

Ivo D. Dinov

Data Science and Predictive Analytics

Biomedical and Health Applications
using R

EXTRAS ONLINE

 Springer

Data Science and Predictive Analytics

Ivo D. Dinov

Data Science and Predictive Analytics

Biomedical and Health Applications using R



Springer

Ivo D. Dinov
University of Michigan–Ann Arbor
Ann Arbor, Michigan, USA

Additional material to this book can be downloaded from <http://extras.springer.com>.

ISBN 978-3-319-72346-4 ISBN 978-3-319-72347-1 (eBook)
<https://doi.org/10.1007/978-3-319-72347-1>

Library of Congress Control Number: 2018930887

© Ivo D. Dinov 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature

The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

... dedicated to my lovely and encouraging wife, Magdalena, my witty and persuasive kids, Anna-Sophia and Radina, my very insightful brother, Konstantin, and my nurturing parents, Yordanka and Dimitar ...

Foreword

Instructors, formal and informal learners, working professionals, and readers looking to enhance, update, or refresh their interactive data skills and methodological developments may selectively choose sections, chapters, and examples they want to cover in more depth. Everyone who expects to gain new knowledge or acquire computational abilities should review the overall textbook organization before they decide what to cover, how deeply, and in what order. The organization of the chapters in this book reflects an order that may appeal to many, albeit not all, readers.

Chapter 1 (Motivation) presents (1) the DSPA mission and objectives, (2) several driving biomedical challenges including Alzheimer's disease, Parkinson's disease, drug and substance use, and amyotrophic lateral sclerosis, (3) provides demonstrations of brain visualization, neurodegeneration, and genomics computing, (4) identifies the six defining characteristics of big (biomedical and healthcare) data, (5) explains the concepts of *data science* and *predictive analytics*, and (6) sets the DSPA expectations.

Chapter 2 (Foundations of R) justifies the use of the statistical programming language *R* and (1) presents the fundamental programming principles; (2) illustrates basic examples of data transformation, generation, ingestion, and export; (3) shows the main mathematical operators; and (4) presents basic data and probability distribution summaries and visualization.

In **Chap. 3 (Managing Data in R)**, we present additional *R* programming details about (1) loading, manipulating, visualizing, and saving *R* Data Structures; (2) present sample-based statistics measuring central tendency and dispersion; (3) explore different types of variables; (4) illustrate scrapping data from public websites; and (5) show examples of cohort-rebalancing.

A detailed discussion of **Visualization** is presented in **Chap. 4** where we (1) show graphical techniques for exposing composition, comparison, and relationships in multivariate data; and (2) present 1D, 2D, 3D, and 4D distributions along with surface plots.

The foundations of **Linear Algebra and Matrix Computing** are shown in **Chap. 5**. We (1) show how to create, interpret, process, and manipulate

second-order tensors (matrices); (2) illustrate variety of matrix operations and their interpretations; (3) demonstrate linear modeling and solutions of matrix equations; and (4) discuss the eigen-spectra of matrices.

Chapter 6 (Dimensionality Reduction) starts with a simple example reducing 2D data to 1D signal. We also discuss (1) matrix rotations, (2) principal component analysis (PCA), (3) singular value decomposition (SVD), (4) independent component analysis (ICA), and (5) factor analysis (FA).

The discussion of machine learning model-based and model-free techniques commences in **Chap. 7 (Lazy Learning – Classification Using Nearest Neighbors)**. In the scope of the k-nearest neighbor algorithm, we present (1) the general concept of divide-and-conquer for splitting the data into training and validation sets, (2) evaluation of model performance, and (3) improving prediction results.

Chapter 8 (Probabilistic Learning: Classification Using Naive Bayes) presents the naive Bayes and linear discriminant analysis classification algorithms, identifies the assumptions of each method, presents the Laplace estimator, and demonstrates step by step the complete protocol for training, testing, validating, and improving the classification results.

Chapter 9 (Decision Tree Divide and Conquer Classification) focuses on decision trees and (1) presents various classification metrics (e.g., entropy, misclassification error, Gini index), (2) illustrates the use of the C5.0 decision tree algorithm, and (3) shows strategies for pruning decision trees.

The use of linear prediction models is highlighted in **Chap. 10 (Forecasting Numeric Data Using Regression Models)**. Here, we present (1) the fundamentals of multivariate linear modeling, (2) contrast regression trees vs. model trees, and (3) present several complete end-to-end predictive analytics examples.

Chapter 11 (Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines) lays out the foundation of Neural Networks as silicon analogues to biological neurons. We discuss (1) the effects of network layers and topology on the resulting classification, (2) present support vector machines (SVM), and (3) demonstrate classification methods for optical character recognition (OCR), iris flowers clustering, Google trends and the stock market prediction, and quantifying quality of life in chronic disease.

Apriori Association Rules Learning is presented in **Chap. 12** where we discuss (1) the foundation of association rules and the Apriori algorithm, (2) support and confidence measures, and (3) present several examples based on grocery shopping and head and neck cancer treatment.

Chapter 13 (k-Means Clustering) presents (1) the basics of machine learning clustering tasks, (2) silhouette plots, (3) strategies for model tuning and improvement, (4) hierarchical clustering, and (5) Gaussian mixture modeling.

General protocols for measuring the performance of different types of classification methods are presented in **Chap. 14 (Model Performance Assessment)**. We discuss (1) evaluation strategies for binary, categorical, and continuous outcomes; (2) confusion matrices quantifying classification and prediction accuracy; (3) visualization of algorithm performance and ROC curves; and (4) introduce the foundations of internal statistical validation.

Chapter 15 (Improving Model Performance) demonstrates (1) strategies for manual and automated model tuning, (2) improving model performance with meta-learning, and (3) ensemble methods based on bagging, boosting, random forest, and adaptive boosting.

Chapter 16 (Specialized Machine Learning Topics) presents some technical details that may be useful for some computational scientists and engineers. There, we discuss (1) data format conversion; (2) SQL data queries; (3) reading and writing XML, JSON, XLSX, and other data formats; (4) visualization of network bioinformatics data; (4) data streaming and on-the-fly stream classification and clustering; (5) optimization and improvement of computational performance; and (6) parallel computing.

The classical approaches for feature selection are presented in **Chap. 17 (Variable/Feature Selection)** where we discuss (1) filtering, wrapper, and embedded techniques, and (2) show the entire protocols from data collection and preparation to model training, testing, evaluation and comparison using recursive feature elimination.

In **Chap. 18 (Regularized Linear Modeling and Controlled Variable Selection)**, we extend the mathematical foundation we presented in Chap. 5 to include *fidelity* and *regularization* terms in the objective function used for model-based inference. Specifically, we discuss (1) computational protocols for handling complex high-dimensional data, (2) model estimation by controlling the false-positive rate of selection of critical features, and (3) derivations of effective forecasting models.

Chapter 19 (BigBig Longitudinal Data Analysis) is focused on interrogating time-varying observations. We illustrate (1) time series analysis, e.g., ARIMA modeling, (2) structural equation modeling (SEM) with latent variables, (3) longitudinal data analysis using linear mixed models, and (4) the generalized estimating equations (GEE) modeling.

Expanding upon the term-frequency and inverse document frequency techniques we saw in Chap. 8, **Chap. 20 (Natural Language Processing/Text Mining)** provides more details about (1) handling unstructured text documents, (2) term frequency (TF) and inverse document frequency (IDF), and (3) the cosine similarity measure.

Chapter 21 (Prediction and Internal Statistical Cross Validation) provides a broader and deeper discussion of method validation, which started in Chap. 14. Here, we present (1) general prediction and forecasting methods, (2) demonstrate internal statistical n-fold cross-validation, and (3) comparison strategies for multiple prediction models.

Chapter 22 (Function Optimization) presents technical details about minimizing objective functions, which are present virtually in any data science oriented inference or evidence-based translational study. Here, we explain (1) constrained and unconstrained cost function optimization, (2) Lagrange multipliers, (3) linear and quadratic programming, (4) general nonlinear optimization, and (5) data denoising.

The last chapter of this textbook is **Chap. 23 (Deep Learning)**. It covers (1) perceptron activation functions, (2) relations between artificial and biological

neurons and networks, (3) neural nets for computing exclusive OR (XOR) and negative AND (NAND) operators, (3) classification of handwritten digits, and (4) classification of natural images.

We compiled a few dozens of biomedical and healthcare case-studies that are used to demonstrate the presented DSPA concepts, apply the methods, and validate the software tools. For example, **Chap. 1** includes high-level driving biomedical challenges including dementia and other neurodegenerative diseases, substance use, neuroimaging, and forensic genetics. **Chapter 3** includes a traumatic brain injury (TBI) case-study, **Chap. 10** described a heart attacks case-study, and **Chap. 11** uses a quality of life in chronic disease data to demonstrate optical character recognition that can be applied to automatic reading of handwritten physician notes. **Chapter 18** presents a predictive analytics Parkinson's disease study using neuroimaging-genetics data. **Chapter 20** illustrates the applications of natural language processing to extract quantitative biomarkers from unstructured text, which can be used to study hospital admissions, medical claims, or patient satisfaction. **Chapter 23** shows examples of predicting clinical outcomes for amyotrophic lateral sclerosis and irritable bowel syndrome cohorts, as well as quantitative and qualitative classification of biological images and volumes. Indeed, these represent just a few examples, and the readers are encouraged to try the same methods, protocols and analytics on other research-derived, clinically acquired, aggregated, secondary-use, or simulated datasets.

The online appendices (<http://DSPA.predictive.space>) are continuously expanded to provide more details, additional content, and expand the DSPA methods and applications scope. Throughout this textbook, there are cross-references to appropriate chapters, sections, datasets, web services, and live demonstrations (Live Demos). The sequential arrangement of the chapters provides a suggested reading order; however, alternative sorting and pathways covering parts of the materials are also provided. Of course, readers and instructors may further choose their own coverage paths based on specific intellectual interests and project needs.

Preface

Genesis

Since the turn of the twenty-first century, the evidence overwhelming reveals that the rate of increase for the amount of data we collect doubles each 12–14 months (Kryder’s law). The growth momentum of the volume and complexity of digital information we gather far outpaces the corresponding increase of computational power, which doubles each 18 months (Moore’s law). There is a substantial imbalance between the increase of data inflow and the corresponding computational infrastructure intended to process that data. This calls into question our ability to extract valuable information and actionable knowledge from the mountains of digital information we collect. Nowadays, it is very common for researchers to work with petabytes (PB) of data, $1PB = 10^{15}$ bytes, which may include nonhomologous records that demand unconventional analytics. For comparison, the Milky Way Galaxy has approximately 2×10^{11} stars. If each star represents a byte, then one petabyte of data correspond to 5,000 Milky Way Galaxies.

This data storage-computing asymmetry leads to an explosion of innovative data science methods and disruptive computational technologies that show promise to provide effective (semi-intelligent) decision support systems. Designing, understanding and validating such new techniques require deep within-discipline basic science knowledge, transdisciplinary team-based scientific collaboration, open-scientific endeavors, and a blend of exploratory and confirmatory scientific discovery. There is a pressing demand to bridge the widening gaps between the needs and skills of practicing data scientists, advanced techniques introduced by theoreticians, algorithms invented by computational scientists, models constructed by biosocial investigators, network products and Internet of Things (IoT) services engineered by software architects.

Purpose

The purpose of this book is to provide a sufficient methodological foundation for a number of modern data science techniques along with hands-on demonstration of implementation protocols, pragmatic mechanics of protocol execution, and interpretation of the results of these methods applied on concrete case-studies. Successfully completing the Data Science and Predictive Analytics (DSPA) training materials (<http://predictive.space>) will equip readers to (1) understand the computational foundations of Big Data Science; (2) build critical inferential thinking; (3) lend a tool chest of *R* libraries for managing and interrogating raw, derived, observed, experimental, and simulated big healthcare datasets; and (4) furnish practical skills for handling complex datasets.

Limitations/Prerequisites

Prior to diving into DSPA, the readers are strongly encouraged to review the prerequisites and complete the self-assessment pretest. Sufficient remediation materials are provided or referenced throughout. The DSPA materials may be used for variety of graduate level courses with durations of 10–30 weeks, with 3–4 instructional credit hours per week. Instructors can refactor and present the materials in alternative orders. The DSPA chapters in this book are organized sequentially. However, the content can be tailored to fit the audience’s needs. Learning data science and predictive analytics is not a linear process – many alternative pathways

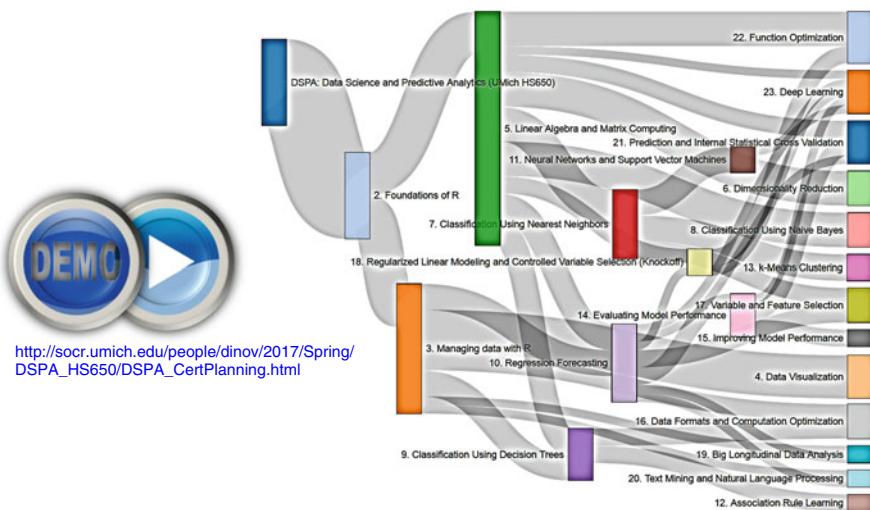


Fig. 1 DSPA topics flowchart

can be completed to gain complementary competencies. We developed an interactive and dynamic flowchart (http://socr.umich.edu/people/dinov/courses/DSPA_Book_FlowChart.html) that highlights several tracks illustrating reasonable pathways starting with Foundations of *R* and ending with specific competency topics. The content of this book may also be used for self-paced learning or as a refresher for working professionals, as well as for formal and informal data science training, including massive open online courses (MOOCs). The DSPA materials are designed to build specific data science skills and predictive analytic competencies, as described by the Michigan Institute for Data Science (MIDAS).

