1960). The algorithm consists of two phases, the *forward phase* and the *backward phase*. Essentially, the forward phase provides an opportunity for the model to initially learn the weights based on the data; the backward phase gives the model a chance to “correct” or “improve” on the weight estimates based on *feedback* for how it performed in the forward phase. It is in this sense that neural networks *learn*, just as a human being learns from his or her errors (in studying statistics, for instance) and improves performance. This is why the popularization of the backpropagation algorithm in the 1980s was so important to neural networks. It would not be an exaggeration to say that the modern rise of neural networks is mostly related to the discovery of the backpropagation algorithm along with the computational power to handle increasingly large networks.

But what function shall the neural network optimize when doing the learning? While a number of functions can be minimized, one convenient and familiar function for continuous responses is that of minimizing error sums of squares with respect to the network’s connection weights,

$$E^2 = \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (15.11)$$

The extent to which  $E^2$  is minimized can also be used as a performance metric for the neural network. The backpropagation algorithm itself is an *iterative gradient-based algorithm*, using randomly chosen initial values for the weights and employing primarily the chain rule from calculus in conducting its optimization. Essentially, backpropagation features repeated application of the chain rule to compute and update weights in the network by differentiating relative to an error function (e.g., error sums of squares),

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} \quad (15.12)$$

where  $E$  is the error function,  $w_{ij}$  are the weights from neurons  $j$  to  $i$ , that is, the elements we wish to *optimize*,  $s_i$  is the output, and  $net_i$  is the weighted sum of the inputs for a given neuron  $i$  (Riedmiller & Braun, 1993, p. 586). The gradient  $\frac{\partial E}{\partial w_{ij}}$  is then updated via

$$w_{ij}(t+1) = w_{ij}(t) - \varepsilon \frac{\partial E}{\partial w_{ij}}(t) \quad (15.13)$$

where  $t$  is the iteration and  $\varepsilon$  is the learning rate. That is,  $w_{ij}(t+1)$  denotes the weight  $w_{ij}$  at iteration  $t+1$ , where  $w_{ij}(t)$  is the current weight at iteration  $t$ , from which we subtract

$$\varepsilon \frac{\partial E}{\partial w_{ij}}(t)$$

which denotes the learning rate  $\varepsilon$  multiplied by the gradient  $\frac{\partial E}{\partial w_{ij}}$  at time (iteration)  $t$ . We see

there are two steps, one is computing the gradient, the other is updating the weights  $w_{ij}$  at  $(t+1)$ . If the learning rate is set too small, it may take a longer time to train the neural network than if the learning rate is set at a larger value; if set too large, such may result in less than optimal solutions. Aggarwal (2015) suggests beginning with a relatively large weight then gradually reducing the weight as more optimal values are settled upon.

Another option is to use a *momentum* adaptation,

$$\Delta w_{ij}(t) = -\varepsilon \frac{\partial E}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1) \quad (15.14)$$

where  $\mu$  is a momentum parameter (Riedmiller & Braun, 1993, p. 586) which serves as a direct scaling factor for the previous step (i.e.,  $w_{ij}(t-1)$ ). The effort to adapt weights took a big step with the work of Riedmiller and Braun (1993) who proposed *resilient propagation*, commonly known today as simply *resilient backpropagation*, which provides an improved adaptation process. Details of the algorithm are given in Riedmiller and Braun (1993) and Riedmiller (1994). Resilient backpropagation is what is used in the R package *neuralnet* that we will feature later in the chapter as a demonstration of neural networks using software. For much deeper details on how optimization works in the multilayer network, see Izenman (2008) or Warner and Misra (1996, pp. 288–289), who provide an excellent account and overview.

As noted, backpropagation helped neural networks “rise to fame” in the 1980s. The development of the backpropagation algorithm is primarily responsible for the relatively recent resurgence of interest in neural networks (Warner & Misra, 1996). The training of neural networks is a major challenge that requires both ingenuity and much experimentation. The algorithms for training neural networks with great depth are collectively referred to as *deep learning methods*.

Despite its success, there is no proof that the backpropagation algorithm always converges. Further, it can be a slow learner, its estimates may be unstable, and there may exist many local minima as the function being optimized is a nonconvex function of the model parameters (Bishop, 2006). Further, because backpropagation utilizes the chain rule, derivatives are multiplied which can cause problems of the gradient either vanishing or exploding (Alpaydin, 2020). Overfitting can also be a problem in backpropagation and attempts for it to converge can be addressed by early stopping as well as through other approaches we will soon discuss.

## 15.8 How Many Hidden Nodes and Layers to Include?

How many nodes and layers should a network have? This is an empirical question, and surprisingly continues to be an open question in the literature on neural networks. In general, *the closer the approximation desired, the larger the number of hidden nodes required*. In this sense, deep neural networks are usually preferred over shallow ones and empirical and theoretical evidence supports this. Yet at the same time, Ockham’s razor should be kept in mind, keeping the model as simple as possible while maintaining its ability to generalize well. Cross-validation can be used, as well as trial-and-error in determining the number of hidden nodes and layers. As we will see shortly when we discuss the universal approximation theorem, it can be shown that a multilayer perceptron with only a single hidden layer is a *universal approximator*. Such implies that a multilayer perceptron can model any suitably smooth function to any desired level of accuracy assuming a sufficient number of hidden units (Murphy, 2022).



It should be emphasized as well that a feedforward neural network with linear activation functions and  $n$  layers is equivalent to a linear neural network without hidden layers. Hence, the idea of “adding layers” in this case does not increase the approximation “power” of a linear neural network at all, unlike for nonlinear neural networks. In this sense then, the nonlinearity of neural networks is essential when it comes to better approximating functions based on increasing hidden layers. This is one reason why neural networks are almost always described as nonlinear. However, formally, they need not necessarily be so. If the number of hidden units in a single-layer neural network is chosen to be smaller than the number of inputs and all activation functions are linear, then information is essentially lost in the dimension reduction, in which case the neural network would essentially mimic a principal components analysis (Bishop, 2006).

## 15.9 Overfitting in Neural Networks

Overfitting is a problem with neural networks perhaps more so than in any other technique. This is largely due to their complexity. It is very easy to design a quite complex network that will provide a reasonably good fit to one’s training data. However, as emphasized throughout this book, what is a good fit on the *training* data does not necessarily imply an equally good fit on *test* data. We have just discussed that the complexity of neural networks should be minimized, *all else equal*, as to avoid an unnecessarily complex network when a simpler one will garner the same or similar fit. Beyond that general principle, the question then is how to prevent overfitting? We survey some of the more common methods, though many more approaches can be found in the literature.

### 15.9.1 Early Stopping

One approach to preventing overfitting is to simply stop the model short of fully iterating. This is known as the *early stopping* solution to mitigating overfitting. The longer a network is trained, the more it capitalizes on *noise or chance factors* and thus fits the training data quite well, but at some point will increasingly generalize poorly to validation data. The complexity of the system is increased with training and hence generalizability suffers (Alpaydin, 2020). In other words, the network might fit really well on the training data, but error increases substantially on new data because, essentially, what the network was so “good” at on the training data past a certain point was in effect capitalizing on chance or random noise. The case study analogy once more works well here, such that the network becomes an “expert” on the current case, but the theory (i.e., in the form of optimized estimated weights) developed on that case does not do nearly as well predicting the outcome of new cases. If one purposely stops the network early from iterating fully, one can possibly mitigate overfitting.

### 15.9.2 Dropout Method

A second method to dealing with overfitting is that of the *dropout approach*, which prevents *complex adaptation* of the hidden units (Srivastava et al., 2014). The method purposely drops some of the units out on each iteration and hence each unit learns to perform well even if some other units are missing. This approach estimates what is known as a *noisy version* of the weights since dropping weights equates to adding noise to units. As a result of the method randomly dropping weights, it is generally referred to as a *stochastic regularization* technique. An excellent summary and implementation of the dropout method is provided in Srivastava et al. (2014).

A related solution to the dropout method is instead of randomly dropping weights, to instead equate (or in some forms of the method, *encourage* equality) some of the weights as a way of

regularizing the model and discouraging overfitting. This solution is known as *soft weight sharing* (Bishop, 2006; Nowlan & Hinton, 1992). Details are in Bishop (2006, p. 269).

### 15.9.3 Regularized Network

A third approach to prevent, or at least reduce the possibility of overfitting is to introduce a regularization parameter, that is, a penalty function to shrink estimates similar to what is accomplished in ridge and lasso regression, for instance. One form of regularization used in neural networks is that of *weight decay* where a penalty term is added to the error function. As detailed in Hastie et al. (2009), if the error function for the parameter vector  $\theta$  is  $R(\theta)$ , then the error function plus penalty is equal to

$$R(\theta) + \lambda J(\theta) \quad (15.15)$$

where  $J(\theta)$  is some penalty (e.g., see Hastie et al., 2009) and  $\lambda$  a tuning parameter. It is easy to see that as  $\lambda$  increases for  $J(\theta) \neq 0$ ,  $R(\theta)$  will increase which implies the error of the model will be larger than not and weights will be shrunk toward 0. For regularized least-squares specifically, the error function becomes (Bishop, 2006),

$$\begin{aligned} E(\mathbf{w}) &= R(\theta) + \lambda J(\theta) \\ &= E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \end{aligned}$$

where  $\mathbf{w}$  is the weight vector and which is an example of  $L_2$  regularization.

So, what could be the benefit to increasing error? As was the case with ridge and lasso regression, imposing a penalty has the effect of shrinking weights toward 0 and hence reduces overfitting. Both  $L_1$  and  $L_2$  are possible for neural networks just as they were for ridge and lasso regression. For further details, consult Hastie et al. (2009), who also provide a powerful demonstration of regularizing a neural network through weight decay (pp. 402–403). The weights used to initialize a network can be a factor in weight decay as well. Initial weight values should be small, random, and typically in the neighborhood of  $[-0.01, 0.01]$  (Alpaydin, 2020) from a uniform distribution, though much larger intervals have also been proposed.

## 15.10 Types of Networks

As we saw when we considered the simpler neural network, the multilayer network is likewise not that far removed from traditional regression. In fact, multilayer networks are analogous to nonlinear regression models for continuous responses and to discriminant analysis models for binary outcomes (Sarle, 1994).

A *fully connected network* is one in which all input nodes are connected to the nodes in the first hidden layer, and all nodes in the first hidden layer are connected to all nodes in the second hidden layer, and so on. If some of the connections are missing, then we have a *partially connected network*. By setting or constraining some of the weights to zero, we can obtain a partially connected network from a wider and more general fully connected network. This idea of constraining parameters is familiar by now; *most models constrain select parameters either implicitly or explicitly*. In general, in any statistical model we entertain, parameters can be fixed or constrained to particular values, or even more generally chosen to not be estimated at all. Usually, substantive theory and prior knowledge of the research situation will help determine which weights to fix. Sometimes

fixing parameters is a necessity to aid in identifying the model as in structural equation modeling. The important point for now is to appreciate the similarity of fixing weights in a neural network to that of fixing weights in other statistical models. It is an entirely general *concept* which simultaneously frees up degrees of freedom (such as we saw in regression splines when constraining the solution to have no jumps or discontinuities).

The neural networks we have surveyed thus far are all examples of what are known as *feed-forward neural networks* such that the “flow” of the network is from one end to the other (typically from left to right). However, there are many other types of networks. In fact, feedforward networks can be viewed as special cases of *recurrent networks*, where recurrent networks are not necessarily always forward-feeding and instead allow output from some nodes to influence nearby or subsequent inputs to those very nodes. Recurrent neural networks are often used in the processing of sequential data (Goodfellow et al., 2016). *Convolutional networks* are yet another type of network which typically feature the use of kernels and are usually subjected to regularization to mitigate overfitting. Such networks are often used in image and video recognition to help narrow in on specific or *local regions* in the case of an image, for example, and combine local features (e.g., the smaller details of what in totality make up a vehicle called a “truck”) to generate compound features (e.g., identifying the object as a truck instead of a car). Both recurrent and convolutional networks are well beyond the scope of the current text, and can become quite complex. For a good introduction, see Goodfellow et al. (2016) or Bishop (2006).