Scope of the Book

Throughout this book, we use a constructive definition of “Big Data” derived by examining the common characteristics of many dozens of biomedical and healthcare case-studies, involving complex datasets that required special handling, advanced processing, contemporary analytics, interactive visualization tools, and translational interpretation. These six characteristics of “Big Data” are defined in the Motivation Chapter as size, heterogeneity and complexity, representation incongruency, incompleteness, multiscale format, and multisource origins. All supporting electronic materials, including datasets, assessment problems, code, software tools, videos, and appendices, are available online at <http://DSPA.predictive.space>.

This textbook presents a balanced view of the mathematical formulation, computational implementation, and health applications of modern techniques for managing, processing, and interrogating big data. The intentional focus on human health applications is demonstrated by a diverse range of biomedical and healthcare case-studies. However, the same techniques could be applied in other domains, e.g., climate and environmental sciences, biosocial sciences, high-energy physics, astronomy, etc., that deal with complex data possessing the above characteristics. Another specific feature of this book is that it solely utilizes the statistical computing language *R*, rather than any other scripting, user-interface based, or software programming alternatives. The choice for *R* is justified in the Foundations Chapter.

All techniques presented here aim to obtain data-driven and evidence-based scientific inference. This process starts with collecting or retrieving an appropriate dataset, and identifying sources of data that need to be harmonized and aggregated into a joint computable data object. Next, the data are typically split into training and testing components. Model-based or model-free methods are fit, estimated, or learned on the training component and then validated on the complementary testing data. Different types of expected outcomes and results from this process include prediction, prognostication, or forecasting of specific clinical traits (computable phenotypes), clustering, or classification that labels units, subjects, or cases in the data. The final steps include algorithm fine-tuning, assessment, comparison, and statistical validation.

Acknowledgements

The work presented in this textbook relies on deep basic science, as well as holistic interdisciplinary connections developed by scholars, teams of scientists, and transdisciplinary collaborations. Ideas, datasets, software, algorithms, and methods introduced by the wider scientific community were utilized throughout the DSPA resources. Specifically, methodological and algorithmic contributions from the fields of computer vision, statistical learning, mathematical optimization, scientific inference, biomedical computing, and informatics drove the concept presentations, data-driven demonstrations, and case-study reports. The enormous contributions from the entire *R* statistical computing community were critical for developing these resources. We encourage community contributions to expand the techniques, bolster their scope and applications, enhance the collection of case-studies, optimize the algorithms, and widen the applications to other data-intense disciplines or complex scientific challenges.

The author is profoundly indebted to all of his direct mentors and advisors for nurturing my curiosity, inspiring my studies, guiding the course of my career, and providing constructive and critical feedback throughout. Among these scholars are Gencho Skordev (Sofia University); Kenneth Kuttler (Michigan Tech University); De Witt L. Sumners and Fred Huffer (Florida State University); Jan de Leeuw, Nicolas Christou, and Michael Mega (UCLA); Arthur Toga (USC); and Brian Athey, Patricia Hurn, Kathleen Potempa, Janet Larson, and Gilbert Omenn (University of Michigan).

Many other colleagues, students, researchers, and fellows have shared their expertise, creativity, valuable time, and critical assessment for generating, validating, and enhancing these open-science resources. Among these are Christopher Aakre, Simeone Marino, Jiachen Xu, Ming Tang, Nina Zhou, Chao Gao, Alexander Kalinin, Syed Husain, Brady Zhu, Farshid Sepehrband, Lu Zhao, Sam Hobel, Hanbo Sun, Tuo Wang, and many others. Many colleagues from the Statistics Online Computational Resource (SOCR), the Big Data for Discovery Science (BDDS) Center, and the Michigan Institute for Data Science (MIDAS) provided encouragement and valuable suggestions.

The development of the DSPA materials was partially supported by the US National Science Foundation (grants 1734853, 1636840, 1416953, 0716055, and 1023115), US National Institutes of Health (grants P20 NR015331, U54 EB020406, P50 NS091856, P30 DK089503, P30AG053760), and the Elsie Andressen Fiske Research Fund.

Ann Arbor, MI, USA

Ivo D. Dinov

DSPA Application and Use Disclaimer

The Data Science and Predictive Analytics (DSPA) resources are designed to help scientists, trainees, students, and professionals learn the foundation of data science, practical applications, and pragmatics of dealing with concrete datasets, and to experiment in a sandbox of specific case-studies. Neither the author nor the publisher have control over, or make any representation or warranties, expressed or implied, regarding the use of these resources by researchers, users, patients, or their healthcare provider(s), or the use or interpretation of any information stored on, derived, computed, suggested by, or received through any of the DSPA materials, code, scripts, or applications. All users are solely responsible for deriving, interpreting, and communicating any information to (and receiving feedback from) the user's representatives or healthcare provider(s).

Users, their proxies, or representatives (e.g., clinicians) are solely responsible for reviewing and evaluating the accuracy, relevance, and meaning of any information stored on, derived by, generated by, or received through the application of any of the DSPA software, protocols, or techniques. The author and the publisher cannot and do not guarantee said accuracy. *The DSPA resources, their applications, and any information stored on, generated by, or received through them are not intended to be a substitute for professional or expert advice, diagnosis, or treatment. Always seek the advice of a physician or other qualified professional with any questions regarding any real case-study (e.g., medical diagnosis, conditions, prediction, and prognostication). Never disregard professional advice or delay seeking it because of something read or learned through the use of the DSPA material or any information stored on, generated by, or received through the SOCR resources.*

All readers and users acknowledge that the DSPA copyright owners or licensors, in their sole discretion, may from time to time make modifications to the DSPA resources. Such modifications may require corresponding changes to be made in the code, protocols, learning modules, activities, case-studies, and other DSPA materials. Neither the author, publisher, nor licensors shall have any obligation to furnish any maintenance or support services with respect to the DSPA resources.

The DSPA resources are intended for educational purposes only. They are not intended to offer or replace any professional advice nor provide expert opinion. Please speak to qualified professional service providers if you have any specific concerns, case-studies, or questions.

Biomedical, Biosocial, Environmental, and Health Disclaimer

All DSPA information, materials, software, and examples are provided for general education purposes only. Persons using the DSPA data, models, tools, or services for any medical, social, healthcare, or environmental purposes should not rely on accuracy, precision, or significance of the DSPA reported results. While the DSPA resources may be updated periodically, users should independently check against other sources, latest advances, and most accurate peer-reviewed information.

Please consult appropriate professional providers prior to making any lifestyle changes or any actions that may impact those around you, your community, or various real, social, and virtual environments. Qualified and appropriate professionals represent the single best source of information regarding any Biomedical, Biosocial, Environmental, and Health decisions. None of these resources have either explicit or implicit indication of FDA approval!

Any and all liability arising directly or indirectly from the use of the DSPA resources is hereby disclaimed. The DSPA resources are provided “as is” and without any warranty expressed or implied. All direct, indirect, special, incidental, consequential, or punitive damages arising from any use of the DSPA resources or materials contained herein are disclaimed and excluded.

Notations

The following common notations are used throughout this textbook.

Notation	Description
 http://www.socr.umich.edu/people/dinov/courses/DSPA_Topoics.html	A link to an interactive live web demonstration. Some of these Live Demos require modern Java and JavaScript enabled browsers and Internet access.
<pre>require(ggplot2) # Comments Loading required package: ggplot2 Data_R_SAS_SPSS_Pubs <- read.csv('https://umich.edu/data', header=T) df <- data.frame(Data_R_SAS_SPSS_Pubs) # convert to long format df <- melt(df, id.vars = 'Year', variable.name = 'Software') ggplot(data=df, aes(x=Year, y=value, color=Software, group = Software)) + geom_line() + geom ## 3 1 a ... ## 20 3 c data_Long ## CaseID Gender Feature Measurement ## 1 1 M Age 5.0 ## 2 2 F Age 6.0</pre>	<p><i>R</i> fragments of code, reported results in the output shell, or comments. The complete library of all code presented in the textbook is available in electronic format on the DSPA site.</p> <p>Note that:</p> <ul style="list-style-type: none">"#" is used for comments,"##" indicates <i>R</i> textual output,the <i>R</i> code is color-coded to identify different types of comments, instructions, commands and parameters,Output like "## ... ##" suggests that some of the <i>R</i> output is deleted or compressed to save space, andindenting is used to visually determine the scope of a method, command, or an expression
<code>← or →</code>	In an asymptotic or limiting sense, tending to, convergence, or approaching a value or a limit.
<code><< or >></code>	Left hand side is substantially smaller or larger than the right hand side.
<code>~</code>	Depending on the context, model definition, similar to, approximately equal to, or equivalent (in probability distribution sense).
<code>package::function</code>	A standard reference notation to functions members of specific <i>R</i> packages.
<code>Case-studies</code>	https://umich.instructure.com/courses/38100/files/folder/Case_Studies
<code>Electronic Materials</code>	http://DSPA.predictive.space
Also see the Glossary and the Index , located in the end of the book.	

Fig. 2 Common DSPA notations

Contents

1	Motivation	1
1.1	DSPA Mission and Objectives	1
1.2	Examples of Driving Motivational Problems and Challenges	2
1.2.1	Alzheimer's Disease	2
1.2.2	Parkinson's Disease	2
1.2.3	Drug and Substance Use	3
1.2.4	Amyotrophic Lateral Sclerosis	4
1.2.5	Normal Brain Visualization	4
1.2.6	Neurodegeneration	4
1.2.7	Genetic Forensics: 2013–2016 Ebola Outbreak	5
1.2.8	Next Generation Sequence (NGS) Analysis	6
1.2.9	Neuroimaging-Genetics	7
1.3	Common Characteristics of Big (Biomedical and Health) Data	8
1.4	Data Science	9
1.5	Predictive Analytics	9
1.6	High-Throughput Big Data Analytics	10
1.7	Examples of Data Repositories, Archives, and Services	10
1.8	DSPA Expectations	11
2	Foundations of R	13
2.1	Why Use R?	13
2.2	Getting Started	15
2.2.1	Install Basic Shell-Based R	15
2.2.2	GUI Based R Invocation (RStudio)	15
2.2.3	RStudio GUI Layout	15
2.2.4	Some Notes	16
2.3	Help	16
2.4	Simple Wide-to-Long Data format Translation	17
2.5	Data Generation	18
2.6	Input/Output (I/O)	22

2.7	Slicing and Extracting Data	24
2.8	Variable Conversion	25
2.9	Variable Information	25
2.10	Data Selection and Manipulation	27
2.11	Math Functions	30
2.12	Matrix Operations	32
2.13	Advanced Data Processing	32
2.14	Strings	37
2.15	Plotting	39
2.16	QQ Normal Probability Plot	41
2.17	Low-Level Plotting Commands	45
2.18	Graphics Parameters	45
2.19	Optimization and model Fitting	47
2.20	Statistics	48
2.21	Distributions	49
2.21.1	Programming	49
2.22	Data Simulation Primer	50
2.23	Appendix	56
2.23.1	HTML SOCR Data Import	56
2.23.2	R Debugging	57
2.24	Assignments: 2. R Foundations	60
2.24.1	Confirm that You Have Installed R/RStudio	60
2.24.2	Long-to-Wide Data Format Translation	61
2.24.3	Data Frames	61
2.24.4	Data Stratification	61
2.24.5	Simulation	61
2.24.6	Programming	62
	References	62
3	Managing Data in R	63
3.1	Saving and Loading R Data Structures	63
3.2	Importing and Saving Data from CSV Files	64
3.3	Exploring the Structure of Data	66
3.4	Exploring Numeric Variables	66
3.5	Measuring the Central Tendency: Mean, Median, Mode	67
3.6	Measuring Spread: Quartiles and the Five-Number Summary	68
3.7	Visualizing Numeric Variables: Boxplots	70
3.8	Visualizing Numeric Variables: Histograms	71
3.9	Understanding Numeric Data: Uniform and Normal Distributions	72
3.10	Measuring Spread: Variance and Standard Deviation	73
3.11	Exploring Categorical Variables	76

3.12	Exploring Relationships Between Variables	77
3.13	Missing Data	79
3.13.1	Simulate Some Real Multivariate Data	84
3.13.2	TBI Data Example	98
3.13.3	Imputation via Expectation-Maximization	122
3.14	Parsing Webpages and Visualizing Tabular HTML Data	130
3.15	Cohort-Rebalancing (for Imbalanced Groups)	135
3.16	Appendix	138
3.16.1	Importing Data from SQL Databases	138
3.16.2	R Code Fragments	139
3.17	Assignments: 3. Managing Data in R	140
3.17.1	Import, Plot, Summarize and Save Data	140
3.17.2	Explore some Bivariate Relations in the Data	140
3.17.3	Missing Data	141
3.17.4	Surface Plots	141
3.17.5	Unbalanced Designs	141
3.17.6	Aggregate Analysis	141
	References	141
4	Data Visualization	143
4.1	Common Questions	143
4.2	Classification of Visualization Methods	144
4.3	Composition	144
4.3.1	Histograms and Density Plots	144
4.3.2	Pie Chart	147
4.3.3	Heat Map	149
4.4	Comparison	152
4.4.1	Paired Scatter Plots	152
4.4.2	Jitter Plot	157
4.4.3	Bar Plots	159
4.4.4	Trees and Graphs	164
4.4.5	Correlation Plots	167
4.5	Relationships	171
4.5.1	Line Plots Using <code>ggplot</code>	171
4.5.2	Density Plots	173
4.5.3	Distributions	173
4.5.4	2D Kernel Density and 3D Surface Plots	174
4.5.5	Multiple 2D Image Surface Plots	176
4.5.6	3D and 4D Visualizations	178
4.6	Appendix	183
4.6.1	Hands-on Activity (Health Behavior Risks)	183
4.6.2	Additional <code>ggplot</code> Examples	187

4.7	Assignments 4: Data Visualization	198
4.7.1	Common Plots	198
4.7.2	Trees and Graphs	198
4.7.3	Exploratory Data Analytics (EDA)	199
	References	199
5	Linear Algebra & Matrix Computing	201
5.1	Matrices (Second Order Tensors)	202
5.1.1	Create Matrices	202
5.1.2	Adding Columns and Rows	203
5.2	Matrix Subscripts	204
5.3	Matrix Operations	204
5.3.1	Addition	204
5.3.2	Subtraction	205
5.3.3	Multiplication	205
5.3.4	Element-wise Division	207
5.3.5	Transpose	207
5.3.6	Multiplicative Inverse	207
5.4	Matrix Algebra Notation	209
5.4.1	Linear Models	209
5.4.2	Solving Systems of Equations	210
5.4.3	The Identity Matrix	212
5.5	Scalars, Vectors and Matrices	213
5.5.1	Sample Statistics (Mean, Variance)	215
5.5.2	Least Square Estimation	218
5.6	Eigenvalues and Eigenvectors	219
5.7	Other Important Functions	220
5.8	Matrix Notation (Another View)	220
5.9	Multivariate Linear Regression	224
5.10	Sample Covariance Matrix	227
5.11	Assignments: 5. Linear Algebra & Matrix Computing	229
5.11.1	How Is Matrix Multiplication Defined?	229
5.11.2	Scalar Versus Matrix Multiplication	229
5.11.3	Matrix Equations	229
5.11.4	Least Square Estimation	230
5.11.5	Matrix Manipulation	230
5.11.6	Matrix Transpose	230
5.11.7	Sample Statistics	230
5.11.8	Least Square Estimation	230
5.11.9	Eigenvalues and Eigenvectors	231
	References	231
6	Dimensionality Reduction	233
6.1	Example: Reducing 2D to 1D	233
6.2	Matrix Rotations	237
6.3	Notation	242

6.4	Summary (PCA vs. ICA vs. FA)	242
6.5	Principal Component Analysis (PCA)	243
6.5.1	Principal Components	243
6.6	Independent Component Analysis (ICA)	250
6.7	Factor Analysis (FA)	254
6.8	Singular Value Decomposition (SVD)	256
6.9	SVD Summary	258
6.10	Case Study for Dimension Reduction (Parkinson's Disease)	258
6.11	Assignments: 6. Dimensionality Reduction	265
6.11.1	Parkinson's Disease Example	265
6.11.2	Allometric Relations in Plants Example	266
	References	266
7	Lazy Learning: Classification Using Nearest Neighbors	267
7.1	Motivation	268
7.2	The kNN Algorithm Overview	269
7.2.1	Distance Function and Dummy Coding	269
7.2.2	Ways to Determine k	270
7.2.3	Rescaling of the Features	270
7.2.4	Rescaling Formulas	271
7.3	Case Study	271
7.3.1	Step 1: Collecting Data	271
7.3.2	Step 2: Exploring and Preparing the Data	272
7.3.3	Normalizing Data	273
7.3.4	Data Preparation: Creating Training and Testing Datasets	274
7.3.5	Step 3: Training a Model On the Data	274
7.3.6	Step 4: Evaluating Model Performance	274
7.3.7	Step 5: Improving Model Performance	275
7.3.8	Testing Alternative Values of k	276
7.3.9	Quantitative Assessment (Tables 7.2 and 7.3)	282
7.4	Assignments: 7. Lazy Learning: Classification Using Nearest Neighbors	286
7.4.1	Traumatic Brain Injury (TBI)	286
7.4.2	Parkinson's Disease	286
7.4.3	KNN Classification in a High Dimensional Space	287
7.4.4	KNN Classification in a Lower Dimensional Space	287
	References	287
8	Probabilistic Learning: Classification Using Naive Bayes	289
8.1	Overview of the Naive Bayes Algorithm	289
8.2	Assumptions	290
8.3	Bayes Formula	290
8.4	The Laplace Estimator	292

8.5	Case Study: Head and Neck Cancer Medication	293
8.5.1	Step 1: Collecting Data	293
8.5.2	Step 2: Exploring and Preparing the Data	293
8.5.3	Step 3: Training a Model on the Data	299
8.5.4	Step 4: Evaluating Model Performance	300
8.5.5	Step 5: Improving Model Performance	301
8.5.6	Step 6: Compare Naive Bayesian against LDA	302
8.6	Practice Problem	303
8.7	Assignments 8: Probabilistic Learning: Classification Using Naive Bayes	304
8.7.1	Explain These Two Concepts	304
8.7.2	Analyzing Textual Data	305
	References	305
9	Decision Tree Divide and Conquer Classification	307
9.1	Motivation	307
9.2	Hands-on Example: Iris Data	308
9.3	Decision Tree Overview	310
9.3.1	Divide and Conquer	311
9.3.2	Entropy	312
9.3.3	Misclassification Error and Gini Index	313
9.3.4	C5.0 Decision Tree Algorithm	313
9.3.5	Pruning the Decision Tree	315
9.4	Case Study 1: Quality of Life and Chronic Disease	316
9.4.1	Step 1: Collecting Data	316
9.4.2	Step 2: Exploring and Preparing the Data	316
9.4.3	Step 3: Training a Model On the Data	319
9.4.4	Step 4: Evaluating Model Performance	322
9.4.5	Step 5: Trial Option	323
9.4.6	Loading the Misclassification Error Matrix	324
9.4.7	Parameter Tuning	325
9.5	Compare Different Impurity Indices	331
9.6	Classification Rules	331
9.6.1	Separate and Conquer	331
9.6.2	The One Rule Algorithm	332
9.6.3	The RIPPER Algorithm	332
9.7	Case Study 2: QoL in Chronic Disease (Take 2)	332
9.7.1	Step 3: Training a Model on the Data	332
9.7.2	Step 4: Evaluating Model Performance	333
9.7.3	Step 5: Alternative Model1	334
9.7.4	Step 5: Alternative Model2	334
9.8	Practice Problem	337

9.9	Assignments 9: Decision Tree Divide and Conquer	342
	Classification	342
9.9.1	Explain These Concepts	342
9.9.2	Decision Tree Partitioning	342
	References	343
10	Forecasting Numeric Data Using Regression Models	345
10.1	Understanding Regression	345
10.1.1	Simple Linear Regression	345
10.2	Ordinary Least Squares Estimation	347
10.2.1	Model Assumptions	349
10.2.2	Correlations	349
10.2.3	Multiple Linear Regression	350
10.3	Case Study 1: Baseball Players	352
10.3.1	Step 1: Collecting Data	352
10.3.2	Step 2: Exploring and Preparing the Data	352
10.3.3	Exploring Relationships Among Features: The Correlation Matrix	356
10.3.4	Visualizing Relationships Among Features: The Scatterplot Matrix	356
10.3.5	Step 3: Training a Model on the Data	358
10.3.6	Step 4: Evaluating Model Performance	359
10.4	Step 5: Improving Model Performance	361
10.4.1	Model Specification: Adding Non-linear Relationships	369
10.4.2	Transformation: Converting a Numeric Variable to a Binary Indicator	370
10.4.3	Model Specification: Adding Interaction Effects	371
10.5	Understanding Regression Trees and Model Trees	373
10.5.1	Adding Regression to Trees	373
10.6	Case Study 2: Baseball Players (Take 2)	374
10.6.1	Step 2: Exploring and Preparing the Data	374
10.6.2	Step 3: Training a Model On the Data	375
10.6.3	Visualizing Decision Trees	375
10.6.4	Step 4: Evaluating Model Performance	377
10.6.5	Measuring Performance with Mean Absolute Error	378
10.6.6	Step 5: Improving Model Performance	378
10.7	Practice Problem: Heart Attack Data	380
10.8	Assignments: 10. Forecasting Numeric Data Using Regression Models	381
	References	381

11	Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines	383
11.1	Understanding Neural Networks	383
11.1.1	From Biological to Artificial Neurons	383
11.1.2	Activation Functions	384
11.1.3	Network Topology	386
11.1.4	The Direction of Information Travel	386
11.1.5	The Number of Nodes in Each Layer	386
11.1.6	Training Neural Networks with Backpropagation	387
11.2	Case Study 1: Google Trends and the Stock Market: Regression	388
11.2.1	Step 1: Collecting Data	388
11.2.2	Step 2: Exploring and Preparing the Data	389
11.2.3	Step 3: Training a Model on the Data	391
11.2.4	Step 4: Evaluating Model Performance	392
11.2.5	Step 5: Improving Model Performance	393
11.2.6	Step 6: Adding Additional Layers	394
11.3	Simple NN Demo: Learning to Compute $\sqrt{\cdot}$	394
11.4	Case Study 2: Google Trends and the Stock Market – Classification	396
11.5	Support Vector Machines (SVM)	398
11.5.1	Classification with Hyperplanes	399
11.6	Case Study 3: Optical Character Recognition (OCR)	403
11.6.1	Step 1: Prepare and Explore the Data	404
11.6.2	Step 2: Training an SVM Model	405
11.6.3	Step 3: Evaluating Model Performance	406
11.6.4	Step 4: Improving Model Performance	408
11.7	Case Study 4: Iris Flowers	409
11.7.1	Step 1: Collecting Data	409
11.7.2	Step 2: Exploring and Preparing the Data	409
11.7.3	Step 3: Training a Model on the Data	411
11.7.4	Step 4: Evaluating Model Performance	412
11.7.5	Step 5: RBF Kernel Function	413
11.7.6	Parameter Tuning	413
11.7.7	Improving the Performance of Gaussian Kernels	415
11.8	Practice	416
11.8.1	Problem 1 Google Trends and the Stock Market	416
11.8.2	Problem 2: Quality of Life and Chronic Disease	416
11.9	Appendix	420
11.10	Assignments: 11. Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines	421
11.10.1	Learn and Predict a Power-Function	421
11.10.2	Pediatric Schizophrenia Study	421
	References	422

12	Apriori Association Rules Learning	423
12.1	Association Rules	423
12.2	The Apriori Algorithm for Association Rule Learning	424
12.3	Measuring Rule Importance by Using Support and Confidence	424
12.4	Building a Set of Rules with the Apriori Principle	425
12.5	A Toy Example	426
12.6	Case Study 1: Head and Neck Cancer Medications	427
12.6.1	Step 1: Collecting Data	427
12.6.2	Step 2: Exploring and Preparing the Data	427
12.6.3	Step 3: Training a Model on the Data	432
12.6.4	Step 4: Evaluating Model Performance	433
12.6.5	Step 5: Improving Model Performance	435
12.7	Practice Problems: Groceries	438
12.8	Summary	441
12.9	Assignments: 12. Apriori Association Rules Learning	442
	References	442
13	k-Means Clustering	443
13.1	Clustering as a Machine Learning Task	443
13.2	Silhouette Plots	446
13.3	The k-Means Clustering Algorithm	447
13.3.1	Using Distance to Assign and Update Clusters	447
13.3.2	Choosing the Appropriate Number of Clusters	448
13.4	Case Study 1: Divorce and Consequences on Young Adults	448
13.4.1	Step 1: Collecting Data	448
13.4.2	Step 2: Exploring and Preparing the Data	449
13.4.3	Step 3: Training a Model on the Data	450
13.4.4	Step 4: Evaluating Model Performance	451
13.4.5	Step 5: Usage of Cluster Information	454
13.5	Model Improvement	455
13.5.1	Tuning the Parameter k	457
13.6	Case Study 2: Pediatric Trauma	459
13.6.1	Step 1: Collecting Data	459
13.6.2	Step 2: Exploring and Preparing the Data	460
13.6.3	Step 3: Training a Model on the Data	461
13.6.4	Step 4: Evaluating Model Performance	462
13.6.5	Practice Problem: Youth Development	465
13.7	Hierarchical Clustering	467
13.8	Gaussian Mixture Models	470
13.9	Summary	472
13.10	Assignments: 13. k-Means Clustering	472
	References	473

14	Model Performance Assessment	475
14.1	Measuring the Performance of Classification Methods	475
14.1.1	Evaluation Strategies	477
14.1.1.1	Binary Outcomes	477
14.1.1.2	Confusion Matrices	478
14.1.1.3	Other Measures of Performance Beyond Accuracy	480
14.1.1.4	The Kappa (κ) Statistic	481
14.1.1.5	Computation of Observed Accuracy and Expected Accuracy	484
14.1.1.6	Sensitivity and Specificity	485
14.1.1.7	Precision and Recall	486
14.1.1.8	The F-Measure	487
14.2	Visualizing Performance Tradeoffs (ROC Curve)	488
14.3	Estimating Future Performance (Internal Statistical Validation)	491
14.3.1	The Holdout Method	491
14.3.2	Cross-Validation	492
14.3.3	Bootstrap Sampling	494
14.4	Assignment: 14. Evaluation of Model Performance	495
	References	496
15	Improving Model Performance	497
15.1	Improving Model Performance by Parameter Tuning	497
15.2	Using <i>caret</i> for Automated Parameter Tuning	497
15.2.1	Customizing the Tuning Process	501
15.2.2	Improving Model Performance with Meta-learning	502
15.2.3	Bagging	503
15.2.4	Boosting	505
15.2.5	Random Forests	506
15.2.6	Adaptive Boosting	508
15.3	Assignment: 15. Improving Model Performance	510
15.3.1	Model Improvement Case Study	511
	References	511
16	Specialized Machine Learning Topics	513
16.1	Working with Specialized Data and Databases	513
16.1.1	Data Format Conversion	514
16.1.2	Querying Data in SQL Databases	515
16.1.3	Real Random Number Generation	521
16.1.4	Downloading the Complete Text of Web Pages	522
16.1.5	Reading and Writing XML with the XML Package	523
16.1.6	Web-Page Data Scraping	524
16.1.7	Parsing JSON from Web APIs	525
16.1.8	Reading and Writing Microsoft Excel Spreadsheets Using XLSX	526