### 15.11 The Universal Approximation Theorem: The Appeal of Neural Networks

We have discussed thus far the architecture of neural networks, but we have not yet discussed why, theoretically, they are so appealing. Why bother with building a neural network? The key motivation behind neural networks, just as is the wish for any statistical method, is that they are remarkably good at *approximating functions*. This concept is housed in the *universal approximation theorem* and is the key to understanding the more theoretical side of neural networks. The theorem, as stated by Hornik (1991, p. 359) word for word, is the following:

Standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

This is a very profound and remarkable result! For instance, Cybenko (1989) has shown that a two-layer feedforward neural network with a sufficient number of hidden units can approximate any continuous function to any degree of accuracy. The approximation theorem also holds for a great deal of activation functions except for the polynomial (Bishop, 2006). See Hornik (1991) for approximation capacities for arbitrary bounded and nonconstant activation functions.

Typically, the closer the approximation desired, to reiterate, the larger the number of hidden nodes required (Izenman, 2008), but the key point to emphasize is that we are able in the first place to approximate functions as closely as we would like using neural networks. The theorem, due to Kolmogorov (1957) tells us that any continuous high-dimensional map can be represented as the function composition of much simpler one-dimensional maps. Again, this is a very powerful idea, as Izenman (2008) summarizes very well:

The universal approximation theorem is an existence theorem: it shows, theoretically, that one can approximate an arbitrary continuous function by a single hidden-layer network. Unfortunately, it does not specify how to find that approximation; that is, how to determine the weights and the number,  $t$ , of nodes in the hidden layer (a problem known as network complexity). It also assumes that we know the continuous function being approximated and that the available set of hidden nodes is of unlimited size. Furthermore, the theorem is not an optimality result: it does not show that a single hidden layer is the best-possible multilayer network for carrying out the approximation.

(p. 334)

What does it take for the theorem to work? It turns out that the only thing that is required to realize the approximation are mild assumptions on the activation function. And while the theorem typifies modern neural networks, its mathematical origins can be traced back to Hilbert’s 13th problem or conjecture, which held that *all high-dimensional continuous functions are a combination of two-variable continuous functions* (Kratsios, 2020). In machine learning, more generally, universal approximation is about “a model class’s ability to generically approximate any member of a large topological space whose elements are functions, or more rigorously, equivalence classes of functions” (Kratsios, 2020, p. 2). For a highly theoretical read of the approximation theorem, see Kratsios (2022).

**15.11.1 Visualizing the Universal Approximation Theorem**

Approximating functions has a long history in mathematics. Indeed, as Steffens (2006) has remarked regarding the centrality of approximation in mathematics, “Whenever one computes, one *approximates*” (p. vii). But what exactly does *function approximation* mean? Perhaps the easiest way to understand it is to liken it to what occurs in differential calculus as the prototypical elementary example of function approximation. For example, consider the function in Figure 15.10, that of a parabola along with its graph (left):

$$f(x) = -x^2 + 3$$

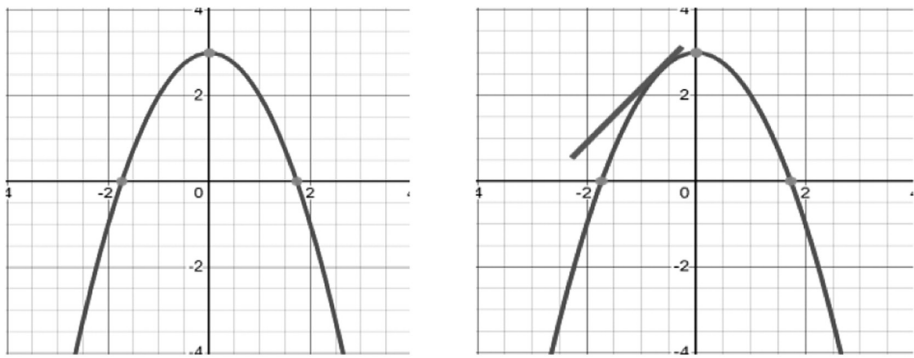


Figure 15.10 Inverted parabola (left) and concept of approximating a function at a point (right).

Suppose we would like to approximate the function at a value of  $x = -1$ . Of course, we could compute the slope of the line at  $x = -1$  which would give us the derivative, shown in Figure 15.10

(right). The derivative, as the reader will recall, is the *slope of the tangent to the curve* (or, more precisely, *the limit of the slope of secant lines approaching the given point*). That is, the derivative is *approximating the function* at that particular point. Indeed, if we computed derivatives at all infinitesimally small and infinite numbers of points along the  $x$ -axis and summed them up, we would have approximated the actual function of the inverted parabola.

As may seem obvious to the reader, the more derivatives we use the better the polynomial approximations become. They get better as well if we use higher-order polynomials to better capture the curve. These ideas suggest a much more general result which is that of the *Taylor polynomial* of degree  $n$  ( $P_n(x)$ ), which  $x$  for near  $x_0$  is given by,

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \quad (15.16)$$

and for which (15.16) is called the  $n^{\text{th}}$  Taylor polynomial for  $f$  at  $x_0$ . The approximation for points near  $x_0$  is given by  $x - x_0$ , and the approximation is rarely perfect, yielding a remainder term  $f - P_n$  (i.e., the actual function minus the approximation). If we allow the degree  $n$  of the polynomial to go to infinity, that is,  $n \rightarrow \infty$ , then the Taylor polynomial becomes simply a *Taylor series*. That is, a Taylor series is simply a Taylor polynomial of infinite degree. Put differently, a Taylor series is the limiting sum of a Taylor polynomial as  $n \rightarrow \infty$ .

Though a bit more complicated and involved than the simple Taylor series featured here, the essential idea is analogous in neural networks. Approximating functions is what neural networks, in the end, actually *do*. Often these functions are unknown and must be “learned” from the data. This is, in part, the *nonparametric* element of neural networks.

## 15.12 Neural Networks and Projection Pursuit

There is often a link drawn in the literature between neural networks and a nonparametric technique called *projection pursuit* (Friedman & Tukey, 1974; Friedman & Stuetzle, 1981; Kruskal, 1969). Understanding the latter can provide a window into better understanding neural networks. As the name suggests, in projection pursuit we are “pursuing” the best means by which to make a projection. The primary qualification for the pursuit is that the projection be in some sense, *interesting*. The original conception of projection pursuit was formulated by Kruskal (1969, 1972).

More to the point, projection pursuit is the search in a potentially nonlinear structure of high-dimensional data lower-dimensional linear projections (Posse, 1995). It is considered nonparametric since nonparametric estimators (*smoothers*) are used in the estimation process. Many excellent examples of projection pursuit can be found in Posse (1995). Projection pursuit is primarily a visualization technique with limited output as is traditional with methods such as principal components analysis (Laa & Cook, 2020).

Projection pursuit itself was first coined by Friedman and Tukey (1974) in what is an exceptionally well-written paper “A Projection Pursuit Algorithm for Exploratory Data Analysis,” but as mentioned had its origins somewhat earlier with Kruskal (1969). Friedman and Tukey introduced the method as an algorithm that seeks to find one- and two-dimensional linear projections of multivariate data that are relatively highly revealing:

Mapping of multivariate data onto low-dimensional manifolds for visual inspection is a commonly used technique in data analysis. The discovery of mappings that reveal the salient features of the multidimensional point swarm is often far from trivial. Even when every adequate description of the data requires more variables than can be conveniently perceived (at one

time) by humans, it is quite often still useful to map the data into a lower, humanly perceivable, dimensionality where the human gift for pattern recognition can be applied.

(Friedman & Tukey, 1974, p. 1)

Interestingly, Friedman and Tukey chose not to treat nonlinear dimension reduction techniques stating that they are more difficult to interpret but also for their computational complexity, a problem that has since been overcome by the modern uprise in computing power. Projection pursuit uses a *projection index* as a quantity that needs to be optimized in the search for interesting projections. Hence, we see immediately that *principal components analysis can be regarded as a special case of projection pursuit where the index is that of maximizing variance of orthogonal linear combinations*.

Components analysis, as we have seen, is by far not the only way to project data. For instance, returning to a simpler example, one could project data via ordinary least-squares. Most of the indexes used in projection pursuit are built around detecting departures from known distributions, such as multivariate normality, or projections that are best able to distinguish between groups (Laa & Cook, 2020). As Jolliffe (2002) notes, most projection pursuit techniques begin by assuming that multivariate normality is the *least* interesting basis for a projection, and hence many indices in the projection essentially measure deviations from this “uneventful” status. Numerically, the actual index is a scalar computed in the  $d$ -dimensional space and is maximized over all possible  $d$ -dimensional projections of a data set. A number of indexes are discussed in Laa and Cook (2020). For a concise summary of projection pursuit, see Jolliffe (2002, pp. 219–221).

### 15.12.1 *Projection Pursuit Regression and Relation to Neural Networks*

The model for *projection pursuit regression*, which can be considered a special case of the wider more general projection pursuit framework, can be stated as follows:

$$y_i = \beta_0 + \sum_{j=1}^r f_j(\beta_j^T x_i) + \varepsilon_i \quad (15.17)$$

where the model looks pretty much the same as we would expect from a typical linear regression model,

$$y_i = \beta_0 + \sum_{j=1}^r (\beta_j^T x_i) + \varepsilon_i$$

However, note the key difference, the inclusion of  $f_j$  in the projection pursuit of (15.17). For convenience, we will drop the error term  $\varepsilon_i$  for now and simply focus on the systematic portion of the model. In projection pursuit, we can have numerous functions  $m$  through  $M$  and hence we account for these via summation,

$$y_i = \sum_{j=1}^M f_j(\beta_j^T x_i)$$

and seek to minimize the following error function

$$\sum_i^N \left[ y_i - \sum_{m=1}^M f_j(\beta_j^T x_i) \right]^2 \quad (15.18)$$

We can think of  $\beta_j^T$  in projection pursuit as containing *directions* instead of merely scalars as we might typically think of such in ordinary regression (Hastie et al., 2009) though it definitely does not hurt to think of these weights as directions in classical regression as well. In projection pursuit,  $f_j(\beta_j^T x_i)$  is called a *ridge function*, with its direction determined by values in  $\beta_j^T$ . We note that essentially the only difference between this and the classic linear model is the inclusion of  $f_j$ , where these are a *collection* of unknown smoothing functions (i.e., “unknown” until they are “learned” from the data). That is, in  $f_j(\beta_j^T x_i)$  we are taking a function  $f_j$  of another function  $\beta_j^T x_i$  and hence once more, we are taking a *composite function* just as we do in a neural network. The number of these smoothing functions we will take is a *flexible hyperparameter*. When that number is one the model is sometimes referred to as a *single index model* (Hastie et al., 2009). Cross-validation can be used to determine an optimal number of smoothing functions. Its value can also be determined based on how much each successive term improves the model in terms of visualization.

We can see that the projection pursuit model is entirely general. Indeed, as noted by Hastie et al. (2009, p. 390), “the operation of forming nonlinear functions of linear combinations generates a surprisingly large class of models.” The link to neural networks is rather obvious then, such that in neural networks we are taking, as we saw, (*typically*) *nonlinear functions of linear (or nonlinear) combinations*. In projection pursuit, we could do the same if we chose a linear function instead of smoothing functions. That is, with a bit of generality, projection pursuit can be conceptualized as a layered neural network where  $f_j$  essentially becomes the activation function. Once we make  $f_j$  a classic activation function such as the sigmoid function instead of smoothing functions, projection pursuit essentially becomes analogous to a simple neural network.

One key difference between projection pursuit and neural networks, however, is that the activation function in the former can be different for each hidden unit and is typically estimated *nonparametrically*, where in neural networks such are usually prespecified parametric functions. However, this does not make neural networks entirely parametric, since they are especially useful when we have no theoretical preconceptions about the data. That is, neural networks “learn” from the data, which is a very “nonparametric” idea. And while it is technically possible to assign a *unique* activation function to each hidden unit in a neural network, it is not the norm.

Sarle (1994) perhaps distinguishes best between the parametric vs. nonparametric debate as it pertains to neural networks:

With a small number of hidden neurons, an MLP [multilayer perceptron] is a parametric model that provides a useful alternative to polynomial regression. With a moderate number of hidden neurons, an MLP can be considered a quasi-parametric model similar to projection pursuit regression ... An MLP with one hidden layer is essentially the same as the projection pursuit regression model except that an MLP uses a predetermined functional form for the activation function in the hidden layer, whereas projection pursuit uses a flexible nonlinear smoother. If the number of hidden neurons is allowed to increase with the sample size, an MLP becomes a nonparametric sieve that provides a useful alternative to methods such as kernel regression and smoothing splines. MLPs are especially valuable because you can vary the complexity of the model from a simple parametric model to a highly flexible, nonparametric model.