16.2	Working with Domain-Specific Data	527
16.2.1	Working with Bioinformatics Data	527
16.2.2	Visualizing Network Data	528
16.3	Data Streaming	533
16.3.1	Definition	533
16.3.2	The <code>stream</code> Package	534
16.3.3	Synthetic Example: Random Gaussian Stream	534
16.3.4	Sources of Data Streams	536
16.3.5	Printing, Plotting and Saving Streams	537
16.3.6	Stream Animation	538
16.3.7	Case-Study: SOCR Knee Pain Data	540
16.3.8	Data Stream Clustering and Classification (DSC)	542
16.3.9	Evaluation of Data Stream Clustering	545
16.4	Optimization and Improving the Computational Performance	546
16.4.1	Generalizing Tabular Data Structures with <code>dplyr</code>	547
16.4.2	Making Data Frames Faster with <code>Data.Table</code>	548
16.4.3	Creating Disk-Based Data Frames with <code>ff</code>	548
16.4.4	Using Massive Matrices with <code>bigmemory</code>	549
16.5	Parallel Computing	549
16.5.1	Measuring Execution Time	550
16.5.2	Parallel Processing with Multiple Cores	550
16.5.3	Parallelization Using <code>foreach</code> and <code>doParallel</code>	552
16.5.4	GPU Computing	553
16.6	Deploying Optimized Learning Algorithms	553
16.6.1	Building Bigger Regression Models with <code>biglm</code>	553
16.6.2	Growing Bigger and Faster Random Forests with <code>bigrf</code>	553
16.6.3	Training and Evaluation Models in Parallel with <code>caret</code>	554
16.7	Practice Problem	554
16.8	Assignment: 16. Specialized Machine Learning Topics	555
16.8.1	Working with Website Data	555
16.8.2	Network Data and Visualization	555
16.8.3	Data Conversion and Parallel Computing	555
	References	556
17	Variable/Feature Selection	557
17.1	Feature Selection Methods	557
17.1.1	Filtering Techniques	557
17.1.2	Wrapper Methods	558
17.1.3	Embedded Techniques	558
17.2	Case Study: ALS	559
17.2.1	Step 1: Collecting Data	559
17.2.2	Step 2: Exploring and Preparing the Data	559

17.2.3	Step 3: Training a Model on the Data	560
17.2.4	Step 4: Evaluating Model Performance	564
17.3	Practice Problem	569
17.4	Assignment: 17. Variable/Feature Selection	571
17.4.1	Wrapper Feature Selection	571
17.4.2	Use the PPMI Dataset	571
	References	572
18	Regularized Linear Modeling and Controlled Variable Selection	573
18.1	Questions	574
18.2	Matrix Notation	574
18.3	Regularized Linear Modeling	574
18.3.1	Ridge Regression	576
18.3.2	Least Absolute Shrinkage and Selection Operator (LASSO) Regression	579
18.3.3	Predictor Standardization	582
18.3.4	Estimation Goals	582
18.4	Linear Regression	582
18.4.1	Drawbacks of Linear Regression	583
18.4.2	Assessing Prediction Accuracy	583
18.4.3	Estimating the Prediction Error	583
18.4.4	Improving the Prediction Accuracy	584
18.4.5	Variable Selection	585
18.5	Regularization Framework	586
18.5.1	Role of the Penalty Term	586
18.5.2	Role of the Regularization Parameter	586
18.5.3	LASSO	587
18.5.4	General Regularization Framework	587
18.6	Implementation of Regularization	588
18.6.1	Example: Neuroimaging-Genetics Study of Parkinson's Disease Dataset	588
18.6.2	Computational Complexity	590
18.6.3	LASSO and Ridge Solution Paths	590
18.6.4	Choice of the Regularization Parameter	598
18.6.5	Cross Validation Motivation	599
18.6.6	n -Fold Cross Validation	599
18.6.7	LASSO 10-Fold Cross Validation	600
18.6.8	Stepwise OLS (Ordinary Least Squares)	601
18.6.9	Final Models	602
18.6.10	Model Performance	604
18.6.11	Comparing Selected Features	604
18.6.12	Summary	605
18.7	Knock-off Filtering: Simulated Example	605
18.7.1	Notes	607

18.8	PD Neuroimaging-Genetics Case-Study	608
18.8.1	Fetching, Cleaning and Preparing the Data	608
18.8.2	Preparing the Response Vector	609
18.8.3	False Discovery Rate (FDR)	617
18.8.4	Running the Knockoff Filter	620
18.9	Assignment: 18. Regularized Linear Modeling and Knockoff Filtering	621
	References	622
19	Big Longitudinal Data Analysis	623
19.1	Time Series Analysis	623
19.1.1	Step 1: Plot Time Series	626
19.1.2	Step 2: Find Proper Parameter Values for ARIMA Model	628
19.1.3	Check the Differencing Parameter	629
19.1.4	Identifying the AR and MA Parameters	630
19.1.5	Step 3: Build an ARIMA Model	632
19.1.6	Step 4: Forecasting with ARIMA Model	637
19.2	Structural Equation Modeling (SEM)-Latent Variables	638
19.2.1	Foundations of SEM	638
19.2.2	SEM Components	641
19.2.3	Case Study – Parkinson’s Disease (PD)	642
19.2.4	Outputs of Lavaan SEM	647
19.3	Longitudinal Data Analysis-Linear Mixed Models	648
19.3.1	Mean Trend	648
19.3.2	Modeling the Correlation	652
19.4	GLMM/GEE Longitudinal Data Analysis	653
19.4.1	GEE Versus GLMM	655
19.5	Assignment: 19. Big Longitudinal Data Analysis	657
19.5.1	Imaging Data	657
19.5.2	Time Series Analysis	658
19.5.3	Latent Variables Model	658
	References	658
20	Natural Language Processing/Text Mining	659
20.1	A Simple NLP/TM Example	660
20.1.1	Define and Load the Unstructured-Text Documents	661
20.1.2	Create a New VCorpus Object	663
20.1.3	To-Lower Case Transformation	664
20.1.4	Text Pre-processing	664
20.1.5	Bags of Words	666
20.1.6	Document Term Matrix	667

20.2	Case-Study: Job Ranking	669
20.2.1	Step 1: Make a VCorpus Object	670
20.2.2	Step 2: Clean the VCorpus Object	670
20.2.3	Step 3: Build the Document Term Matrix	670
20.2.4	Area Under the ROC Curve	674
20.3	TF-IDF	676
20.3.1	Term Frequency (TF)	676
20.3.2	Inverse Document Frequency (IDF)	676
20.3.3	TF-IDF	677
20.4	Cosine Similarity	685
20.5	Sentiment Analysis	686
20.5.1	Data Preprocessing	686
20.5.2	NLP/TM Analytics	689
20.5.3	Prediction Optimization	692
20.6	Assignment: 20. Natural Language Processing/Text Mining	694
20.6.1	Mining Twitter Data	694
20.6.2	Mining Cancer Clinical Notes	695
	References	695
21	Prediction and Internal Statistical Cross Validation	697
21.1	Forecasting Types and Assessment Approaches	697
21.2	Overfitting	698
21.2.1	Example (US Presidential Elections)	698
21.2.2	Example (Google Flu Trends)	698
21.2.3	Example (Autism)	700
21.3	Internal Statistical Cross-Validation is an Iterative Process	701
21.4	Example (Linear Regression)	702
21.4.1	Cross-Validation Methods	703
21.4.2	Exhaustive Cross-Validation	703
21.4.3	Non-Exhaustive Cross-Validation	704
21.5	Case-Studies	704
21.5.1	Example 1: Prediction of Parkinson's Disease Using Adaptive Boosting (AdaBoost)	705
21.5.2	Example 2: Sleep Dataset	708
21.5.3	Example 3: Model-Based (Linear Regression) Prediction Using the Attitude Dataset	710
21.5.4	Example 4: Parkinson's Data (ppmi_data)	711
21.6	Summary of CV output	712
21.7	Alternative Predictor Functions	712
21.7.1	Logistic Regression	713
21.7.2	Quadratic Discriminant Analysis (QDA)	714
21.7.3	Foundation of LDA and QDA for Prediction, Dimensionality Reduction, and Forecasting	715
21.7.4	Neural Networks	717
	SVM	718

21.7.6	k-Nearest Neighbors Algorithm (k-NN)	719
21.7.7	k-Means Clustering (k-MC)	720
21.7.8	Spectral Clustering	727
21.8	Compare the Results	730
21.9	Assignment: 21. Prediction and Internal Statistical Cross-Validation	733
	References	734
22	Function Optimization	735
22.1	Free (Unconstrained) Optimization	735
22.1.1	Example 1: Minimizing a Univariate Function (Inverse-CDF)	736
22.1.2	Example 2: Minimizing a Bivariate Function	738
22.1.3	Example 3: Using Simulated Annealing to Find the Maximum of an Oscillatory Function	739
22.2	Constrained Optimization	740
22.2.1	Equality Constraints	740
22.2.2	Lagrange Multipliers	740
22.2.3	Inequality Constrained Optimization	741
	Quadratic Programming (QP)	747
22.3	General Non-linear Optimization	748
22.3.1	Dual Problem Optimization	749
22.4	Manual Versus Automated Lagrange Multiplier Optimization	753
22.5	Data Denoising	756
22.6	Assignment: 22. Function Optimization	761
22.6.1	Unconstrained Optimization	761
22.6.2	Linear Programming (LP)	761
22.6.3	Mixed Integer Linear Programming (MILP)	762
22.6.4	Quadratic Programming (QP)	762
22.6.5	Complex Non-linear Optimization	762
22.6.6	Data Denoising	763
	References	763
23	Deep Learning, Neural Networks	765
23.1	Deep Learning Training	766
23.1.1	Perceptrons	766
23.2	Biological Relevance	768
23.3	Simple Neural Net Examples	770
23.3.1	Exclusive OR (XOR) Operator	770
23.3.2	NAND Operator	771
23.3.3	Complex Networks Designed Using Simple Building Blocks	772
23.4	Classification	773
23.4.1	Sonar Data Example	774
23.4.2	MXNet Notes	781

23.5	Case-Studies	782
23.5.1	ALS Regression Example	783
23.5.2	Spirals 2D Data	785
23.5.3	IBS Study	789
23.5.4	Country QoL Ranking Data	792
23.5.5	Handwritten Digits Classification	795
23.6	Classifying Real-World Images	806
23.6.1	Load the Pre-trained Model	806
23.6.2	Load, Preprocess and Classify New Images – US Weather Pattern	806
23.6.3	Lake Mapourika, New Zealand	810
23.6.4	Beach Image	811
23.6.5	Volcano	812
23.6.6	Brain Surface	814
23.6.7	Face Mask	815
23.7	Assignment: 23. Deep Learning, Neural Networks	816
23.7.1	Deep Learning Classification	816
23.7.2	Deep Learning Regression	817
23.7.3	Image Classification	817
	References	817
	Summary	819
	Glossary	823
	Index	825

Chapter 1

Motivation



1.1 DSPA Mission and Objectives

This textbook is based on the Data Science and Predictive Analytics (DSPA) course taught by the author at the University of Michigan. These materials collectively aim to provide learners with a solid foundation of the challenges, opportunities, and strategies for designing, collecting, managing, processing, interrogating, analyzing, and interpreting complex health and biomedical datasets. Readers that finish this textbook and successfully complete the examples and assignments will gain unique skills and acquire a tool-chest of methods, software tools, and protocols that can be applied to a broad spectrum of Big Data problems.

The DSPA textbook vision, values, and priorities are summarized below:

- **Vision:** Enable active learning by integrating driving motivational challenges with mathematical foundations, computational statistics, and modern scientific inference.
- **Values:** Effective, reliable, reproducible, and transformative data-driven discovery supporting open science.
- **Strategic priorities:** Trainees will develop scientific intuition, computational skills, and data-wrangling abilities to tackle big biomedical and health data problems. Instructors will provide well-documented *R*-scripts and software recipes implementing atomic data filters as well as complex end-to-end predictive big data analytics solutions.

Before diving into the mathematical algorithms, statistical computing methods, software tools, and health analytics covered in the remaining chapters, we will discuss several *driving motivational problems*. These will ground all the subsequent scientific discussions, data modeling techniques, and computational approaches.

1.2 Examples of Driving Motivational Problems and Challenges

For each of the studies below, we illustrate several clinically relevant scientific questions, identify appropriate data sources, describe the types of data elements, and pinpoint various complexity challenges.

1.2.1 *Alzheimer's Disease*

- Identify the relation between observed clinical phenotypes and expected behavior.
- Prognosticate future cognitive decline (3–12 months, prospectively) as a function of imaging data and clinical assessment (both model-based and model-free machine learning prediction methods will be used).
- Derive and interpret the classifications of subjects into clusters using the harmonized and aggregated data from multiple sources (Fig. 1.1).

1.2.2 *Parkinson's Disease*

- Predict the clinical diagnosis of patients using all available data (with and without the unified Parkinson's disease rating scale (UPDRS) clinical assessment, which is the basis of the clinical diagnosis by a physician).
- Compute derived neuroimaging and genetics biomarkers that can be used to model the disease progression and provide automated clinical decisions support.
- Generate decision trees for numeric and categorical responses (representing clinically relevant outcome variables) that can be used to suggest an appropriate course of treatment for specific clinical phenotypes (Fig. 1.2).

Data Source	Sample Size/Data Type	Summary
ADNI Archive	Clinical data: demographics, clinical assessments, cognitive assessments; Imaging data: sMRI, fMRI, DTI, PiB/FDG PET; Genetics data: Illumina SNP genotyping; Chemical biomarker: lab tests, proteomics. Each data modality comes with a different number of cohorts. Generally, $200 \leq N \leq 1200$. For instance, previously conducted ADNI studies with $N > 500$ [doi: 10.3233/JAD-150335 , doi: 10.1111/ion.12252 , doi: 10.3389/fninf.2014.00041].	ADNI provides interesting data modalities, multiple cohorts (e.g., early-onset, mild, and severe dementia, controls) that allow effective model training and validation NACC Archive .

Fig. 1.1 Outline of an Alzheimer's disease case-study

Data Source	Sample Size/Data Type	Summary
PPMI Archive	Demographics: age, medical history, sex; Clinical data: physical, verbal learning and language, neurological and olfactory (University of Pennsylvania Smell Identification Test, UPSIT) tests, vital signs, MDS-UPDRS scores (Movement Disorder; Society-Unified Parkinson's Disease Rating Scale), ADL (activities of daily living), Montreal Cognitive Assessment (MoCA), Geriatric Depression Scale (GDS-15); Imaging data: structural MRI; Genetics data: Illumina ImmunoChip (196,524 variants) and NeuroX (covering 240,000 exonic variants) with 100% sample success rate, and 98.7% genotype success rate genotyped for APOE e2/e3/e4. Three cohorts of subjects; Group 1 = {de novo PD Subjects with a diagnosis of PD for two years or less who are not taking PD medications}, N1 = 263; Group 2 = {PD Subjects with Scans without Evidence of a Dopaminergic Deficit (SWEDD)}, N2 = 40; Group 3 = {Control Subjects without PD who are 30 years or older and who do not have a first degree blood relative with PD}, N3 = 127.	The longitudinal PPMI dataset including clinical, biological, and imaging data (screening, baseline, 12, 24, and 48 month follow-ups) may be used conduct model-based predictions as well as model-free classification and forecasting analyses.