(p. 5)

Differences between the two methods also include the fact that parameters in a neural network are typically estimated *simultaneously* where in projection pursuit regression they are usually fit *successively* for each hidden unit (Klinke & Grassmann, 2000). Though not an exact analogy, the optimal number of hidden units in a neural network can be likened to the number of terms to add in a projection pursuit.

Thus, projection pursuit is often considered an historical and conceptual “precursor,” especially pedagogically, to understanding neural networks, and according to Hastie et al. (2009), projection pursuit was not widely used at the time of its inception largely due to computational requirements for obtaining a solution, computational “roadblocks” that have long been overcome with modern computing advances. Perhaps had more computing power existed back in the day of projection pursuit, these models would have a more significant legacy. Pedagogically, however, projection pursuit is very valuable in putting neural networks and other models (e.g., PCA and generalized linear models) into proper context. Indeed, a generalized version of PPR, aptly named *Generalized Projection Pursuit Regression* (Roosen & Hastie, 1993) has been proposed that is similar in structure to the generalized linear model paradigm of McCullagh and Nelder (1989) in its use of link functions and that allow for binary responses.

Lastly, it can be shown that as the number of smoothing functions increases without bound to infinity, for an appropriate set of functions in  $f_j$ , as was the case for neural networks, projection pursuit regression can approximate virtually any continuous function and is considered a *universal approximator*. How many smoothing functions to include in the approximation? There are different recommendations regarding this. For example, Friedman et al. (1984) found that adding more terms than necessary then using a backward selection algorithm helps avoid getting stuck in local minima. Roosen and Hastie (1993) have found that adding terms to maximize generalized cross-validation then pruning back somewhat may be an effective strategy.

Having compared projection pursuit to neural networks, it is also useful to recap where we began this discussion, in that projection pursuit can be understood as a higher-order and more general framework to principal components analysis. Such emphasizes that the latter is but only one *type* of projection, not *the* unique projection or dimensionality reduction of data. As we have seen throughout our discussion of dimension reduction techniques, what one technique might reveal from the data another technique may do better. There is no such thing as finding the *unique* dimensions of one’s data; *what you find is as much to do with what tool you use to do the finding*, which, incidentally is why some dislike statistics in the first place as it is perceived to provide no “hard” truths, a misguided perception based on an incomplete understanding of the subject.

*Principal components analysis is only one type of projection of the data. Projection pursuit (and by extension, neural networks) makes this clear – how you project your data will, to a large extent, determine what you find, and thus “see” from that data.*

Projection pursuit in R can be performed using the R package *tourr* (Wickham et al., 2011). It can also be performed using `ppr()` in R.

### 15.13 Summary, Warnings, and Caveats of Neural Networks

Neural networks can be easily misunderstood by the uninitiated. Their very name can encourage newcomers to give them powers they may not have, or at minimum, exaggerate the powers they actually do have. Neural networks are tremendously useful, but in learning them for the first time, it is important to understand exactly what they *are* and what they *are not*. Throughout this chapter



we have issued many caveats and warnings about neural networks. We summarize, and in some cases extend on those caveats and warnings in what follows.

### 15.13.1 Neural Networks Are Not Biological Nor Are They Machines

A first obvious point is that though neural networks were motivated by psychological and biological entities in the brain, they themselves have nothing to do with psychology or biology! They have everything to do with *abstract mathematics*. What is more, though they, like virtually all other statistical or machine learning methods are run on high-speed computers, there is nothing about neural networks themselves that have anything to do with computers or other “machines.” A neural network can, at least in theory and with enough time, be fit by pencil and paper just as any other mathematical technique. “Enough time” is more lifetimes than anyone might ever have, hence the benefit of using computers.

Why is this point so relevant? It is so because too often some users get into the habit of drawing conclusions from data based on the *jargon* of the statistical or machine learning method. As we discussed earlier in the book, in social statistics, for instance, all too often researchers will surprisingly conclude that such and such variable *mediates* or *moderates* another and impose an *action quality* on those words when no action is necessarily supported by the research design or the *organicity* of the variables used in the modeling. As emphasized throughout this book, *physical action of any sort can never be implied or concluded from the application of a statistical model alone, whether those models are of mediation, moderation, or neural networks*. The statistical or machine learning model never “knows” the kind of data or research design to which it is being applied; all models are able to do is process the data fed to them. *Models are entirely abstract entities*. So-called *action statements* must come from the *design* or *science*, not the *statistics*.

### 15.13.2 Neural Networks Need Not Be Nonlinear

As we have discussed, neural networks are typically *nonlinear* models. However, they need not necessarily be so. The special case, as we have seen, is the *linear* activation function. Of course, imposing a linear activation function minimizes the power of neural networks, and hence most neural networks are nonlinear. However, to say that they *necessarily* need to be nonlinear, or to *necessarily* associate nonlinearity with neural networks is a pedagogical shortcoming, as a linear neural network also, technically, counts as a neural network. Part of the confusion is simply in how writers and researchers choose to define a neural network, often as nonlinear. However, newcomers should be aware of this distinction.

### 15.13.3 Neural Networks Are (Mostly) Nonparametric

One question often asked about neural networks is whether they are *parametric* or *nonparametric*. In truth, they are a bit of both. For one, a user determining the number of nodes to have in a network is a *parametric-like* move. However, others regard the choice of the number of hidden neurons to include as essentially making neural networks nonparametric, especially as sample size increases. The debate as to whether neural networks are parametric or nonparametric is ongoing and is unlikely to be resolved anytime soon. This is due mostly to the wide variety of neural network possibilities and more precisely, how one operationalizes a specific given implementation of a neural network.

We recall that the traditional statistical definition of *parametric* holds that assumptions are made about the underlying population from which the sample data was drawn. Some say that since neural networks typically require no parametric assumptions about distributions, means, variances, or even correlation of errors (King, 1999), we can conclude that neural networks are essentially, and *mostly*, nonparametric models. The problem with this characterization, however, is that the user is able to freely impose conditions that do imply distributional assumptions to some extent, as well as through regularization techniques (such as weight sharing) that strongly resemble “parametric-like” actions. Hence, in the final analysis, neural networks *can* be nonparametric just as they *can* take on parametric features, depending on how the user sets up the network as well as any assumptions the user makes regarding the population to which the network is being applied.

In the end then, on the nonparametric to parametric *spectrum*, neural networks probably fall somewhere in between, and where exactly depends on the given neural network under consideration.

#### 15.13.4 Including More Layers Leads to Increased Approximation on the Training Data

In general, adding more layers to a neural network results in increased approximation accuracy of the underlying function, at least on the training data. However, this may lead to poor generalizability on test data and hence, overfitting. Even simple neural networks do well in approximating functions. For instance, Hornik et al. (1989) found that a neural network using arbitrary squashing functions could approximate virtually any function provided enough hidden units are supplied. The take-away here is that even though one can achieve accuracy by adding successive layers, the true test, as always, is on the *test data*, not the *training data*. By analogy once more, nobody cares much about how much you practiced your piano before the concert. They evaluate you based on your *actual performance*.

#### 15.14 Neural Networks in Python

We can quite easily build a simple neural network in Python using *sklearn* for data on  $X = \begin{bmatrix} 0. & 0. \\ 1. & 1. \end{bmatrix}$  and response  $y = [0, 1]$  having two layers, where the first layer has five neurons and the second, two:

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
... hidden_layer_sizes=(5, 2), random_state=1)
>>> clf.fit(X, y)
Out [222]:
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
```

The neural network has now been fit for classifying values on  $y$ .

```
clf.predict([[2., 2.], [-1., -2.]])
Out [223]: array([1, 0])
```

We see that the new observations are classified into groups 1 and 0 respectively. The  $\alpha$  parameter serves as a regularization term and helps avoid overfitting (try adjusting it and rerunning the model).

For further details on the above example, see *scikit-learn.org* as well as Müller and Guido (2017) for more examples using `MLPClassifier` as well as plotting decision boundaries. We encourage the reader to modify the layer sizes (`hidden_layer_sizes=(5, 2)`) as well in the above example to observe the corresponding impact on the ensuing class predictions.

### 15.15 Neural Networks in R

We adapt a very simple example of a neural network from the package *neuralnet* (Fritsch et al., 2019) for a binary classification problem on the iris data. An introduction to the package is provided by Günther and Fritsch (2010). The package is excellent for fitting neural networks and contains many more options than we are able to cover here. In our example below, we merely demonstrate how a neural network can be quickly fit. We encourage the reader to consult the package for further details.

For the current example, we model *setosa* (binary, “yes” vs. “no”) as a function of *petal length* and *petal width*:

```
> library(neuralnet)
> nn <- neuralnet(Species == "setosa" ~ Petal.Length + Petal.Width, iris,
linear.output = FALSE)
> nn
```

We obtain selective output, which reports the error function used (*sse*) and the activation function (*sigmoid* or “logistic”).

```
$err.fct
function (x, y)
  1/2 * (y - x)^2
[1] "sse"
$act.fct
function (x)
  1/(1 + exp(-x))
attr(,"type")
[1] "logistic"
```

We proceed with obtaining a plot of the network and note the error has been reduced to 0.012565 (Figure 15.11).

```
> plot(nn)
```

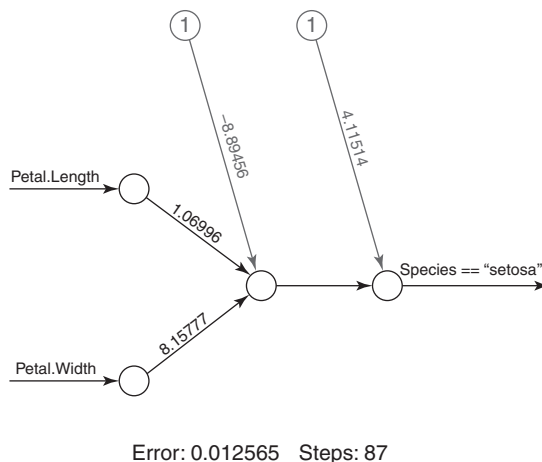


Figure 15.11 Plot of neural network on the iris data.

We can specify a custom activation function for the network other than the sigmoid function that is used as default. Here, as demonstrated in the package, we fit the same model but with a predefined *softplus* activation function (`act.fct = softplus`):

```
> softplus <- function(x) log(1 + exp(x))
> nn <- neuralnet((Species == "setosa") ~ Petal.Length + Petal.Width,
iris, linear.output = FALSE, hidden = c(3,2), act.fct = softplus)
```

We have also specified the structure of hidden layers to be three and two (`hidden = c(3, 2)`) moving left to right across the network (i.e., the first hidden layer has three nodes and the second, two nodes).

A plot of the network follows (Figure 15.12, left).

```
> plot(nn)
```

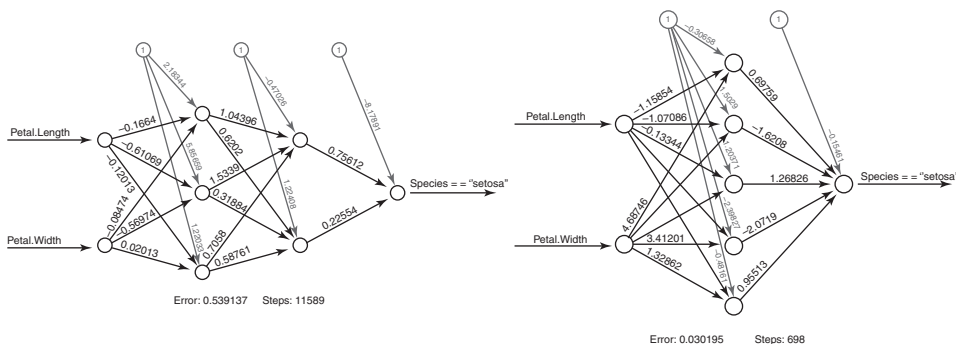


Figure 15.12 Neural networks with various numbers of layers and nodes along with estimated error.

We now run the same model but with only a single hidden layer having five nodes (Figure 15.12, right).

```
> nn <- neuralnet((Species == "setosa") ~ Petal.Length + Petal.Width,
iris, linear.output = FALSE, hidden = 5, act.fct = softplus)
> plot(nn)
```

Next we fit a model with two hidden layers, each with five nodes, for which we are able to get the error down to 0.003624 (Figure 15.13).