Fig. 1.2 Outline of a Parkinson's disease case-study

Data Source	Sample Size/Data Type	Summary
MAWS Data / UMHS EHR / WHO AWS Data	Scores from Alcohol Use Disorders Identification Test-Consumption (AUDIT-C), including dichotomous variables for any current alcohol use (AUDIT-C, question 1), total AUDIT-C score > 8, and any positive history of alcohol withdrawal syndrome (RAWS).	~1,000 positive cases per year among 10,000 adult medical inpatients, % RAWS screens completed, % positive screens, % entered into MAWS protocol who receive pharmacological treatment for AWS, % entered into MAWS protocol without a completed RAWS screen.

Fig. 1.3 Outline of a substance use case-study

1.2.3 Drug and Substance Use

- Is the Risk for Alcohol Withdrawal Syndrome (RAWS) screen a valid and reliable tool for predicting alcohol withdrawal in an adult medical inpatient population?
- What is the optimal cut-off score from the AUDIT-C to predict alcohol withdrawal based on RAWS screening?
- Should any items be deleted from, or added to, the RAWS screening tool to enhance its performance in predicting the emergence of alcohol withdrawal syndrome in an adult medical inpatient population? (Fig. 1.3)

Data Source	Sample Size/Data Type	Summary
ProAct Archive	Over 100 clinical variables are recorded for all subjects including: Demographics: age, race, medical history, sex; Clinical data: Amyotrophic Lateral Sclerosis Functional Rating Scale (ALSFRS), adverse events, onset_delta, onset_site, drugs use (riluzole). The PRO-ACT training dataset contains clinical and lab test information of 8,635 patients. Information of 2,424 study subjects with valid gold standard ALSFRS slopes will be used in out processing, modeling and analysis.	The time points for all longitudinally varying data elements will be aggregated into signature vectors. This will facilitate the modeling and prediction of ALSFRS slope changes over the first three months (baseline to month 3).

Fig. 1.4 Outline of an amyotrophic lateral sclerosis (Lou Gehrig's disease) case-study

1.2.4 Amyotrophic Lateral Sclerosis

- Identify the most highly significant variables that have power to jointly predict the progression of ALS (in terms of clinical outcomes like ALSFRS and muscle function).
- Provide a decision tree prediction of adverse events based on subject phenotype and 0–3-month clinical assessment changes (Fig. 1.4).

1.2.5 Normal Brain Visualization

The SOCR Brain Visualization tool (<http://socr.umich.edu/HTML5/BrainViewer>) has preloaded sMRI, ROI labels, and fiber track models for a normal brain. It also allows users to drag and drop their data into the browser to visualize and navigate through the stereotactic data (including imaging, parcellations, and tractography) (Fig. 1.5).

1.2.6 Neurodegeneration

A recent study of Structural Neuroimaging in Alzheimer's disease (<https://www.ncbi.nlm.nih.gov/pubmed/26444770>) illustrates the Big Data challenges in modeling complex neuroscientific data. Specifically, 808 ADNI subjects were divided into 3 groups: 200 subjects with Alzheimer's disease (AD), 383 subjects with mild cognitive impairment (MCI), and 225 asymptomatic normal controls (NC). Their sMRI data were parcellated using BrainParser, and the 80 most important neuroimaging biomarkers were extracted using the global shape analysis pipeline workflow. Using a pipeline implementation of Plink, the authors obtained 80 SNPs highly associated with the imaging biomarkers. The authors observed significant

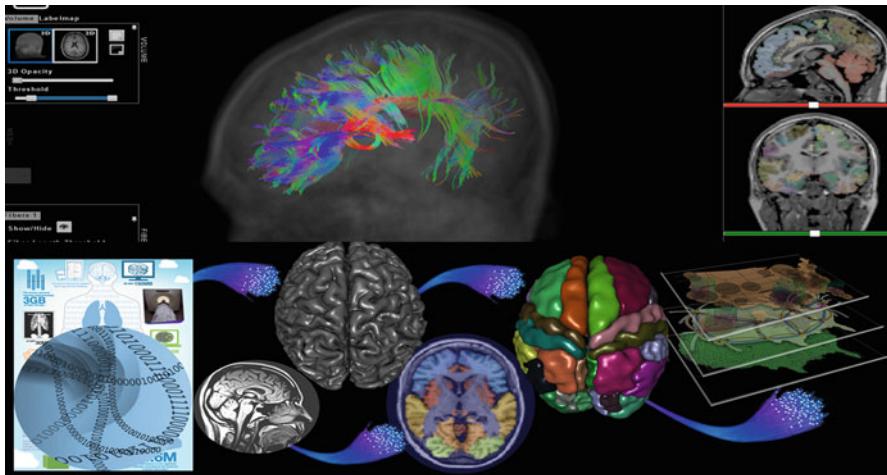
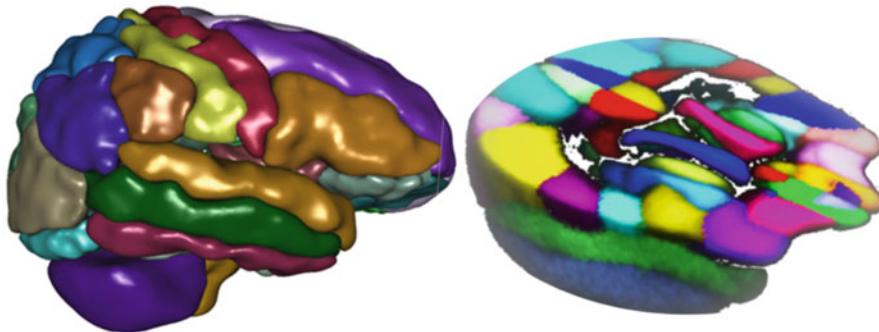


Fig. 1.5 Interactive 3D brain visualization

correlations between genetic and neuroimaging phenotypes in the 808 ADNI subjects. These results suggest that differences between AD, MCI, and NC cohorts may be examined by using powerful joint models of morphometric, imaging, and genotypic data (Fig. 1.6).

1.2.7 *Genetic Forensics: 2013–2016 Ebola Outbreak*

This Howard Hughes Medical Institute (HHMI) disease detective activity illustrates the genetic analysis of sequences of Ebola viruses isolated from patients in Sierra Leone during the Ebola outbreak of 2013–2016. Scientists track the spread of the virus using the fact that most of the genome is identical among individuals of the same species, most similar for genetically related individuals, and more different as the hereditary distance increases. DNA profiling capitalizes on these genetic differences particularly in regions of noncoding DNA, which is DNA that is not transcribed and translated into a protein. Variations in noncoding regions have less impact on individual traits. Such changes in noncoding regions may be immune to natural selection. DNA variations called **short tandem repeats (STRs)** are comprised on short bases, typically 2–5 bases long, that repeat multiple times. The repeat units are found at different locations, or loci, throughout the genome. Every STR has multiple alleles. These allele variants are defined by the **number of repeat units** present or by the **length of the repeat sequence**. STRs are surrounded by nonvariable segments of DNA known as flanking regions. The STR allele in Fig. 1.7 could be denoted by “6”, as the repeat unit (GATA) repeats 6 times, or as 70 base pairs (bps) because its length is 70 bases in length, including the starting/ending flanking regions. Different alleles of the same STR may correspond to different number of GATA repeats, with the same flanking regions.



A: Individual brain parcellation

B: LPBA40 atlas

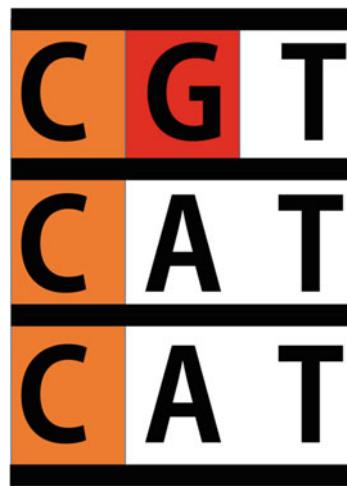
Index	Volume Intensity	ROI Name	Index	Volume Intensity	ROI Name
1	21	L superior frontal gyrus	29	65	L inferior occipital gyrus
2	24	R middle frontal gyrus	30	164	R putamen
3	50	R precuneus	31	61	L superior occipital gyrus
4	181	cerebellum	32	30	R middle orbitofrontal gyrus
5	47	L angular gyrus	33	42	R postcentral gyrus
6	122	R cingulate gyrus	34	27	L precentral gyrus
7	83	L middle temporal gyrus	35	32	R lateral orbitofrontal gyrus
8	90	R lingual gyrus	36	121	L cingulate gyrus
9	81	L superior temporal gyrus	37	31	L lateral orbitofrontal gyrus
10	91	L fusiform gyrus	38	92	R fusiform gyrus
11	44	R superior parietal gyrus	39	45	L supramarginal gyrus
12	66	R inferior occipital gyrus	40	88	R parahippocampal gyrus
13	87	L parahippocampal gyrus	41	22	R superior frontal gyrus
14	162	R caudate	42	29	L middle orbitofrontal gyrus
15	85	L inferior temporal gyrus	43	68	R cuneus
16	182	brainstem	44	62	R superior occipital gyrus
17	43	L superior parietal gyrus	45	33	L gyrus rectus
18	28	R precentral gyrus	46	48	R angular gyrus
19	23	L middle frontal gyrus	47	64	R middle occipital gyrus
20	89	L lingual gyrus	48	84	R middle temporal gyrus
21	41	L postcentral gyrus	49	49	L precuneus
22	86	R inferior temporal gyrus	50	67	L cuneus
23	163	L putamen	51	161	L caudate
24	26	R inferior frontal gyrus	52	165	L hippocampus
25	102	R insular cortex	53	166	R hippocampus
26	25	L inferior frontal gyrus	54	82	R superior temporal gyrus
27	46	R supramarginal gyrus	55	63	L middle occipital gyrus
28	34	R gyrus rectus	56	101	L insular cortex

Fig. 1.6 Indices of the 56 regions of interest (ROIs): A and B – extracted by the BrainParser software using the LPBA40 brain atlas

1.2.8 Next Generation Sequence (NGS) Analysis

Whole-genome and exome sequencing include essential clues for identifying genes responsible for simple Mendelian inherited disorders. A recent paper proposed methods that can be applied to complex disorders based on population genetics.

Fig. 1.7 Snippet of the Ebola STR genomic sequence



Next generation sequencing (NGS) technologies include bioinformatics resources to analyze the dense and complex sequence data. The Graphical Pipeline for Computational Genomics (GPCG) performs the computational steps required to analyze NGS data. The GPCG implements flexible workflows for basic sequence alignment, sequence data quality control, single nucleotide polymorphism analysis, copy number variant identification, annotation, and visualization of results. Applications of NGS analysis provide clinical utility for identifying miRNA signatures in diseases. Enabling hypotheses testing about the functional role of variants in the human genome will help to pinpoint the genetic risk factors many diseases (e.g., neuropsychiatric disorders).

1.2.9 Neuroimaging-Genetics

A computational infrastructure for high-throughput neuroimaging-genetics (doi: <https://doi.org/10.3389/fninf.2014.00041>) facilitates the data aggregation, harmonization, processing, and interpretation of multisource imaging, genomic, clinical, and cognitive data. A unique feature of this architecture is the graphical user interface to the Pipeline environment. Through its client-server architecture, the Pipeline environment provides a graphical user interface for designing, executing, monitoring, validating, and disseminating complex protocols that utilize diverse suites of software tools and web services. These pipeline workflows are represented as portable Extensible Markup Language (XML) objects, which transfer the execution instructions and user specifications from the client user machine to remote pipeline servers for distributed computing. Using Alzheimer's and Parkinson's data, this study provides examples of translational applications using this infrastructure (Figs. 1.8 and 1.9).

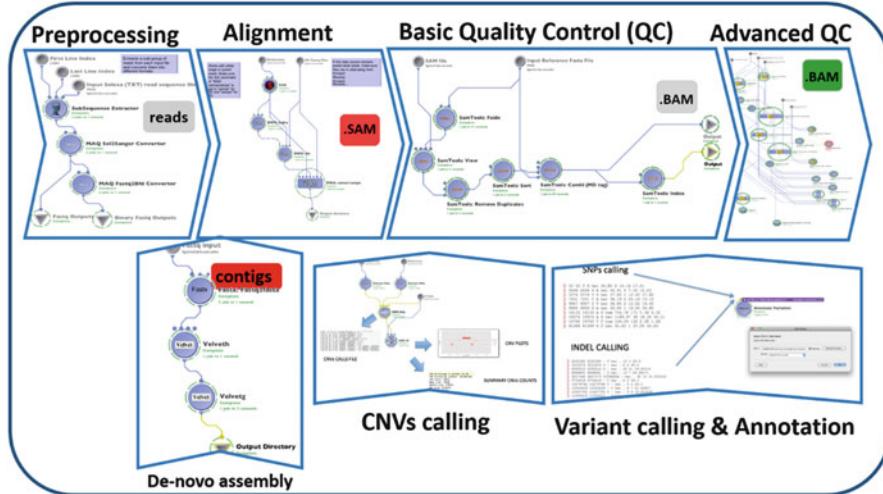


Fig. 1.8 A collage of modules and pipeline workflows from genomic sequence analyses

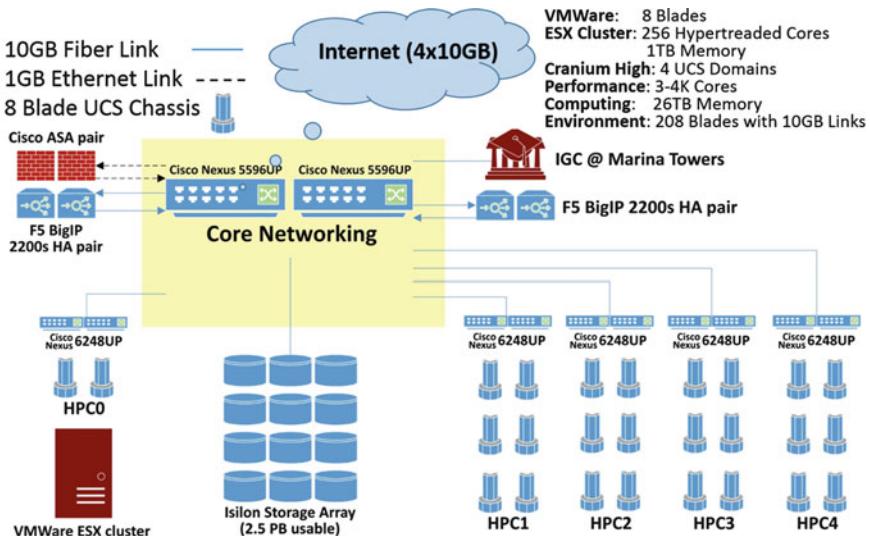


Fig. 1.9 A schematic of a distributed high-throughput computational environment for managing, processing, and visualization of large, complex, and heterogeneous biomedical data

1.3 Common Characteristics of Big (Biomedical and Health) Data

Software developments, student training, utilization of Cloud or IoT (Internet of Things) service platforms, and methodological advances associated with Big Data Discovery Science all present existing opportunities for learners, educators,

Table 1.1 The characteristic six dimensions of Big biomedical and healthcare data

BD dimensions	Necessary techniques, tools, services, and support infrastructure
Size	Harvesting and management of vast amounts of data
Complexity	Wranglers for dealing with heterogeneous data
Incongruency	Tools for data harmonization and aggregation
Multisource	Transfer and joint modeling of disparate elements
Multiscale	Macro to meso- to microscale observations
Incomplete	Reliable management of missing data

researchers, practitioners, and policy makers alike. A review of many biomedical, health informatics, and clinical studies suggests that there are indeed common characteristics of complex big data challenges. For instance, imagine analyzing the observational data of thousands of Parkinson's disease patients, based on tens of thousands of signature biomarkers derived from multisource imaging, genetics, and clinical, physiologic, phenomics, and demographic data elements. IBM had defined the qualitative characteristics of Big Data as 4 Vs: **Volume, Variety, Velocity, and Veracity** (there are additional V-qualifiers that can be added).

More recently (PMID:26998309) we defined a constructive characterization of Big Data that clearly identifies the methodological gaps and necessary tools to handle such archives, Table 1.1.

1.4 Data Science

Data science is an emerging new field that (1) is extremely transdisciplinary – bridging between the theoretical, computational, experimental, and biosocial areas; (2) deals with enormous amounts of complex, incongruent, and dynamic data from multiple sources; and (3) aims to develop algorithms, methods, tools, and services capable of ingesting such datasets and generating semiautomated decision support systems. The latter can mine the data for patterns or motifs, predict expected outcomes, suggest clustering or labeling of retrospective or prospective observations, compute data signatures or fingerprints, extract valuable information, and offer evidence-based actionable knowledge. Data science techniques often involve data manipulation (wrangling), data harmonization and aggregation, exploratory or confirmatory data analyses, predictive analytics, validation, and fine-tuning.

1.5 Predictive Analytics

Predictive analytics is the process of utilizing advanced mathematical formulations, powerful statistical computing algorithms, efficient software tools and services to represent, interrogate, and interpret complex data. As its name suggests, a core aim of predictive analytics is to forecast trends, predict patterns in the data, or

prognosticate the process behavior either within the range or outside the range of the observed data (e.g., in the future, or at locations where data may not be available). In this context, *process* refers to a natural phenomenon that is being investigated by examining proxy data. Presumably, by collecting and exploring the intrinsic data characteristics, we can track the behavior and unravel the underlying mechanism of the system.

The fundamental goal of predictive analytics is to identify relationships, associations, arrangements, or motifs in the dataset, in terms of space, time, and features (variables) that may prune the dimensionality of the data, i.e., reduce its complexity. Using these process characteristics, predictive analytics may predict unknown outcomes, produce estimations of likelihoods or parameters, generate classification labels, or contribute other aggregate or individualized forecasts. We will discuss how the outcomes of these predictive analytics may be refined, assessed, and compared, e.g., between alternative methods. The underlying assumptions of the specific predictive analytics technique determine its usability, affect the expected accuracy, and guide the (human) actions resulting from the (machine) forecasts. In this textbook, we will discuss supervised and unsupervised, model-based and model-free, classification and regression, as well as deterministic, stochastic, classical, and machine learning-based techniques for predictive analytics. The type of the expected outcome (e.g., binary, polytomous, probability, scalar, vector, tensor, etc.) determines if the predictive analytics strategy provides prediction, forecasting, labeling, likelihoods, grouping, or motifs.

1.6 High-Throughput Big Data Analytics

The pipeline environment provides a large tool chest of software and services that can be integrated, merged, and processed. The Pipeline workflow library and the workflow miner illustrate much of the functionality that is available. Java-based and HTML5 webapp graphical user interfaces (GUIs) provide access to a powerful 4,000 core grid compute server (Fig. 1.10).

1.7 Examples of Data Repositories, Archives, and Services

There are many sources of data available on the Internet. A number of them provide open access to the data based on FAIR (Findable, Accessible, Interoperable, Reusable) principles. Below are examples of open-access data sources that can be used to test the techniques presented in this textbook. We demonstrate the tasks of retrieval, manipulation, processing, analytics, and visualization using example datasets from these archives.

- SOCR Wiki Data, http://wiki.socr.umich.edu/index.php/SOCR_Data
- SOCR Canvas datasets, <https://umich.instructure.com/courses/38100/files/folder/data>

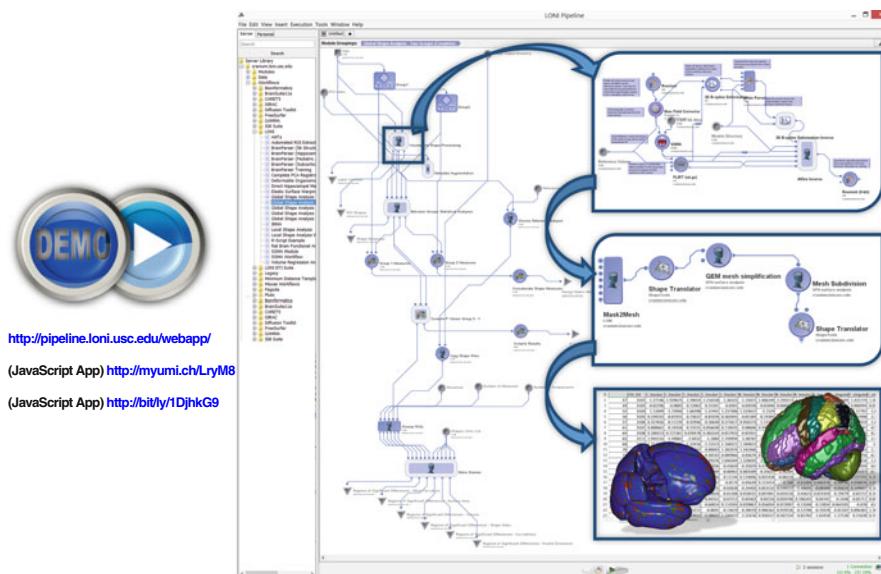


Fig. 1.10 The pipeline environment provides a client-server platform for designing, executing, tracking, sharing, and validating complex data analytic protocols

- SOCR Case-Studies, http://wiki.socr.umich.edu/index.php/SOCR_Data
- XNAT, <https://central.xnat.org>
- IDA, <http://ida.loni.usc.edu>
- NIH dbGaP, <https://dbgap.ncbi.nlm.nih.gov>
- Data.gov (<http://data.gov>)

1.8 DSPA Expectations

The heterogeneity of data science makes it difficult to identify a precise and complete list of prerequisites guaranteeing deep and lasting understanding of all the presented methods and techniques. However, the reader is strongly encouraged to glance over the preliminary prerequisites, the self-assessment pretest and remediation materials, and the outcome competencies. Throughout this journey, it is useful to *remember the following points*:

- You *don't have to* satisfy all prerequisites, be versed in all mathematical foundations, have substantial statistical analysis expertise, or be an experienced programmer.
- You *don't have to complete all chapters and sections* in the order they appear in the DSPA Topics Flowchart. Completing one, or several, of the suggested pathways may be sufficient for many readers.

- The *DSPA textbook* aims to expand the trainees' horizons, improve understanding, enhance skills, and provide a set of advanced, validated, and practice-oriented code, scripts, and protocols.
- To varying degrees, readers will develop abilities to skillfully utilize the **tool chest** of resources provided in the DSPA textbook. These resources can be revised, improved, customized, expanded, and applied to other biomedicine and biosocial studies, as well as to Big Data predictive analytics challenges in other disciplines.
- The DSPA materials will challenge most readers. When *the going gets tough*, seek help, engage with fellow trainees, search for help on the DSPA site and the Internet, communicate via DSPA discussion forum/chat, and review references and supplementary materials. Be proactive! Remember that you will gain, but it will require commitment, prolonged emersion, hard work, and perseverance. If it were easy, its value would be compromised.
- When covering some chapters, some readers may be *underwhelmed or bored*. Feel free to skim over chapters or sections that sound familiar and move forward to the next topic. Still, it is worth trying the corresponding assignments to ensure that you have a firm grasp of the material, and that your technical abilities are sound.
- Although the *return on investment* (e.g., time, effort) may vary between readers, those that complete the DSPA textbook will discover something new, acquire some advanced skills, learn novel data analytic protocols, and may conceive of cutting-edge ideas.
- The complete *R* code (*R* and *Rmd* markdown) for all examples and demonstrations presented in this textbook are available as electronic supplements.
- The author acknowledges that these *materials may be improved*. If you discover problems, typos, errors, inconsistencies, or other problems, please contact us (DSPA.info@umich.edu) to correct, expand, or polish the resources, accordingly. If you have alternative ideas, suggestions for improvements, optimized code, interesting data and case-studies, or any other refinements, please send these along, as well. All suggestions and critiques will be carefully reviewed, and potentially incorporated in revisions or new editions with appropriate credits.

Chapter 2

Foundations of R



This Chapter introduces the foundations of R programming for visualization, statistical computing and scientific inference. Specifically, in this Chapter we will (1) discuss the rationale for selecting R as a computational platform for all DSPA demonstrations; (2) present the basics of installing shell-based R and RStudio user-interface; (3) show some simple R commands and scripts (e.g., translate long-to-wide data format, data simulation, data stratification and subsetting); (4) introduce variable types and their manipulation; (5) demonstrate simple mathematical functions, statistics, and matrix operators; (6) explore simple data visualization; and (7) introduce optimization and model fitting. The chapter appendix includes references to R introductory and advanced resources, as well as a primer on debugging.

2.1 Why Use R?

There are many different classes of software that can be used for data interrogation, modeling, inference, and statistical computing. Among these are R, Python, Java, C/C++, Perl, and many others. The table below compares R to various other statistical analysis software packages and more detailed comparison is available online (Fig. 2.1), https://en.wikipedia.org/wiki/Comparison_of_statistical_packages.

The reader may also review the following two comparisons of various statistical computing software packages:

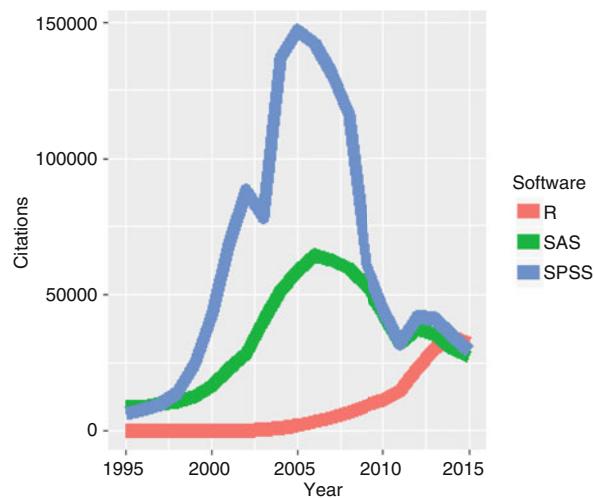
- UCLA Stats Software Comparison
- Wikipedia Stats Software Comparison

Let's start by looking at an exemplary R script that shows the estimates of the citations of three statistical computing software packages over two decades (1995–2015). More details about these command lines will be presented in later chapters. However, it's worth looking at the four specific steps, each indicated by a

Statistical Software	Advantages	Disadvantages
R	R is actively maintained ($\geq 100,000$ developers, $\geq 15K$ packages). Excellent connectivity to various types of data and other systems. Versatile for solving problems in many domains. It's free, open-source code. Anybody can access/review/extend the source code. R is very stable and reliable. If you change or redistribute the R source code, you have to make those changes available for anybody else to use. R runs anywhere (platform agnostic). Extensibility: R supports extensions, e.g., for data manipulation, statistical modeling, and graphics. Active and engaged community supports R. Unparalleled question-and-answer (Q&A) websites. R connects with other languages (Java/C/JavaScript/Python/Fortran) & database systems, and other programs, SAS, SPSS, etc. Other packages have add-ons to connect with R. SPSS has incorporated a link to R, and SAS has protocols to move data and graphics between the two packages.	Mostly scripting language. Steeper learning curve
SAS	Large datasets. Commonly used in business & Government	Expensive. Somewhat dated programming language. Expensive/proprietary
Stata	Easy statistical analyses	Mostly classical stats
SPSS	Appropriate for beginners Simple interfaces	Weak in more cutting edge statistical procedures lacking in robust methods and survey methods

Fig. 2.1 Comparison of several statistical software platforms (R, SAS, Stata, SPSS)

Fig. 2.2 Estimated peer-reviewed publication citations for R, SAS and SPSS softwares



line of code: (1) we start by loading 2 of the necessary R packages for data transformation (`reshape2`) and visualization (`ggplot2`); (2) loading the software citation data from the Internet; (3) reformatting the data; and (4) displaying the composite graph of citations over time (Fig. 2.2).

```
require(ggplot2)
require(reshape2)

Data_R_SAS_SPSS_Pubs <- read.csv('https://umich.instructure.com/files/2361245
/download?download_frd=1', header=T)
df <- data.frame(Data_R_SAS_SPSS_Pubs)
# convert to long format (http://www.cookbook-r.com/Manipulating\_data/Converting\_data\_between\_wide\_and\_long\_format/)
df <- melt(df, id.vars = 'Year', variable.name = 'Software')
ggplot(data=df, aes(x=Year, y=value, color=Software, group = Software)) + 
  geom_line() + geom_line(size=4) + labs(x='Year', y='Citations')
```

2.2 Getting Started

2.2.1 *Install Basic Shell-Based R*

R is a free software that can be installed on any computer. The ‘R’ website is: <http://R-project.org>. There you can download the shell-based R-environment following this protocol:

- click download CRAN in the left bar
- choose a download site
- choose your operation system (e.g., Windows, Mac, Linux)
- click base
- choose the latest version to Download R (3.4, or higher (newer) version for your specific operating system, e.g., Windows).