```
> nn <- neuralnet((Species == "setosa") ~ Petal.Length + Petal.Width,
iris, linear.output = FALSE, hidden = c(5, 5), act.fct = softplus)
> plot(nn)
```

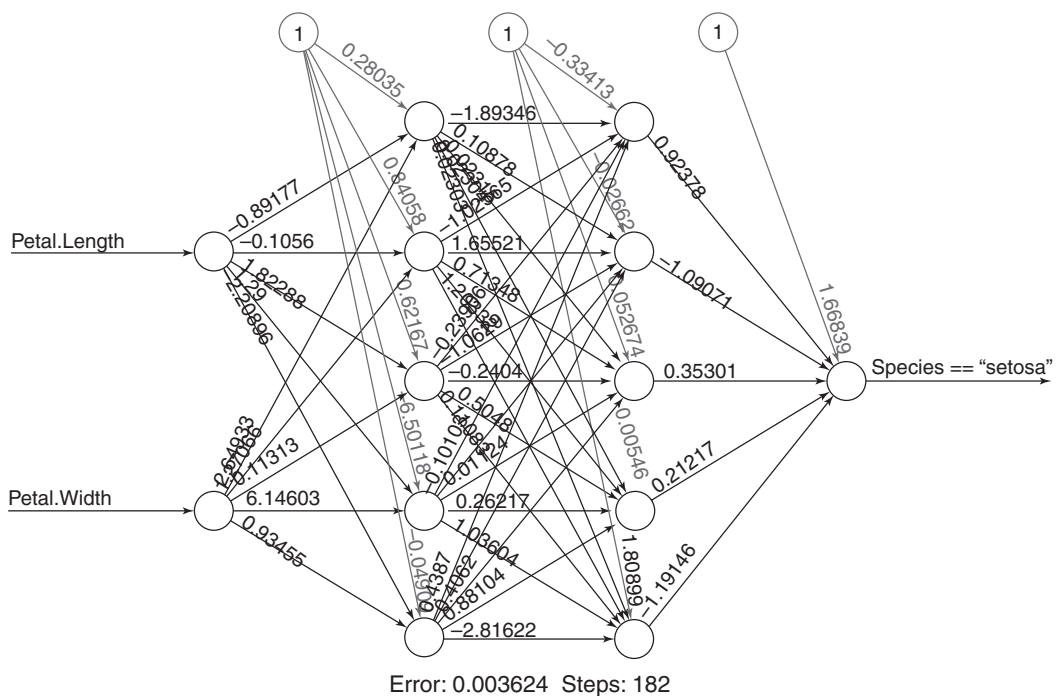


Figure 15.13 Neural network with two hidden layers each with five nodes.

A host of other output and functions are available in the package, such as computing confidence intervals for weights via `confidence.interval()` and obtaining AIC and BIC statistics, these latter two statistics requiring specifying `likelihood = TRUE` (error function = negative log-likelihood):

```
> neuralnet(Species == "setosa" ~ Petal.Length + Petal.Width, iris, like-
likelihood = TRUE, linear.output = FALSE)
```

```
$result.matrix
                                [,1]
Error                          0.009423536
reached.threshold              0.005993899
steps                         95.000000000
aic                           10.018847072
bic                           25.072023543
Intercept.to.1layhid1         5.558388265
Petal.Length.to.1layhid1      -0.085917885
Petal.Width.to.1layhid1       -7.246933549
Intercept.to.Species == "setosa" -4.875343467
1layhid1.to.Species == "setosa" 10.044111027
```

We see that AIC and BIC are equal to 10.02 and 25.07, respectively.

The package *nnet* (Venables & Ripley, 2002) also fits neural networks. For a good example using the package, see Venables and Ripley (2002, p. 342). The *keras* package (Allaire & Chollet, 2024) is a very extensive package for fitting neural networks in R and comes highly recommended. Efforts to visualize what is going on in the “black box” of neural networks have also been advanced (Yosinski et al., 2015) and is a currently expanding area of research in machine learning.

## 15.16 Chapter Review Exercises

1. Describe the *McCulloch-Pitts neuron*, and compare and contrast it to *Rosenblatt’s perceptron*, then compare and contrast both of these to a classic multiple regression model.
2. Describe and explain how neural networks are *composite functions*.
3. Compare and contrast the *activation function* in neural networks to the *link function* in the generalized linear model.
4. Distinguish between a *saturating* vs. *non-saturating* activation function.
5. What is the general problem with the *ReLU* activation function and how does the *leakyReLU* attempt to address this problem? Explain and illustrate.
6. Describe how neural networks are trained by the process of updating parameter weights.
7. Describe three methods to prevent *overfitting* of neural networks.
8. Explain what the *universal approximation theorem* has to do with neural networks.
9. Compare and contrast neural networks to *projection pursuit*.
10. Does a neural network need to be *nonlinear*? How about *nonparametric*? Justify your answer.
11. Run a neural network in R on the iris data using the example in the chapter as a guide, but instead of the species *setosa*, use the species *virginica* as the outcome variable. Compare and contrast your results to those in the chapter.

# Concluding Remarks

This book has featured an overview of multivariate statistics and machine learning, well suited for an introductory text at the advanced undergraduate or beginning graduate level for students primarily in the social or natural sciences. The goal of the book has been to survey methods from these two merging and overlapping areas in the context of the new data science zeitgeist of the 21st century. The primary motivation behind the book was based on the premise that traditional statistics courses for specialists in science are *no longer enough* and should now incorporate features of machine learning as to emphasize the commonalities between *statistics*, *machine learning*, and *data science* more generally, within a scientific context of *research design* and *measurement*.

The book aimed to provide a bird's eye “big picture” foundational view of a wide gamut of methods and techniques, building on a reasonable mix of requisite *theory*, *application*, and *computation* suitable for an introductory course with relatively minimal assumed prerequisites. The book placed great emphasis on unifying *themes* and *foundations* to understanding statistics and machine learning, demonstrating how many techniques and methods can be understood or conceptualized as special cases of wider and more inclusive approaches. A good scientist must understand the quantitative tools they use in addressing the richness of their data, for what one “finds” in their data can to a significant and often surprising extent be a function of what methodological or statistical “lens” was used to do the searching. *The strengths and limitations of the microscope are sometimes just as relevant in a scientific context as what is being observed under it.* It is hoped that this message has been well-received and that the reader has gained renewed awareness of how influential *context* is to the application of quantitative methods in what is essentially and most globally, *the art of generating narratives from experience*.

As also emphasized throughout the text, virtually all methods discussed have book-length treatments in their own right as well as often long and complex histories of development in professional journals and related publications. Hence, it should be recognized that our overview of these methods has simply *scratched the surface* into the richness and complexities of these data-analytic approaches. Still, it is hoped that the book has provided the student or professional a means to get into these techniques as a *starting point* and as a way to better understand and appreciate the *foundations* of the current quantitative landscape in which they practice their science.

# References

- Abbott, S. (2015). *Understanding analysis*. Springer.
- Abécassis, J., Josse, J., & Thirion, B. (2023). Causal mediation analysis with one or multiple mediators: A comparative study. *Technical Report*.
- Abraham, T. H. (2002). (Physio)logical circuits: The intellectual origins of the mcculloch-pitts neural networks. *Journal of the History of the Behavioral Sciences*, 38, 3–25.
- Adolfsson, A., Ackerman, M., & Brownstein, N. (2019). To cluster, or not to cluster: An analysis of clusterability methods. *Pattern Recognition*, 88, 13–26.
- Aggarwal, C. C. (2015). *Data mining: The textbook*. Springer.
- Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22, 207–216.
- Aizerman, M. A., Braverman, E. M., & Rozonoer, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, 821–837.
- Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19, 6, 716–723.
- Allaire, J., & Chollet, F. (2024). *keras: R interface to 'keras'*. R package version 2.15.0.
- Alpaydin, E. (2020). *Introduction to machine learning*. The MIT Press.
- Amemiya, Y., & Yalcin, I. (2001). Nonlinear factor analysis as a statistical method. *Statistical Science*, 16, 275–294.
- Anderson, T. W. (2003). *An introduction to multivariate statistical analysis*. Wiley.
- Anscombe, F. J. (1973). Graphs in statistical analysis. *The American Statistician*, 27, 17–21.
- Anton, H., & Dorres, C. (2000). *Elementary linear algebra*. Wiley.
- Artigue, H., & Smith, G. (2018). The principal problem with principal components regression. *Pomona Faculty Publications and Research*, 495. <https://doi.org/10.1080/25742558.2019.1622190>
- Attias, H. (1999). Independent factor analysis. *Neural Computation*, 11, 803–851.
- Bach, F. R., & Jordan, M. I. (2002). Kernel independent component analysis. *Journal of Machine Learning Research*, 3, 1–48.
- Bach, F. R., & Jordan, M. I. (2006). A probabilistic interpretation of canonical correlation analysis. *Technical Report 688*.
- Baguley, T. (2009). Standardized or simple effect size: What should be reported? *British Journal of Psychology*, 100, 603–617.
- Baron, R. M., & Kenny, D. A. (1986). The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology*, 51, 1173–1182.
- Bartle, R. G., & Sherbert, D. R. (2011). *Introduction to real analysis*. Wiley.
- Bartlett, M. S. (1953). Factor analysis in psychology as a statistician sees it. In *Uppsala symposium on psychological factor analysis*, Nordic Psychology Monographs, 3, 23–34.
- Basirat, M., & Roth, P. M. (2019). The quest for the golden activation function. *Proceedings of the ARW & OAGM Workshop*, 190.



- Bates, D., Maechler, M., Bolker, B., Walker, S. (2015). Fitting Linear mixed-effects models using lme4. *Journal of Statistical Software*, 67, 1–48.
- Baudry, J., Raftery, A., Celeux, G., Lo, K., & Gottardo, R. (2010). Combining mixture components for clustering. *Journal of Computational and Graphical Statistics*, 19, 332–353.
- Bauer, D. J. (2005). A semiparametric approach to modeling nonlinear relations among latent variables. *Structural Equation Modeling*, 12, 513–535.
- Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15, 1373–1396.
- Beltrami, E. (1873). Sulle funzioni bilineari (English: On Bilinear Functions). *Giornale di Matematiche*, 11, 98–106.
- Ben-Shachar, M., Lüdtke, D., & Makowski, D. (2020). effectsize: Estimation of effect size indices and standardized parameters. *Journal of Open Source Software*, 5, 2815.
- Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., & Bengio, Y. (2010). Theano: A CPU and GPU math compiler in python. *Proceedings of the 9th Python in Science Conference*, 1–7.
- Bilodeau, M., & Brenner, D. (1999). *Theory of multivariate analysis*. Springer.
- Bishop, C. M., Svensén, M., & Williams, C. K. I. (1998). GTM: The generative topographic mapping. *Neural Computation*, 10, 215–234.
- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Boehmke, B., & Greenwell, B. (2020). *Hands-on machine learning with R*. CRC Press.
- Boelaert, J., Ollion, E., Sodge, J. (2022). *aweSOM: Interactive self-organizing maps*. R package version 1.3.
- Boker, S. M., Neale, M. C., Maes, H. H., Spiegel, M., Brick, T. R., Estabrook, R., Bates, T. C., Gore, R. J., Hunter, M. S., Pritikin, J. N., Zahery, M., & Kirkpatrick, R. M. (2024). *OpenMx: Extended structural equation modelling*. R package version 2.21.13.
- Bollen, K. A. (1989). *Structural equations with latent variables*. Wiley.
- Bollen, K. A. (1996). An alternative two stage least squares (2SLS) estimator for latent variable equations. *Psychometrika*, 61, 109–121.
- Bollen, K., & Curran, P. (2006). *Latent curve models: A structural equation perspective*. Wiley.
- Bolotov, D. (2020). *neighbr: Classification, regression, clustering with K nearest neighbors*. R package version 1.0.3.
- Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144–152.
- Bottou, L., Curtis, F. E., & Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60, 223–311.
- Boyer, C. B. (1959). *The history of the calculus and its conceptual development*. Dover.
- Breiman, L., Friedman, J. H., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Wadsworth & Brooks.
- Breiman, L. (1995). Better subset regression using the nonnegative garrote. *Technometrics*, 37, 373–384.
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24, 123–140.
- Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics*, 26, 801–824.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.
- Buja, A., & Stuetzle, W. (2000). Bagging does not always decrease mean squared error. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=55ac2ffc39cdeb52fe71f312ab6bb893cbba0337>
- Camargo, A. (2022). PCAtest: Testing the statistical significance of principal component analysis in R. *PeerJ*, 10, e12967. <https://doi.org/10.7717/peerj.12967>
- Canete, L., Monroy, R., Medina-Pérez, M. A. (2021). A review and experimental comparison of multivariate decision trees. *IEEE Access*, 9, 1–1.
- Carrera-Fernández, M. J., Guàrdia-Olmos, J., & Peró-Cebollero, M. (2012). Qualitative methods of data analysis in psychology: An analysis of the literature. *Qualitative Research*, 0, 1–17.
- Casella, G., & Berger, R. L. (2002). *Statistical inference*. Duxbury.
- Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes rendus de l'Académie des Sciences Paris*, 25, 536–538.

- Chatfield, C., & Collins, A. J. (1980). *Introduction to multivariate analysis*. Chapman & Hall.
- Chatfield, C., & Xing, H. (2019). *The analysis of time series: An introduction with R*. Chapman & Hall.
- Cheng, B., & Titterton, D. M. (1994). Neural networks: A review from a statistical perspective. *Statistical Science*, 9, 2–54.
- Cheung, M. (2024). *mulSEM: Some multivariate analyses using structural equation modeling*. R package version 1.0.
- Chipman, H., George, E. I., & McCulloch, R. E. (2010). BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4, 266–298.
- Choi, J., Zou, H., & Oehlert, G. (2010). A penalized maximum likelihood approach to sparse factor analysis. *Statistics and its Inference*, 3, 429–436.
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates.
- Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum Associates Publishers.
- Cokluk, O., & Kocak, D. (2016). Using Horn's parallel analysis method in exploratory factor analysis for determining the number of factors. *Educational Sciences: Theory & Practice*, 16, 537–551.
- Collobert, R., Kavukcuoglu, K., & Farabet, C. (2010). *Torch7: A matlab-like environment for machine learning*. New York University.
- Comrey, A. L. (1962). The minimum residual method of factor analysis. *Psychological Reports*, 11, 15–18.
- Cortes, C., & Vapnik, V. (1995). Support-vector machines. *Machine Learning*, 20, 273–297.
- Cortina, J. M. (1993). What is coefficient alpha? An examination of theory and applications. *Journal of Applied Psychology*, 78, 98–104.
- Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13, 21–27.
- Cryer, J., & Chan, K. (2008). *Time series analysis: With applications in R*. Springer.
- Culp, M., Johnson, K., & Michailidis, G. (2016). *ada: The R package ada for stochastic boosting*. R package version 2.0-5.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2, 3–314.
- da Silva, A. R. (2021) *biotools: Tools for biometry and applied statistics in agricultural science*. R package version 4.2.
- Dasgupta, A., & Raftery, A. E. (1998). Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, 93, 294–302.
- Davies, T. M. (2016). *The book of R: A first course in programming and statistics*. No Starch Press.
- DeCarlo, L. T. (1997). On the meaning and use of kurtosis. *Psychological Methods*, 2, 292–307.
- Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.
- Delicado, P. (2001). Another look at principal curves and surfaces. *Journal of Multivariate Analysis*, 77, 84–116.
- de Leeuw, J., & Mair, P. (2009). Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software*, 31, 1–30.
- Demaine, E., Hesterberg, A., Koehler, F., Lynch, J., & Urschel, J. (2021). Multidimensional scaling: Approximation and complexity. *Proceedings of the 38th International Conference on Machine Learning*, PMLR 139.
- Demidenko, E. (2004). *Mixed models: Theory and applications*. Wiley.
- Denis, D., & Legerski, J. (2006). Causal modeling and the origins of path analysis. *Theory & Science*, 7, 2–10.
- Denis, D. (2020). *Univariate, bivariate, and multivariate statistics using R*. Wiley.
- Denis, D. (2021a). *Applied univariate, bivariate, and multivariate statistics*. Wiley.
- Denis, D. (2021b). *Applied univariate, bivariate, and multivariate statistics using Python*. Wiley.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 269–271.

- Draper, N. R., & Smith, H. (1998). *Applied regression analysis*. Wiley.
- Drton, M., Grosdos, A., Portakal, I., Sturma, N. (2024). Algebraic sparse factor analysis. <https://arxiv.org/pdf/2312.14762>
- Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503, 92–108.
- Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7, 1–26.