2.2.2 *GUI Based R Invocation (RStudio)*

For many readers, it’s best to also install and run R via *RStudio GUI* (graphical user interface). To install RStudio, go to: <http://www.rstudio.org/> and do the following:

- click Download RStudio
- click Download RStudio Desktop
- click Recommended For Your System
- download the .exe file and run it (choose default answers for all questions)

2.2.3 *RStudio GUI Layout*

The RStudio interface consists of several windows.

- *Bottom left:* console window (also called command window). Here you can type simple commands after the “>” prompt and R will then execute your command. This is the most important window, because this is where R actually does stuff.

- *Top left*: editor window (also called script window). Collections of commands (scripts) can be edited and saved. When you don't get this window, you can open it with File > New > R script. Just typing a command in the editor window is not enough; it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can click Run or press CTRL + ENTER to send it to the command window.
- *Top right*: workspace / history window. In the workspace window, you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- *Bottom right*: files / plots / packages / help window. Here you can open files, view plots (also previous plots), install and load packages or use the help function. You can change the size of the windows by dragging the grey bars between the windows.

2.2.4 Some Notes

- The basic R environment installation comes with limited core functionality. Everyone eventually will have to install more packages, e.g., `reshape2`, `ggplot2`, and we will show how to expand your RStudio library throughout these materials.
- The core R environment also has to be upgraded occasionally, e.g., every 3–6 months to get R patches, to fix known problems, and to add new functionality. This is also easy to do.
- The assignment operator in R is `<-` (although `=` may also be used), so to assign a value of 2 to a variable `x`, we can write `x <- 2` or equivalently `x = 2`.

2.3 Help

R provides documentations for different R functions. The function call to get these documentations is `help()`. Just put `help(topic)` in the R console and you can get detailed explanations for each R topic or function. Another way of doing it is to call `?topic`, which is even easier, or more generally `??topic`.

For example, if we want to check the function for linear models (i.e. function `lm()`), we can use the following function.

```
help(lm)
?lm
```

2.4 Simple Wide-to-Long Data format Translation

Let's start by experimenting with an R script for transforming (melting) a simple dataset.

```
rawdata_wide <- read.table(header=TRUE, text='
CaseID Gender Age Condition1 Condition2
 1   M    5    13      10.5
 2   F    6    16      11.2
 3   F    8    10      18.3
 4   M    9    9.5     18.1
 5   M   10    12.1    19
')
# Make the CaseID column a factor
rawdata_wide$subject <- factor(rawdata_wide$CaseID)

rawdata_wide

##   CaseID Gender Age Condition1 Condition2 subject
## 1      1      M   5      13.0      10.5      1
## 2      2      F   6      16.0      11.2      2
## 3      3      F   8      10.0      18.3      3
## 4      4      M   9      9.5      18.1      4
## 5      5      M  10      12.1     19.0      5

library(reshape2)

# Specify id.vars: the variables to keep (don't split apart on!)
melt(rawdata_wide, id.vars=c("CaseID", "Gender"))

##   CaseID Gender  variable value
## 1      1      M     Age     5
## 2      2      F     Age     6
## 3      3      F     Age     8
## 4      4      M     Age     9
## 5      5      M     Age    10
## 6      1      M Condition1 13
## 7      2      F Condition1 16
## 8      3      F Condition1 10
## 9      4      M Condition1 9.5
## 10     5      M Condition1 12.1
## 11     1      M Condition2 10.5
## 12     2      F Condition2 11.2
## 13     3      F Condition2 18.3
## 14     4      M Condition2 18.1
## 15     5      M Condition2 19
## 16     1      M  subject    1
## 17     2      F  subject    2
## 18     3      F  subject    3
## 19     4      M  subject    4
## 20     5      M  subject    5
```

There are options for `melt` that can make the output a little easier to work with:

```
data_Long <- melt(rawdata_wide,
  # ID variables - all the variables to keep but not split apart on
  id.vars=c("CaseID", "Gender"),
  # The source columns
  measure.vars=c("Age", "Condition1", "Condition2" ),
  # Name of the destination column that will identify the original
  # column that the measurement came from
  variable.name="Feature",
  value.name="Measurement"
)
data_Long

##   CaseID Gender   Feature Measurement
## 1      1      M       Age      5.0
## 2      2      F       Age      6.0
## 3      3      F       Age      8.0
## 4      4      M       Age      9.0
## 5      5      M       Age     10.0
## 6      1      M Condition1  13.0
## 7      2      F Condition1  16.0
## 8      3      F Condition1  10.0
## 9      4      M Condition1  9.5
## 10     5      M Condition1 12.1
## 11     1      M Condition2 10.5
## 12     2      F Condition2 11.2
## 13     3      F Condition2 18.3
## 14     4      M Condition2 18.1
## 15     5      M Condition2 19.0
```

For an elaborate justification, detailed description, and multiple examples of handling long-and-wide data, messy and tidy data, and data cleaning strategies see the (JSS Tidy Data article by Hadley Wickham) [<https://www.jstatsoft.org/article/view/v059i10>].

2.5 Data Generation

Popular data generation functions are `c()`, `seq()`, `rep()`, and `data.frame()`. Sometimes, we may also use `list()` and `array()` to generate data.

c()

`c()` creates a (column) vector. With option `recursive = T`, it descends through lists combining all elements into one vector.

```
a<-c(1, 2, 3, 5, 6, 7, 10, 1, 4)
a

## [1]  1  2  3  5  6  7 10  1  4

c(List(A = c(Z = 1, Y = 2), B = c(X = 7), C = c(W = 7, V=3, U=-1.9)), recursive = TRUE)

##  A.Z  A.Y  B.X  C.W  C.V  C.U
##  1.0  2.0  7.0  7.0  3.0 -1.9
```

When combined with `list()`, `c()` successfully created a vector with all the information in a list with three members A, B, and C.

seq(from, to)

`seq(from, to)` generates a sequence. Adding option `by =` can help us specify increment; Option `length=` specifies desired length. Also, `seq(along = x)` generates a sequence `1, 2, ..., length(x)`. This is used for loops to create ID for each element in `x`.

```
seq(1, 20, by=0.5)
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5
## [15] 8.0 8.5 9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5
## [29] 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0
seq(1, 20, length=9)
## [1] 1.000 3.375 5.750 8.125 10.500 12.875 15.250 17.625 20.000
seq(along=c(5, 4, 5, 6))
## [1] 1 2 3 4
```

rep(x, times)

`rep(x, times)` creates a sequence that repeats `x` a specified number of times. The option `each =` allows us to repeat first over each element of `x` certain number of times.

```
rep(c(1, 2, 3), 4)
## [1] 1 2 3 1 2 3 1 2 3 1 2 3
rep(c(1, 2, 3), each=4)
## [1] 1 1 1 1 2 2 2 2 3 3 3 3
```

Compare this to replicating using `replicate()`.

```
X <- seq(along=c(1, 2, 3)); replicate(4, X+1)
##      [,1] [,2] [,3] [,4]
## [1,]    2    2    2    2
## [2,]    3    3    3    3
## [3,]    4    4    4    4
```

data.frame()

`data.frame()` creates a data frame of named or unnamed arguments. We can combine multiple vectors. Each vector is stored as a column. Shorter vectors are recycled to the length of the longest one. With `data.frame()` you can mix numeric and characteristic vectors.

```
data.frame(v=1:4, ch=c("a", "B", "C", "d"), n=c(10, 11))
##   v ch  n
## 1 1 a 10
## 2 2 B 11
## 3 3 C 10
## 4 4 d 11
```

Note that the `1:4` means from 1 to 4. The operator `:` generates a sequence.

list()

Like we mentioned in function `c()`, `list()` creates a list of the named or unnamed arguments – indexing rule: from 1 to n, including 1 and n.

```
l<-List(a=c(1, 2), b="hi", c=-3+3i)
l

## $a
## [1] 1 2
##
## $b
## [1] "hi"
##
## $c
## [1] -3+3i

# Note Complex Numbers a <- -1+3i; b <- -2-2i; a+b
```

We use `$` to call each member in the list and `[[]]` to call the element corresponding to specific index. For example,

```
l$a[[2]]
## [1] 2
l$b
## [1] "hi"
```

Note that R uses *1-based numbering* rather than 0-based like some other languages (C/Java), so the first element of a list has index 1.

array(x, dim=)

`array(x, dim=)` creates an array with specific dimensions. For example, `dim = c(3, 4, 2)` means two 3×4 matrices. We use `[]` to extract specific elements in the array. `[2, 3, 1]` means the element at the second row third column in the first page. Leaving one number in the dimensions empty would help us to get a specific row, column or page. `[2, , 1]` means the second row in the 1st page. See this image (Fig. 2.3):

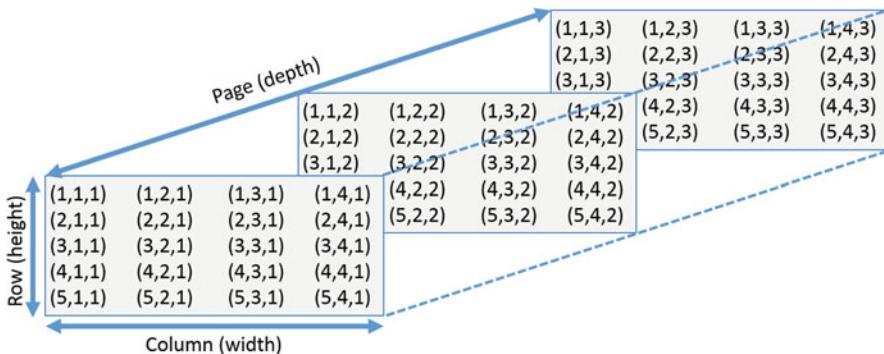


Fig. 2.3 Indexing cell values in multidimensional arrays (tensors)

```
ar <- array(1:24, dim=c(3, 4, 2)); ar
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]     1     4     7    10
## [2,]     2     5     8    11
## [3,]     3     6     9    12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    13    16    19    22
## [2,]    14    17    20    23
## [3,]    15    18    21    24
ar[2, 3, 1]
## [1] 8
ar[2, ,1]
## [1] 2 5 8 11
```

In general, multi-dimensional arrays are called “tensors” (of order = number of dimensions).

Other useful functions are:

- `matrix(x, nrow=, ncol=)`: creates matrix elements of `nrow` rows and `ncol` columns.
- `factor(x, levels=)`: encodes a vector `x` as a factor.
- `gl(n, k, length=n*k, labels = 1:n)`: generate levels (factors) by specifying the pattern of their levels. `k` is the number of levels, and `n` is the number of replications.
- `expand.grid()`: a data frame from all combinations of the supplied vectors or factors.
- `rbind()` combine arguments by rows for matrices, data frames, and others.
- `cbind()` combine arguments by columns for matrices, data frames, and others.

2.6 Input/Output (I/O)

The first pair of functions we will talk about are `load()`, which helps us reload datasets written with the `save()` function.

Let's create some data first.

```
x <- seq(1, 10, by=0.5)
y <- list(a = 1, b = TRUE, c = "oops")
save(x, y, file="xy.RData")
Load("xy.RData")
```

`data(x)` loads the specified data sets and `library(x)` loads the necessary add-on packages.

```
data("iris")
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
```

`read.table(file)` reads a file in table format and creates a data frame from it. The default separator `sep = ""` is any whitespace. Use `header = TRUE` to read the first line as a header of column names. Use `as.is = TRUE` to prevent character vectors from being converted to factors. Use `comment.char = ""` to prevent `"#"` from being interpreted as a comment. Use `skip = n` to skip `n` lines before reading data. See the help for options on row naming, NA treatment, and others.