- Efron, G., & Hastie, T. (2016). *Computer age statistical inference: Algorithms, evidence, and data science*. Cambridge University Press.
- Einbeck, J., Darnell, R., & Hinde, J. (2018). *npmlreg: Nonparametric maximum likelihood estimation for random effect models*. R package version 0.46-5.
- Everitt, B. S., & Dunn, G. (2001). *Applied multivariate data analysis*. Wiley.
- Everitt, B. E. (2007). *An R and S-PLUS companion to multivariate analysis*. Springer.
- Fiedler, K., Schott, M., & Meiser, T. (2011). What mediation analysis can (not) do. *Journal of Experimental Social Psychology*, 47, 1231–1236.
- Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London, Series A, Containing Papers of a Mathematical or Physical Character*, 222, 309–368.
- Fisher, R. A. (1925/1992). *Statistical methods for research workers*. Springer.
- Fisher, R. A. (1935). *The design of experiments* (2nd ed.). Oliver and Boyd.
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, 179–188.
- Fix, E., & Hodges, J. L. (1951/1989). Discriminatory analysis: Non-parametric discrimination: Consistency properties. *International Statistical Review*, 57, 238–247.
- Fleischmann, M. (2023). Clustergram: Visualization and diagnostics for cluster analysis. *The Journal of Open Source Software*, 8, 5240.
- Flury, B. (1998). *A first course in multivariate statistics*. Springer.
- Fox, J. (2016). *Applied regression analysis and generalized linear models*. Sage.
- Fradkov, A. L. (2020). Early history of machine learning. *IFAC-PapersOnLine*, 53, 1385–1390.
- Frank, I. E., & Friedman, J. H. (1993). A statistical view of some chemometrics regression tools. *Technometrics*, 35, 109–135.
- Fraley, C., & Raftery, A. E. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis. *The Computer Journal*, 41, 578–588.
- Friedman, J. H. (1997). Data mining and statistics: What's the connection? In *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*.
- Friedman, J. H., & Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, 76, 817–823.
- Friedman, J. H., & Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23, 881–889.
- Friedman, J. H., Stuetzle, W., & Schroeder, A. (1984). Projection pursuit density estimation. *Journal of the American Statistical Association*, 79, 599–608.
- Friedman, J. H. (1989). Regularized discriminant analysis. *Journal of the American Statistical Association*, 84, 165–175.
- Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 19, 1–67.
- Friedman, J., Tibshirani, R., & Hastie, T. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33, 1–22.
- Friedman, J., Tibshirani, R., & Hastie, T. (2011). Regularization paths for Cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39, 1–13.
- Frühwirth-Schnatter, S., Hosszejni, D., & Lopes, H. F. (2024). Sparse Bayesian factor analysis when the number of factors is unknown. *Bayesian Anal. Advance Publication*, 1–97.
- Fryda, T., LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., et al. (2023). h2o: R interface for 488 the 'H2O' scalable machine learning platform. R package version 3.40.0.4.

- Galton, F. (1888). Co-relations and their measurement, chiefly from anthropometric data. *Proceedings of the Royal Society of London*, 45, 135–145.
- Garcia, S. R., & Horn, R. A. (2017). *A second course in linear algebra*. Cambridge.
- Gauss, C. F. (1799). *Demonstratio nova theorematis omnem functionem algebraicam ratio nalem integram unius variabilis in factores reales primi vel secundi gradus resolvi posse* [PhD thesis]. Universität Helmstedt. In *Werke III*, 1–30.
- Ge, L., Wang, J., Shi, C., Wu, Z., & Song, R. (2023). A reinforcement learning framework for dynamic mediation analysis. *Proceedings of the 40th International Conference on Machine Learning, PMLR 202: 11050–11097*.
- Genolini, C., Falissard, B., Kiener, P. (2024). *kml: K-means for longitudinal data*. R package version 2.5.0.
- Géron, A. (2017). *Hands-on machine learning with scikit-learn & TensorFlow*. O'Reilly.
- Ghahramani, Z., & Hinton, G. E. (1997). The EM algorithm for mixtures of factor analyzers. *Technical Report CRG-TR-96-1*.
- Gigerenzer, G. (2003). *Calculated risks: How to know when numbers deceive you*. Simon and Schuster.
- González, I., Déjean, S. (2023). *CCA: Canonical correlation analysis*. R package version 1.2.2.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.
- Grassi, M., Palluzzi F., & Tarantino, B. (2024). *SEMgraph: Network analysis and causal inference through structural equation modeling*. R package version 1.2.2.
- Greifer, N. (2023). *fwb: Fractional weighted bootstrap*. R package version 0.2.0.
- Gu, C. (2014). Smoothing spline ANOVA models: R package gss. *Journal of Statistical Software*, 58, 1–25.
- Günther, F., & Fritsch, S. (2010). neuralnet: Training of neural networks. *The R Journal*, 2, 30–38.
- Gutttag, J. V. (2013). *Introduction to computation and programming using Python*. The MIT Press.
- Hahsler, M., Buchta, C., Gruen, B., & Hornik, K. (2024). *arules: Mining association rules and frequent itemsets*. R package version 1.7–8.
- Hastie, T., & Efron, B. (2022). *lars: Least angle regression, lasso and forward stagewise*. R package version 1.3.
- Hastie, T., Qian, J., & Tay, K. (2023). *An introduction to glmnet*. <https://cran.r-project.org/web/packages/glmnet/vignettes/glmnet.pdf>
- Hastie, T., & Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association*, 84, 502–516.
- Hastie, T., & Tibshirani, R. (1986). Generalized additive models. *Statistical Science*, 1, 297–310.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction (12th printing 2017)*. Springer.
- Haussler, D. (1990). Probably approximately correct learning. *AAAI-90 Proceedings*.
- Hayes, A. F. (2018). *Introduction to mediation, moderation, and conditional process analysis*. The Guildford Press.
- Hayfield, T., & Racine, J. F. (2008). Nonparametric econometrics: The np Package. *Journal of Statistical Software*, 27, 1–32.
- Hays, W. (1994). *Statistics*. Cengage Learning.
- Hebb, D. O. (2002). *The organization of behavior: A neuropsychological theory*. Psychology Press.
- Hebbali, A. (2020). *blorr: Tools for developing binary logistic regression models*. R package version 0.3.0.
- Helwig, N. E (2022). *ica: Independent component analysis*. R package version 1.0-3.
- Hinton, G., & Roweis, S. (1998). Stochastic neighbor embedding. *Advances in Neural Information Processing Systems*, 15 (NIPS 2002).
- Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12, 55–67.
- Højsgaard, S., Halekoh, U., & Yan J. (2006) *The R Package geepack for Generalized Estimating Equations* *Journal of Statistical Software*, 15, 2, 1–11.
- Holst, K., & Budtz-Jorgensen, E. B. (2020). A two-stage estimation procedure for non-linear structural equation models. *Biostatistics*, 21, 676–691.
- Horn, J. (1965). A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30, 179–185.

- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359–366.
- Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4, 251–257.
- Huang, F. (2022). *hornpa: Horn's (1965) test to determine the number of components/factors*. R package version 1.1.0.
- Hubbard, J. H., & Hubbard, B. B. (2002). *Vector calculus, linear algebra, and differential forms: A unified approach*. Prentice Hall.
- Hyvärinen, A., & Oja, E. (2000). Independent components analysis: Algorithms and applications. *Neural Networks*, 13, 411–430.
- Ihler, A. (2003). Nonlinear manifold learning. [https://www.mit.edu/~6.454/www\\_fall\\_2003/ihler/summary.pdf](https://www.mit.edu/~6.454/www_fall_2003/ihler/summary.pdf)
- Imai, K., Keele, L., & Yamamoto, T. (2010). Identification, inference and sensitivity analysis for causal mediation effects. *Statistical Science*, 25, 51–71.
- Imdad, M. U., & Aslam, M. (2018). *lmridge: Linear ridge regression with ridge penalty and ridge statistics*. <https://cran.r-project.org/package=lmridge>
- Izenman, A. J. (1975). Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5, 248–264.
- Izenman, A. J. (2008). *Modern multivariate statistical techniques: Regression, classification, and manifold learning (2nd printing 2013)*. Springer.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning (8th printing, 2017)*. Springer.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning*. Springer.
- Japkowicz, N., & Gluck, J. (2000). Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, 12, 531–545.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). *Caffe: Convolutional architecture for fast feature embedding*. <https://arxiv.org/abs/1408.5093>
- Jiang, H., & Guntuboyina, A. (2021). A nonparametric maximum likelihood approach to mixture of regression. <https://arxiv.org/abs/2108.09816>
- Johnson, R. A., & Wichern, D. W. (2007). *Applied multivariate statistical analysis*. Prentice Hall.
- Jolliffe, I. T. (1982). *Principal components analysis*. Springer.
- Jolliffe, I. T. (2002). *Principal components analysis*. Springer.
- Jones, P. W., Osipov, A., & Rokhlin, V. (2011). Randomized approximate nearest neighbors algorithm. *Proceedings of the National Academy of Sciences*, 108, 15679–15686.
- Jöreskog, K. G. (1970). A general method for analysis of covariance structures. *Biometrika*, 57, 239–251.
- Jöreskog, K. G. (1973). A general method for estimating a linear structural equation system. In A. S. Goldberger & O. D. Duncan (Eds.), *Structural equation models in the social sciences* (pp. 85–112). Seminar.
- Judd, C. M., Kenny, D. A., & McClelland, G. H. (2001). Estimating and testing mediation and moderation in within-subject designs. *Psychological Methods*, 6, 115–134.
- Kalantan, Z. I., & Alqahtani, N. A. (2019). A study of principal components analysis for mixed data. *International Journal of Advanced and Applied Sciences*, 6, 99–104.
- Karatoglou, A., Smola, A., & Hornik, K. (2024). *kernlab: Kernel-based machine learning lab*. R package version 0.9–32.
- Karhunen, K. (1947). Über lineare methoden in der wahrscheinlichkeitsrechnung. *Annales Academiae Scientiarum Fennicae Series A1. Mathematica-Physica*, 37, 3–79.
- Kassambara, A., & Mundt, F. (2017). *Extract and visualize the results of multivariate data analyses*. R package version 1.0.5.
- Kelava, A., & Brandt, H. (2014). A general non-linear multilevel structural equation mixture model. *Frontiers in Psychology*, 18, 1–16.
- Kenny, D., & Judd, C. M. (1984). Estimating the nonlinear and interactive effects of latent variables. *Psychological Bulletin*, 96, 201–210.



- King, S. L. (1999). Neural networks vs. multiple linear regression for estimating previous diameter. *Proceedings, 12th Central Hardwood Forest Conference*.
- Kirk, R. E. (1995). *Experimental design: Procedures for the behavioral sciences*. Wadsworth Publishing.
- Klein, A. G., & Muthén, B. O. (2007). Quasi-maximum likelihood estimation of structural equation models with multiple interaction and quadratic effects. *Multivariate Behavioral Research*, 42, 647–673.
- Klein, A., & Moosbrugger, H. (2000). Maximum likelihood estimation of latent interaction effects with the LMS method. *Psychometrika*, 65, 457–474.
- Kleiner, A., Talwalkar, A., Sarkar, P., & Jordan, M. I. (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 76, 795–816.
- Kline, M. (1972). *Mathematical thought from ancient to modern times* (Vol. 3). Oxford University Press.
- Klinke, S., & Grassmann, J. (2000). Projection pursuit regression. In M. G. Schimek (Ed.), *Smoothing and regression: Approaches, computation, and application*. Wiley.
- Knuth, D. E. (1972). Ancient babylonian algorithms. *Communications of the ACM*, 15, 671–677.
- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biol. Cybern*, 43, 59–69.
- Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78, 1464–1480.
- Kohonen, T. (1995). *Self-organizing maps*. Springer.
- Kolmogorov, A. (1957). The representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSR*, 114, 953–956.
- Kratsios, A., & Bilokopytov, E. (2020). *Non-euclidean universal approximation*. 34th Conference on Neural Information Processing Systems.
- Kroese, D. P., Botev, Z. I., Taimre, T., & Vaisman, R. (2020). *Data science and machine learning: Mathematical and statistical methods*. CRC Press.
- Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a non-metric hypothesis. *Psychometrika*, 29, 1–27.
- Kruskal, J. B. (1969). Towards a practical method which helps uncover the structure of a set of observations by finding the line transformation which optimizes a new “index of condensation. In R. C. Milton & J. A. Nelder (Eds.), *Statistical Computation* (pp. 427–440). Academic Press.
- Kruskal, J. B. (1972). Linear transformation of multivariate data to reveal clustering. In R. N. Shepard, A. K. Romney, S. B. Nerlove (Eds.), *Multidimensional scaling: Theory and application in the behavioral sciences, I, theory*. Seminar Press.
- Krijthe, J. H. (2015). *Rtsne: T-distributed stochastic neighbor embedding using a barnes-hut implementation*. R package version 0.17.
- Kuhn, M. (2008). Building Predictive Models in R Using the caret package. *Journal of Statistical Software*, 28(5), 1–26.
- Kuhn, M. (2024). *baguette: Efficient model functions for bagging*. R package version 1.0.2.
- Laa, U., & Cook, D. (2020). Using tours to visually investigate properties of new projection pursuit indexes with application to problems in physics. *Computational Statistics*, 35, 1171–1205.
- Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6, 1783–1816.
- Lawson, R. G., & Jurs, P. C. (1990). New index for clustering tendency and its applications to chemical problems. *Journal of Chemical Information and Computer Sciences*, 30, 36–41.
- Lederer, J. (2021). Activation functions in artificial neural networks: A systematic overview. ArXiv, abs/2101.09957.
- Lee, J. A., & Verleysen, M. (2005). Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67, 29–53.
- Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. Courcier.
- Leisch, F., & Dimitriadou, E. (2024). *mlbench: Machine learning benchmark problems*. R package version 2.1-5.
- Lent, B., Swami, A., & Widom, J. (1997). Clustering association rules. *Proceedings of the Thirteenth International Conference on Data Engineering*, 220–231.
- Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge University Press.

- Li, J., Hu, K., Lu, X., Liu, W., & Julie Zhu, L. (2024). *StepReg: Stepwise regression analysis*. R package version 1.5.1.
- Liao, L., Chen, Y., Yang, Z., Dai, B., Wang, Z., & Kolar, M. (2020). Provably efficient neural estimation of structural equation model: An adversarial approach. *ArXiv*, abs/2007.01290
- Liaw, A., & Wiener, M. (2002). Classification and regression by *randomForest*. *R News*, 2, 18–22.
- Likert, R. (1932). A technique for measurement of attitudes. *Archives of Psychology*, 140, 5–55.
- Liland, K., Mevik, B., & Wehrens, R. (2023). *pls: Partial least squares and principal component regression*. R package version 2.8-3.
- Lippi, V., & Ceccarelli, G. (2019). Incremental principal component analysis exact implementation and continuity corrections. *Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics*, 473–480.
- Little, T. D., Card, N. A., Bovaird, J. A., Preacher, K. J., & Crandall, C. S. (2007). Structural equation modeling of mediation and moderation with contextual factors. In T. D. Little, J. A. Bovaird, & N. A. Card (Eds.), *Modeling contextual effects in longitudinal studies* (pp. 207–230). Lawrence Erlbaum Associates Publishers.
- Liu, T., Lugosi, G., Neu, G., & Tao, D. (2017). *Algorithmic stability and hypothesis complexity*. Proceedings of the 34th International Conference on Machine Learning, Australia.
- Liu, J. (2023). *Examination of nonlinear longitudinal processes with latent variables, latent processes, latent changes, and latent classes in the structural equation modeling framework*. The R package nlsem.
- Locantore, N., Marron, J. S., Simpson, D. G., Tripoli, N., Zhang, J. T., Cohen, K. L., Boente, G., Fraiman, R., Brumback, B., Croux, C., Fan, J., Kneip, A., Marden, J. I., Peña, D., Prieto, J., Ramsay, J. O., Valderrama, M. J., Aguilera, A. M., Locantore, N., ... Cohen, K. L. (1999). Robust principal component analysis for functional data. *Springer Nature Link*, 8, 1–73.
- Lucci, S., & Kopec, D. (2016). *Artificial intelligence in the 21st century: A living introduction*. Mercury Learning and Information.
- Lyubchich, V., Gel, Y., Vishwakarma, S. (2023). *funtimes: Functions for time series analysis*. R package version 9.1.
- MacKinnon, D. P., Fairchild, A. J., & Fritz, M. S. (2007). Mediation analysis. *Annual Review of Psychology*, 58, 593–614.
- Madhulatha, T. S. (2012). An overview of clustering methods. *IOSR Journal of Engineering*, 2, 719–725.
- Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2023). *cluster: Cluster analysis basics and extensions*. R package version 2.1.6.
- Mahalanobis, P. C. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Science of India*, 12, 49–55.
- Mahmud, J. (2017). Item response theory: A basic concept. *Educational Research and Reviews*, 12, 258–266.
- Mair, P. (2018). *Modern psychometrics with R*. Springer.
- Majka, M. (2024). *naivebayes: High performance implementation of the naive bayes algorithm in R*. R package version 1.0.0.
- Marden, J. I. (2015). *Multivariate statistics: Old school*. CreateSpace Independent Publishing Platform.
- Matloff, N. (2011). *The art of R programming: A tour of statistical software design*. No Starch Press.
- Matloff, N., Cheng, X., Mohanty, P., Khomtchouk, B., Kotila, M., Yancey, R., Tucker, R., Zhao, A., Jiang, T. (2022). *polyreg: Polynomial regression*. R package version 0.8.0.
- May, R. (2015). *The meaning of anxiety*. W.W. Norton & Company.
- McCullagh, P., & Nelder, J. A. (1989). *Generalized linear models*. Chapman and Hall.
- McDonald, R. P. (1999). *Test theory: A unified treatment*. Routledge.
- McLachlan, G. J., & Chang, S. U. (2004). Mixture modelling for cluster analysis. *Statistical Methods in Medical Research*, 13, 347–361.
- Meehl, P. E. (1978). Theoretical risks and tabular asterisks: Sir Karl, Sir Ronald, and the slow progress of soft psychology. *Journal of Consulting and Clinical Psychology*, 46, 806–834.
- Meilä, M., & Zhang, H. (2023). Manifold learning: What, how, and why. *ArXiv*, abs/2311.03757.
- Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, 209, 415–446.

- Mevik, B., & Wehrens, R. (2024). Introduction to the pls package. <https://cran.r-project.org/web/packages/pls/vignettes/pls-manual.pdf>
- Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F. (2023). *e1071: Misc functions of the department of statistics, probability theory group (Formerly: E1071), TU Wien*. R package version 1.7-14.
- Min, E. J., Chi, E. C., & Zhou, H. (2020). Tensor canonical correlation analysis. *Stat*, 8, e253.
- Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. MIT Press.
- Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58, 415–434.
- Mueller, M. (2016). *gplm: Generalized Partial Linear Models (GPLM)*. R package version 0.7-4.
- Muggleton, S. (1991). Inductive logic programming. *New generation computing*, 8, 295–318.
- Mulaik, S. (1987). A brief history of the philosophical foundations of exploratory factor analysis. *Multivariate Behavioral Research*, 22, 267–305.
- Müller, A. C., & Guido, S. (2017). *Introduction to machine learning with Python: A guide for data scientists*. O'Reilly.
- Murray, K. P. (2022). *Probabilistic machine learning: An introduction*. The MIT Press.
- Muthén, B., & Kaplan, D. (1985). A comparison of some methodologies for the factor analysis of non-normal likert variables. *British Journal of Mathematical and Statistical Psychology*, 38, 171–189.
- Muthén, L. K., & Muthén, B. O. (2017). *Mplus user's guide* (8th ed.). Muthén & Muthén.
- Nadaraya, E. A. (1964). On estimating regression: Theory of probability & its applications. *Society for Industrial and Applied Mathematics*, 9, 141–142.
- Nanga, S., Bawah, A. T., Acquaye, B. A., Billa, M., Baeta, F. D., Odai, N. A., Obeng, S. K., & Nsiah, A. D. (2021). Review of dimension reduction methods. *Journal of Data Analysis and Information Processing*, 9, 189–231.
- Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized linear models. *Journal of the Royal Statistical Society, Series A (General)*, 135, 370–384.
- Neter, J., Kutner, M. H., Nachtsheim, C. L., & Wasserman, W. (1996). *Applied linear statistical models*. McGraw-Hill.
- Neykov, M. (2019). Isotonic regression meets lasso. *Electronic Journal of Statistics*, 13, 710–746.
- Nguéma, E. N., Nono, B. F., & Gwét, H. (2021). Nonparametric estimation in linear mixed models with uncorrelated homoscedastic errors. *Open Journal of Statistics*, 11, 558–605.
- Novikoff, A. B. (1963). *On convergence proofs for perceptrons*. Technical Report 298 268. Stanford Research Institute.
- Nowlan, S. J., & Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4, 473–493.
- Okada, K. (2017). Negative estimate of variance-accounted-for effect size: How often is it obtained, and what happens if it is treated as zero. *Behavior Research*, 49, 979–987.
- Oken, A. (2017). An introduction to and applications of neural networks. <https://www.semanticscholar.org/paper/An-Introduction-To-and-Applications-of-Neural-Oken-May/5185bad2bad10c86688179aa01c0326f9c3120aa>
- Olejnik, S., & Algina, J. (2003). Generalized eta and omega squared statistics: Measures of effect size for some common research designs. *Psychological Methods*, 8, 434–447.
- Oliveira, M. C. F., & Levkowitz, H. (2003). From visual data exploration to visual data mining: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 9, 378–394.
- Original, S., & StatLib, F. (2019). *bootstrap: Functions for the book “an introduction to the bootstrap.”* R package version 2019.6.
- Ott, R. L., & Longnecker, M. (2016). *An introduction to statistical methods & data analysis*. Cengage Learning.
- Ozertem, U., & Erdogmus, D. (2011). Locally defined principal curves and surfaces. *Journal of Machine Learning Research*, 12, 1249–1286.
- Padgham, M., & Sumner, M. (2024). *geodist: Fast, dependency-free geodesic distance calculations*. R package version 0.1.0.
- Panayides, P. (2013). Coefficient alpha: Interpret with caution. *Europe's Journal of Psychology*, 9, 687–696.



- Paradis, E., & Schliep, K. (2019). ape 5.0: An environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35, 526–528.
- Pateras, J., Rana, P., & Ghosh, P. (2023). A taxonomic survey of physics-informed machine learning. *Applied Sciences*, 13, 6892.
- Pearl, J. (2012). *The causal foundations of structural equation modeling*. In R. H. Hoyle (Ed.), *Handbook of structural equation modeling*. The Guilford Press.
- Pearl, J. (2014). Interpretation and identification of causal mediation. *Psychological Methods*, 19, 459–481.
- Peres-Neto, P. R., Jackson, D. A., & Somers, K. M. (2005). How many principal components? Stopping rules for determining the number of non-trivial axes revisited. *Computational Statistics and Data Analysis*, 49, 974–997.
- Perperoglou, A., Sauerbrei, W., Abrahamowicz, M., & Schmid, M. (2019). A review of spline function procedures in R. *BMC Medical Research Methodology*, 19, 46.
- Perrone, M., & Cooper, L. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone (Ed.), *Neural networks for speech and image processing*.
- Peters, A., & Hothorn, T. (2023). *ipred: Improved predictors*. R package version 0.9-14.
- Pinheiro, J., Bates, D., & R Core Team. (2023). *nlme: Linear and nonlinear mixed effects models*. R package version 3.1-164.
- Plaut, E. (2018). From principal subspaces to principal components with linear autoencoders. arXiv preprint arXiv:1804.10253.
- Posse, C. (1995). Tools for two-dimensional exploratory projection pursuit. *Journal of Computational and Graphical Statistics*, 4, 83–100.
- Prentice, D. A., & Miller, D. T. (1992). When small effects are impressive. *Psychological Bulletin*, 112, 160–164.
- Qin, X. (2024). *moderate.mediation: Causal moderated mediation analysis*. R package version 0.0.10.
- Rdz-Navarro, K., & Yang-Wallentin, F. (2020). Specification issues in nonlinear SEM: The moderation that wasn't. *Psicothema*, 32, 115–121.
- Rencher, A. C., & Christensen, W. F. (2012). *Methods of multivariate analysis*. Wiley.
- Revelle, W. (2024). *psych: Procedures for psychological, psychometric, and personality research*. Northwestern University, Evanston, Illinois. R package version 2.4.3.
- Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). *Stochastic backpropagation and approximate inference in deep generative models*. Proceedings of the 31st International Conference on Machine Learning.
- Ridgeway, G., Developers, G. (2024). *gbm: Generalized boosted regression models*. R package version 2.2.2.
- Riedmiller, M., & Braun, H. (1993). *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*. IEEE International Conference on Neural Networks.
- Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons: From backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16, 265–278.
- Ripley, B. (1996). *Pattern recognition and neural networks*. Cambridge University Press.
- Ripley B (2023). *tree: Classification and regression trees*. R package version 1.0-43.
- Rizzo, M. L. (2019). *Statistical computing with R*. CRC Press.
- Rodgers, J. L., & Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42, 59–66.
- Rodosthenous, T., Shahrezaei, V., & Evangelou, M. (2024). Multi-view data visualisation via manifold learning. *PeerJ - Computer Science*, 10, e1993. <https://doi.org/10.7717/peerj-cs.1993>
- Roman, S. (2008). *Advanced linear algebra*. Springer.
- Roosen, C., & Hastie, T. (1993). Logistic response projection pursuit regression. In *Statistics and data analysis research department*, AT&T Bell Laboratories, Document No. BL011214-930806-09TM.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386–408.
- Rosenblatt, F. (1959). Two theorems of statistical separability in the perceptron. In *Proceedings of a symposium on the mechanization of thought processes* (pp. 421–456). Her Majesty's Stationary Office.
- Rosenblatt, F. (1962). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books.

- Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, 48, 1–36.
- Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53–65.
- Roweis, S. (1997). *EM algorithms for PCA and SPCA*. Proceedings of the 10th International Conference on Neural Information Processing Systems, 626–632.
- Roweis, S.T. (1998). EM algorithms for PCA and SPCA . In Jordan, M., Kearns, M., & Solla, S. (1997). *Advances in Neural Information Processing Systems*, 10, 626–632. MIT Press. Also Tech Report CNS-TR-97-02, Computation and Neural Systems, Institute of Technology.
- Roweis, S., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290, 2323–2326.
- Rubin, D. B., & Thayer, D. T. (1982). EM algorithms for ML factor analysis. *Psychometrika*, 47, 69–76.
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. ArXiv, abs/1609.04747.
- Sabanes, B., Li, L., Dedic, J., Kelkhoff, D., Kunzmann, K., Lang, B., Stock, C., Wang, Y., James, D., Sidi, J., Leibovitz, D., & Sjöberg, D. (2024). *mmrm: Mixed models for repeated measures*. R package version 0.3.11.
- Saha, A. (2015). *Doing math with python*. No Starch Press.
- Salas, S. L., Hille, E., & Etgen, G. L. (1998). *Calculus: One and several variables*. Wiley.
- Sandilya, S., & Kulkarni, S. R. (2002). Principal curves with bounded turn. *IEEE Transactions on Information Theory*, 48, 2789–2793.
- Sarle, W. S. (1994). *Neural networks and statistical methods*. Proceedings of the Nineteenth Annual SAS Users Group International Conference.
- Saul, L. K., & Roweis, S. T. (2003). Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4, 119–155.
- Savino, M. (2024). *absorber: Variable selection in nonparametric models using B-splines*. R package version 1.0.
- Schapire, R. (2001). *The boosting approach to machine learning: An overview*. MSRI Workshop on Nonlinear Estimation and Classification.
- Scheffé, H. (1959). *The analysis of variance*. Wiley.
- Schölkopf, B., Smola, A., & Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10, 1299–1319.
- Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6, 461–464.
- Schubert, E., Sander, J., Ester, M., Kriegel, H., & Xu, X. (2017). DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, 42, 19.
- Schuetzenmeister, A., & Dufey, F. (2024). *VCA: Variance component analysis*. R package version 1.5.1.
- Scrucca, L., Fraley, C., Murphy, T. B., & Raftery, A. E. (2023). *Model-based clustering, classification, and density estimation using mclust in R*. Chapman and Hall/CRC.
- Serneels, S., & Verdonck, T. (2008). Principal component analysis for data containing outliers and missing elements. *Computational Statistics and Data Analysis*, 52, 1712–1727.
- Shao, J. (2023). *Mathematical statistics*. Springer.
- Siegel, S., & Castellan Jr, N. J. (1988). *Nonparametric statistics for the behavioral sciences*. McGraw-Hill.
- Sijtsma, K. (2009). On the use, the misuse, and the very limited usefulness of Cronbach’s alpha. *Psychometrika*, 74, 107–120.
- Sikhakolli, S. S., & Sikhakolli, A. K. (2023). Effective purity method for measuring the clustering accuracy and its illustration. *International Journal of Computer Applications*, 185, 28–33.
- Simovici, D., & Hua, K. (2019). Data ultrametricity and clusterability. *Journal of Physics: Conference Series*, 1334, 012002.
- Slupphaug, S. K. (2024). *modsem: Latent Interaction (and Moderation) analysis in Structural Equation Models (SEM)*. R package version 0.1.4.
- Smith, L. (2002). *A tutorial on principal components analysis*. [https://www.iro.umontreal.ca/~pift6080/H09/documents/papers/pca\\_tutorial.pdf](https://www.iro.umontreal.ca/~pift6080/H09/documents/papers/pca_tutorial.pdf)
- Sokol, S. (2024). *bspline: B-spline interpolation and regression*. R package version 2.2.2.

- Sol, R., & Baras, K. (2022). *An integrated neural network and structural equation modeling approach for modeling activity trackers use*. Proceedings of the 6th International Conference on Computer-Human Interaction Research and Applications, 49–58.
- Song, L., Anandkumar, A., Dai, B., Xie, B. (2014). Nonparametric estimation of multi-view latent variable models. *Proceedings of the 31st International Conference on Machine Learning, Proceedings of Machine Learning Research*, 32, 640–648.
- Spearman, C. (1904). General intelligence objectively determined and measured. *American Journal of Psychology*, 15, 201–292.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.
- Steffens, K. (2006). *The history of approximation theory: From Euler to Bernstein*. Birkhäuser.
- Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, 103, 677–680.
- Stigler, S. M. (1986). *The history of statistics: The measurement of uncertainty before 1900*. Belknap Press.
- Stillwell, J. (2016). *Elements of mathematics: From Euclid to Gödel*. Princeton University Press.
- Stoffi, F., Cevolani, G., & Gnecco, G. (2022). Simple models in complex worlds: Occam’s razor and statistical learning theory. *Minds and Machines*, 32, 13–42.
- Strang, G. (1993). *Introduction to linear algebra*. Wellesley-Cambridge Press.
- Tanton, J. (2005). *Encyclopedia of mathematics*. Facts on File.
- Tarka, P. (2018). An overview of structural equation modeling: Its beginnings, historical development, usefulness and controversies in the social sciences. *Quality & Quantity*, 52, 313–354.
- Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290, 2319–2323.
- Therneau, T., & Atkinson, B. (2023). *rpart: Recursive partitioning and regression trees*. R package version 4.1.23.
- Thisted, R. A. (1976). Ridge regression, minimax estimation and empirical bayes methods. *Technical Report* 28.
- Thrun, M. C., & Ultsch, A. (2024). Using projection based clustering to find distance and density based clusters in high-dimensional data. *Journal of Classification*, 38, 280–312.
- Thurstone, L. L. (1954). The measurement of values. *Psychological Review*, 61, 47–58.
- Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the Gap statistic. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 63, 411–423.
- Tingley, D., Tappei, K., H., Keele, L., & Kosuke, I. (2014). *mediation*: R package for causal mediation analysis. *Journal of Statistical Software*, 59, 1–38.
- Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society Series B*, 61, 611–622.
- Torgerson, W. S. (1952). Multidimensional scaling I. Theory and method. *Psychometrika*, 17, 401–419.
- Torgerson, W. S. (1958). *Theory and methods of scaling*. Wiley.
- Trosset, M.W., & Buyukbas, G. (2020). Rehabilitating Isomap: Euclidean Representation of Geodesic Structure. *ArXiv, abs/2006.10858*.
- Turner, R. (2023). *Iso: Functions to perform isotonic regression*. R package version 0.0-21.
- Ullah, M. I., Aslam, M., & Altaf, S. (2018). lmrige: A comprehensive R package for ridge regression. *The R Journal*, 10, 326–346.
- Umbach, N., Naumann, K., Brandt, H., & Kelava, A. (2017). Fitting nonlinear structural equation models in R with package nlsem. *Journal of Statistical Software*, 77, 1–20.
- Upton, G., & Cook, I. (2014). *Oxford dictionary of statistics*. Oxford University Press.
- Urban, C. J., & Bauer, D. J. (2021). A deep learning algorithm for high-dimensional exploratory item factor analysis. *Psychometrika*, 86, 1–29.
- Valiant, L. (2013). *Probably approximately correct: Nature’s algorithms for learning and prospering in a complex world*. Basic Books.
- Vallat, R. (2018). Pingouin: Statistics in python. *Journal of Open Source Software*, 31, 1026.
- van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605.

- Vapnik, V. N., & Chervonenkis, A. YA. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its applications*, XVI, 264–280.
- Vegelius, J., & Jin, S. (2021). A semiparametric approach for structural equation modeling with ordinal data. *Structural Equation Modeling: A Multidisciplinary Journal*, 28, 497–505.
- Vegelius, J. (2022). Estimation of nonlinear latent variable and mixture models. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Social Sciences*, 196.
- Venables, W. N., & Ripley, B. D. (2002) *Modern applied statistics with S* (4th ed.). Springer.
- Wakita, T., Ueshima, N., & Noguchi, H. (2012). Psychological distance between categories in the likert scale: Comparing different numbers of options. *Educational and Psychological Measurement*, 72, 533–546.
- Wald, P., & Kronmal, R. (1977). Discriminant functions when covariance are unequal and sample sizes are moderate. *Biometrics*, 33, 479–484.
- Wang, W., Yan, X., Lee, H., & Livescu, K. (2017). Deep variational canonical correlation analysis. <https://arxiv.org/abs/1610.03454>
- Warner, B., & Misra, M. (1996). Understanding neural networks as statistical tools. *The American Statistician*, 50, 284–293.
- Wickham, W., Cook, D., Hofman, H., & Buja, A. (2011). tourr: An R package for exploring multivariate data with projections. *Journal of Statistical Software*, 40, 1–18.
- Wickham, H., Averick, M., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., & Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4, 1686.
- Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record*, 4, 96–104.
- Winkler, R. L. (2003). *An introduction to Bayesian inference and decision*. Probabilistic Publications.
- Witten, D. M., Tibshirani, R. (2018). *sparcl: Perform sparse hierarchical clustering and sparse K-means clustering*. R package version 1.0.4.
- Wold, S. (1978). Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20, 397–405.
- Wood, S. N. (2017). *Generalized additive models: An introduction with R* (2nd ed.). Chapman and Hall/CRC.
- Wright, S. (1918). On the nature of size factors. *Genetics*, 3, 367–74.
- Wright, K. (2022). Will the real hopkins statistic please stand up? *The R Journal*, 14, 282–292.
- Wright, K. (2023). *Hopkins: Calculate Hopkins statistic for clustering*. R package version 1.1.
- Wuertz, D., Setz, T., Chalabi, Y., & Boshnakov, G. N. (2024). *TimeSeries: Financial time series objects (Rmetrics)*. R package version 4041.111.
- Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2, 165–193.
- Yang, M., & Xu, S. (2021). Orthogonal nonnegative matrix factorization using a novel deep autoencoder network. *Knowledge-Based Systems*, 227, 107236.
- Yin, H. (2008). On multidimensional scaling and the embedding of self-organizing maps. *Neural Networks*, 21, 160–169.
- Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. *Deep Learning Workshop, 31st International Conference on Machine Learning*. <https://arxiv.org/abs/1506.06579>
- You, K., Shung, D. (2022). Rdimtools: An R package for dimension reduction and intrinsic dimension estimation. *Software Impacts*, 14.
- Young, B. (2023). *Using geometry and visualization to demystify eigen decomposition in the classroom: An analytical and pedagogical approach* [Doctoral dissertation, University of Montana]. University of Montana Electronic Theses, Dissertations, Professional Papers.
- Zhang, H., & Ye, Y. (2008). A tree-based method for modeling a multivariate ordinal response. *Statistics and Its Interface*, 1, 169–178.
- Zhang, B. (2023). Inverse probability weighting estimation in completely randomized experiments. *The American Statistician*, 78, 26–35.

- Zhang, Z., & Yuan, K. (2023). *Coefficientalpha: Robust coefficient alpha and omega with missing and non-normal data*. R package version 0.7.2.
- Zhou, M., & Shao, Y. (2014). A powerful test for multivariate normality. *Journal of Applied Statistics*, 41, 351–363.
- Zou, H., Hastie, T., & Tibshirani, R. (2004). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15, 265–286.

# Index

- activation functions (in neural networks) 540–1, 546
  - saturating vs. non-saturating 542
- Akaike's information criterion (AIC) 235–6
- algorithm
  - definition 25
  - stability of 25
- analysis of variance 185–97
  - factorial 197–200
  - fixed vs. random effects 200, 201, 205–6
  - repeated-measures 209
- approximation (to the normal distribution) 84
- artificial intelligence 20
- association rules 501
- assumptions
  - analysis of variance 197
  - regression 237
- autoencoders 428, 474
- backpropagation (in neural networks) 548–50
- bagging 380–2
- Bartlett's test of sphericity (in factor analysis) 468
- base rates 32
- basis
  - expansion 155, 297–9, 301
  - functions 297
  - orthogonal 150
- Bayes
  - factor 73
  - naïve 339–43
  - theorem 72
- Bayesian
  - information criterion (BIC) 236, 527
  - model selection 73
- bias-variance trade-off 95
- binomial (distribution) 83–4
- blocking 207
- boosting 388–9
- bootstrap 99–102
- Box-M test 213
- calculus 150
- canonical correlation 260–1
- causal modeling 480–1
  - history 480
- centering 115, 401, 421
- central limit theorem 109, 117
- central tendency 86–90
  - arithmetic mean 87
  - median 90
- chi-square 85, 489
- cluster analysis 498–529
  - DBSCAN 522–3
  - hierarchical 517
    - agglomerative 518–19
  - k-means 506–9
  - k-means ++ 510
  - k-medoids 510
  - mixture models 80, 492, 499, 525–7
  - validation of 528
- coefficient of variation 94
- collinearity 237–8, 247
- committee machines 76, 369, 379
- computers 24
- computational complexity (in kernel PCA) 420–2
- continuity 151
- contrasts (in ANOVA) 191–3
- convergence (of random variables) 77
- correlation 109–12, 173
- covariance 70, 109, 489
- cross-validation 55–6, 216, 236, 250, 281, 284, 301, 317, 320, 360, 377, 382, 550
- curse of dimensionality 53, 298, 346–8, 454
- data
  - science 5, 23
  - train vs. test 55, 180, 234
- decision trees 370–9
- degrees of freedom 51
- derivative (calculus) 152
- design
  - experimental vs. non-experimental 23
- determinant 134–8
- dimension reduction 52, 395, 418
- dimensionality 52–4, 144, 333, 348
- discriminant analysis 217, 327–39
  - linear 330

- multiclass 332
- quadratic 335
- regularized 336
- distance 52
  - Euclidean 147, 347, 430
  - geodesic 443
- distributions
  - multinomial 84
  - Poisson 85
  - univariate 79, 82
- dual representation (via kernel trick) 142–3
- effect size 104
  - absolute vs. relative 106
  - adjusted R-squared 234, 236
  - ANOVA 190
  - Cohen's  $d$  104
  - comparability across studies 107
  - R-squared 234
  - SEM 488
- eigen analysis 124
- encoder 427–8
  - vs. decoder 427
- epistemology 16–17
- error
  - minimization 54
  - training vs. test 55
- estimators
  - in factor analysis 465–6
  - unbiased 232
- expectation (of a random variable) 85
- factor analysis
  - confirmatory 484
  - exploratory 461
  - independent 473
  - item 476
  - mixtures 475
  - nonlinear 474
  - rotation of factors 466–8
    - quartimax 468
    - varimax 467
  - sparse 477
- fit
  - overfitting vs. underfitting 39–41, 346, 372, 384, 551
- functions
  - composite 155, 299, 536–7
  - convex vs. nonconvex 159
- Gauss-Markov theorem 232
- generalized
  - additive models 321, 324
  - linear models 309–10, 314
  - variance 137
- Gini index 374–5
- Gödel
  - incompleteness theorem 27
- gradient descent 160–1
- hidden nodes (in neural networks) 550
- hierarchical (modeling structures) 207
- Hotelling's  $T^2$ , 211
- hyperplane 351, 357
- independent components analysis 410
- infinite regress (of knowledge) 16
- inner product 139–40, 142, 362–3, 421
- instance learning 51
- integral (calculus) 162
- inverse (of a matrix) 137
- isomap 447–9
- justification
  - theoretical vs. empirical 30–1
- kernel
  - null space 131
  - regression 303–6
  - trick 140, 142–4, 420–2
- k-nearest neighbors 344–8
- kurtosis 92
- law of large numbers 75
- learners
  - strong vs. weak 368
- learning
  - bottom-up vs. top-down 8
  - parametric vs. non-parametric 49–50
  - supervised vs. unsupervised 29
- least-angle regression 284–5
- least-squares
  - geometry 231
- level of analysis 31
- linear combinations 132–3, 247, 260
  - correlating 113–14
- linear independence 140
- locally linear embedding 444–6
- logic
  - deductive vs. inductive 26–7
- logistic regression 311–13
- longitudinal (models) 209
- machine learning 5, 20
- manifold learning 441–3
- matrices
  - correlation 114
  - covariance 114
  - identity 127
  - orthogonal 142, 464
  - positive definite 139
- maximum likelihood (estimation) 96–8, 466



- mean squares (ANOVA) 188–9
- measurement
  - scales 42–5
- mediation 242
  - cautions about 246
- missing data 57
- moderated regression 244
- Monte Carlo methods 57
- multidimensional scaling 429–38
  - metric vs. nonmetric 434
- multilevel models 204–8
- multinomial 84–5
- multivariate analysis of variance (MANOVA) 210–17
- multivariate reduced-rank regression 259
  
- nesting 203–4, 206–8
- neural networks 298, 314, 324, 428, 531–63
  - as composite functions 536–7
  - caveats and warnings 558–60
  - rise and history 531–5
  - training of 548–9
- nonlinear 290–307, 325
- no free lunch theorem 48
- null hypothesis significance testing 102–4
  
- Occam’s razor 38, 73
- odds 311–12
  - ratio 312
- one-hot vector 121
- operationalization (of predictors) 108
- optimization 35–7, 150, 156
- orthogonality
  - in linear algebra 142
  - in PCA 402
- out-of-bag error 382
- overfitting 39–41
  - in decision trees 372
  - in neural networks 550–1
  
- packages (in R) 170
- paired data 118
- paradigms (Kuhnian) 3
- partial least-squares (PLS) 252–9
- path analysis 480–4
- perceptron 534
  - multilayer 546
- philosophy of science 13
- Poisson
  - distribution 85
  - regression 318–21
- polynomial regression 291–5
- positive definite (matrix) 139, 347
- principal components
  - analysis (PCA) 247, 396–407, 426–7
  - sensible (probabilistic) 475
  - regression 247–9
  - principal curves 425–7
- probabilistic principal components analysis (PPCA) 414–17
- probability
  - conditional 66
  - density functions 76–7
  - independence vs. dependence 68
  - interpretations 63–4
  - neglect 34
- probably approximately correct (PAC) 47–48
- projection 147–8
  - pursuit 555–8
- proofs (mathematical) 62–3
- p-value (interpretation) 105–6
- Python 176–81
  
- quadratic forms 139
  
- random forests 384–7
- rank (of a matrix) 141
- regression
  - multiple 226–30
  - Poisson 318–21
  - simple 219–24
- regularities
  - statistical 35
- regularization
  - concept of 41, 265–84, 552
- reliability 42
  - coefficient alpha 43
  - test-retest 44
- research
  - quantitative vs. qualitative 58
  
- self-organizing maps 439–40
- sensitivity 71–2
- significance tests
  - one-tailed, two-tailed 104
- singularity 138
- skewness 92–3
- specificity 71–2
- splines (regression) 299–302
- splitting (data) 180–81
- statistics
  - theoretical vs. applied 7–8, 23–4
- stochastic neighborhood embedding 449–53
- structural equation modeling 485–95
  - caveats about 494
  - measures of fit 487–91
  - nonlinear 491–2
  - nonparametric 492–4
- support vector machines 350–67
- symmetric (matrix) 115, 129
  
- textbooks (ways of writing one) 10
- time series 233
- trace (of a matrix) 132



- transpose
  - matrix 128
  - vector 121
- t-SNE 453–6
- t-tests 116–17
- universal approximation theorem 553–5
- validity 44–5
- variability 90
  - mean absolute deviation 92
  - standard deviation 91
- variables
  - exogenous vs. endogenous 214, 484
  - random 76
- variable selection (in regression) 238–9
- variance inflation factor (VIF) 237–8
- V-C dimension 49
- vector
  - addition 122
  - length 122
- vector spaces 121
- weakest link pruning 373
- Welch-Satterthwaite approximation 118
- zero group 34
- z-scores 79, 115–16