Let's use `read.table()` to read a text file in our class file.

```
data.txt<-read.table("https://umich.instructure.com/files/1628628/download?d
ownLoad_frd=1", header=T, as.is = T) # 01a_data.txt
summary(data.txt)

##           Name          Team          Position          Height
##  Length:1034    Length:1034    Length:1034    Min.   :67.0
##  Class :character  Class :character  Class :character  1st Qu.:72.0
##  Mode  :character  Mode  :character  Mode  :character  Median :74.0
##                                         Mean   :73.7
##                                         3rd Qu.:75.0
##                                         Max.   :83.0
##
##           Weight          Age
##  Min.   :150.0  Min.   :20.90
##  1st Qu.:187.0  1st Qu.:25.44
##  Median :200.0  Median :27.93
##  Mean   :201.7  Mean   :28.74
##  3rd Qu.:215.0  3rd Qu.:31.23
##  Max.   :290.0  Max.   :48.52
```

`read.csv("filename", header = TRUE)` is identical to `read.table()` but with defaults set for reading comma-delimited files.

```
data.csv<-read.csv("https://umich.instructure.com/files/1628650/download?down
nload_frd=1", header = T) # 01_hdp.csv
summary(data.csv)

##   tumorsize          co2          pain          wound
##   Min.   : 33.97   Min.   :1.222   Min.   :1.000   Min.   :1.000
##   1st Qu.: 62.49   1st Qu.:1.519   1st Qu.:4.000   1st Qu.:5.000
##   Median : 70.07   Median :1.601   Median :5.000   Median :6.000
##   Mean   : 70.88   Mean   :1.605   Mean   :5.473   Mean   :5.732
##   3rd Qu.: 79.02   3rd Qu.:1.687   3rd Qu.:6.000   3rd Qu.:7.000
##   Max.   :116.46   Max.   :2.128   Max.   :9.000   Max.   :9.000
##   mobility          ntumors        nmorphine      remission
##   Min.   :1.00      Min.   :0.000   Min.   :0.000   Min.   :0.0000
##   1st Qu.:5.00      1st Qu.:1.000   1st Qu.:2.000   1st Qu.:0.0000
##   Median :6.00      Median :3.000   Median :3.000   Median :0.0000
##   Mean   :6.08      Mean   :3.066   Mean   :3.624   Mean   :0.2957
##   3rd Qu.:7.00      3rd Qu.:5.000   3rd Qu.:5.000   3rd Qu.:1.0000
##   Max.   :9.00      Max.   :9.000   Max.   :18.000  Max.   :1.0000
##   Lungcapacity      Age          Married      FamilyHx      SmokingHx
##   Min.   :0.01612   Min.   :26.32   Min.   :0.0      no   :6820   current:1705
##   1st Qu.:0.67647   1st Qu.:46.69   1st Qu.:0.0      yes  :1705   former :1705
##   Median :0.81560   Median :50.93   Median :1.0      never :5115
##   Mean   :0.77409   Mean   :50.97   Mean   :0.6
##   3rd Qu.:0.91150   3rd Qu.:55.27   3rd Qu.:1.0
##   Max.   :0.99980   Max.   :74.48   Max.   :1.0
##   Sex          CancerStage LengthofStay          WBC          RBC
##   female:5115   I   :2558   Min.   : 1.000   Min.   :2131   Min.   :3.919
##   male  :3410    II  :3409   1st Qu.: 5.000   1st Qu.:5323   1st Qu.:4.802
##                  III:1705   Median : 5.000   Median :6007   Median :4.994
##                  IV : 853   Mean   : 5.492   Mean   :5998   Mean   :4.995
##                               3rd Qu.: 6.000   3rd Qu.:6663   3rd Qu.:5.190
##                               Max.   :10.000  Max.   :9776   Max.   :6.065
##   BMI          IL6          CRP          DID
##   Min.   :18.38   Min.   : 0.03521   Min.   : 0.0451   Min.   : 1.0
##   1st Qu.:24.20   1st Qu.: 1.93039   1st Qu.: 2.6968   1st Qu.:100.0
##   Median :27.73   Median : 3.34400   Median : 4.3330   Median :199.0
##   Mean   :29.07   Mean   : 4.01698   Mean   : 4.9730   Mean   :203.3
##   3rd Qu.:32.54   3rd Qu.: 5.40551   3rd Qu.: 6.5952   3rd Qu.:309.0
##   Max.   :58.00   Max.   :23.72777   Max.   :28.7421   Max.   :407.0
##   Experience      School        Lawsuits      HID
##   Min.   : 7.00   average:6405   Min.   :0.000   Min.   : 1.00
##   1st Qu.:15.00   top    :2120    1st Qu.:1.000   1st Qu.: 9.00
##   Median :18.00
##   Mean   :17.64
##   3rd Qu.:21.00
##   Max.   :29.00
##   Medicaid
##   Min.   :0.1416
##   1st Qu.:0.3369
##   Median :0.5215
##   Mean   :0.5125
##   3rd Qu.:0.7083
##   Max.   :0.8187
```

`read.delim("filename", header = TRUE)` is very similar to the first two. However, it has defaults set for reading tab-delimited files.

Also, we have `read.fwf(file, widths, header = FALSE, sep = "\t", as.is = FALSE)` to read a table of fixed width formatted data into a data frame.

`match(x, y)` returns a vector of the positions of (first) matches of its first argument in its second. For a specific element in `x`, if no elements matches it in `y`, the output for that elements would be `NA`.

```
match(c(1, 2, 4, 5), c(1, 4, 4, 5, 6, 7))
## [1] 1 NA 2 4
```

`save.image(file)` saves all objects in the current work space.

`write.table(x, file = "", row.names = TRUE, col.names = TRUE, sep = "")` prints `x` after converting to a data frame and stores it into a specified file. If `quote` is `TRUE`, character or factor columns are surrounded by quotes (""). `sep` is the field separator. `eol` is the end-of-line separator. `na` is the string for missing values. Use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input.

Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `File = ""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables. On windows, the file connection can also be used with `description = "clipboard"`. To read a table copied from Excel, use `x <- read.delim("clipboard")`.

To write a table to the clipboard for Excel, use `write.table(x, "clipboard", sep = "\t", col.names = NA)`. For database interaction, see packages RODBC, DBI, RMySQL, RPgSQL, and ROracle, as well as packages XML, hdf5, netCDF for reading other file formats. We will talk about some of them in later Chapters.

Note, an alternative library called `rio` handles import/export of multiple data types using a simple syntax.

2.7 Slicing and Extracting Data

Table 2.1 shows us how to index vectors.

Indexing lists are similar to indexing vectors, but some of the symbols are different (Table 2.2).

Indexing for matrices is a higher dimensional version of indexing vectors (Table 2.3).

Table 2.1 Vector indexing in R

Expression	Explanation
<code>x[n]</code>	Nth element
<code>x[-n]</code>	All but the nth element
<code>x[1:n]</code>	First n elements
<code>x[-(1:n)]</code>	Elements from n + 1 to the end
<code>x[c(1, 4, 2)]</code>	Specific elements
<code>x["name"]</code>	Element named "name"
<code>x[x > 3]</code>	All elements greater than 3
<code>x[x > 3 & x < 5]</code>	All elements between 3 and 5
<code>x[x %in% c("a", "and", "the")]</code>	Elements in the given set

Table 2.2 List indexing in R

Expression	Explanation
<code>x[n]</code>	List with n elements
<code>x[[n]]</code>	Nth element of the list
<code>x[["name"]]</code>	Element of the list named "name"

Table 2.3 Matrix indexing in R

Expression	Explanation
<code>x[i, j]</code>	Element at row i, column j
<code>x[i,]</code>	Row i
<code>x[, j]</code>	Column j
<code>x[, c(1, 3)]</code>	Columns 1 and 3
<code>x[["name"],]</code>	Row named "name"

2.8 Variable Conversion

The following functions can be used to convert data types:

`as.array(x)`, `as.data.frame(x)`, `as.numeric(x)`, `as.logical(x)`,
`as.complex(x)`, `as.character(x)`, ...

Typing `methods(as)` in the console will generate a complete list for variable conversion functions.

2.9 Variable Information

The following functions will test if the each data element is a specific type:

`is.na(x)`, `is.null(x)`, `is.array(x)`, `is.data.frame(x)`, `is.numeric(x)`, `is.complex(x)`, `is.character(x)`, ...

For a complete list, type `methods(is)` in R console. The output for these functions are a bunch of TRUE or FALSE logical statements. One statement for one element in the dataset.

length(x) gives us the number of elements in x.

```
x<-c(1, 3, 10, 23, 1, 3)
Length(x)
## [1] 6
```

dim(x) retrieves or sets the dimension of an object.

```
x<-1:12
dim(x)<-c(3, 4)
x
##      [,1] [,2] [,3] [,4]
## [1,]     1     4     7    10
## [2,]     2     5     8    11
## [3,]     3     6     9    12
```

dimnames(x) retrieves or sets the dimension names of an object. For higher dimensional objects like matrix or arrays we can combine dimnames () with list.

```
dimnames(x)<-list(c("R1", "R2", "R3"), c("C1", "C2", "C3", "C4")); x
##      C1 C2 C3 C4
## R1  1  4  7 10
## R2  2  5  8 11
## R3  3  6  9 12
```

nrow(x) number of rows; **ncol(x)** number of columns.

```
nrow(x)
## [1] 3
ncol(x)
## [1] 4
```

class(x) get or set the class of x. Note that we can use **unclass(x)** to remove the class attribute of x.

```
class(x)
## [1] "matrix"
class(x)<-"myclass"
x<-unclass(x)
x
##      C1 C2 C3 C4
## R1  1  4  7 10
## R2  2  5  8 11
## R3  3  6  9 12
```

attr(x, which) get or set the attribute which of x.

```
attr(x, "class")
## NULL
attr(x, "dim")<-c(2, 6)
x
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]     1     3     5     7     9    11
## [2,]     2     4     6     8    10    12
```

From the above commands we know that when we unclass x, its class would be `NULL`.

attributes(obj) get or set the list of attributes of object.

```
attributes(x) <- List(mycomment = "really special", dim = 3:4,
  dimnames = List(LETTERS[1:3], letters[1:4]), names = paste(1:12))
x
##   a b c d
## A 1 4 7 10
## B 2 5 8 11
## C 3 6 9 12
## attr(,"mycomment")
## [1] "really special"
## attr(,"names")
## [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
```

2.10 Data Selection and Manipulation

In this section, we will introduce some data manipulation functions. In addition, tools from `dplyr` provide easy dataset manipulation routines.

which.max(x) returns the index of the greatest element of x. **which.min(x)** returns the index of the smallest element of x. **rev(x)** reverses the elements of x. Let's see these three functions first.

```
x<-c(1, 5, 2, 1, 10, 40, 3)
which.max(x)
## [1] 6
which.min(x)
## [1] 1
rev(x)
## [1] 3 40 10 1 2 5 1
```

sort(x) sorts the elements of x in increasing order. To sort in decreasing order we can use **rev(sort(x))**.

```
sort(x)
## [1] 1 1 2 3 5 10 40
rev(sort(x))
## [1] 40 10 5 3 2 1 1
```

cut(x, breaks) divides x into intervals with same length (sometimes factors). **breaks** is the number of cut intervals or a vector of cut points. Cut divides the range of x into intervals coding the values in x according to the intervals they fall into.

```
x
## [1] 1 5 2 1 10 40 3
cut(x, 3)
## [1] (0.961,14] (0.961,14] (0.961,14] (0.961,14] (0.961,14] (27,40]
## [7] (0.961,14]
## Levels: (0.961,14] (14,27] (27,40]
cut(x, c(0, 5, 20, 30))
## [1] (0,5] (0,5] (0,5] (0,5] (5,20] <NA> (0,5]
## Levels: (0,5] (5,20] (20,30]
```

which(x == a) returns a vector of the indices of x if the comparison operation is true (TRUE). For example it returns the value i, if x[i] == a is true. Thus, the argument of this function (like x==a) must be a variable of mode logical.

```
x
## [1] 1 5 2 1 10 40 3
which(x==2)
## [1] 3
```

na.omit(x) suppresses the observations with missing data (NA). It suppresses the corresponding line if x is a matrix or a data frame. **na.fail(x)** returns an error message if x contains at least one NA.

```
df<-data.frame(a=1:5, b=c(1, 3, NA, 9, 8)); df
##   a   b
## 1 1   1
## 2 2   3
## 3 3 NA
## 4 4   9
## 5 5   8

na.omit(df)

##   a   b
## 1 1   1
## 2 2   3
## 4 4   9
## 5 5   8
```

unique(x) If x is a vector or a data frame, it returns a similar object but with the duplicate elements suppressed.

```
df1<-data.frame(a=c(1, 1, 7, 6, 8), b=c(1, 1, NA, 9, 8))
df1
##   a   b
## 1 1   1
## 2 1   1
## 3 7 NA
## 4 6   9
## 5 8   8

unique(df1)

##   a   b
## 1 1   1
## 3 7 NA
## 4 6   9
## 5 8   8
```

table(x) returns a table with the different values of x and their frequencies (typically for integers or factors). Also check `prob.table()`.

```
v<-c(1, 2, 4, 2, 2, 5, 6, 4, 7, 8, 8)
table(v)

## v
## 1 2 4 5 6 7 8
## 1 3 2 1 1 1 2
```

subset(x, ...) returns a selection of x with respect to criteria . . . (typically . . . are comparisons like `x$V1 < 10`). If x is a data frame, the option `select =` gives the variables to be kept or dropped using a minus sign.

```

sub<-subset(df1, df1$a>5); sub
##   a   b
## 3 7 NA
## 4 6  9
## 5 8  8

sub<-subset(df1, select=-a)
sub

##   b
## 1 1
## 2 1
## 3 NA
## 4 9
## 5 8

```

sample(x, size) resamples randomly and without replacement size elements in the vector x, the option `replace = TRUE` allows to resample with replacement.

```

v
##  [1] 1 2 4 2 2 5 6 4 7 8 8
sample(df1$a, 20, replace = T)
##  [1] 7 8 1 6 1 1 7 8 1 7 8 1 6 7 8 7 1 6 8 8

```

prop.table(x, margin=) table entries as fraction of marginal table.

```

prop.table(table(v))

## v
##      1          2          4          5          6          7
## 0.09090909 0.27272727 0.18181818 0.09090909 0.09090909 0.09090909
##      8
## 0.18181818

```

2.11 Math Functions

Basic math functions like `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `log`, `log10`, `exp`. and “set” functions `union(x, y)`, `intersect(x, y)`, `setdiff(x, y)`, `setequal(x, y)`, `is.element(el, set)` are available in R.

`lsf.str("package:base")` displays all base functions built in a specific R package (like base).

Also we have the Table 2.4 of functions that you might need when using R for calculations.

Note: many math functions have a logical parameter `na.rm. = FALSE` to specify missing data (NA) removal.

Table 2.4 Common mathematics, statistics, and processing R functions

Expression	Explanation
<code>choose (n, k)</code>	Computes the combinations of k events among n objects. Mathematically it equals to $\frac{n!}{(n-k)!k!}$
<code>max (x)</code>	Maximum of the elements of x
<code>min (x)</code>	Minimum of the elements of x
<code>range (x)</code>	Minimum and maximum of the elements of x
<code>sum (x)</code>	Sum of the elements of x
<code>diff (x)</code>	Lagged and iterated differences of vector x
<code>prod (x)</code>	Product of the elements of x
<code>mean (x)</code>	Mean of the elements of x
<code>median (x)</code>	Median of the elements of x
<code>quantile (x, probs=)</code>	Sample quantiles corresponding to the given probabilities (defaults to 0, 0.25, 0.5, 0.75, 1)
<code>weighted.mean (x, w)</code>	Mean of x with weights w
<code>rank (x)</code>	Ranks of the elements of x
<code>var (x) or cov (x)</code>	Variance of the elements of x (calculated on n > 1). If x is a matrix or a data frame, the variance-covariance matrix is calculated
<code>sd (x)</code>	Standard deviation of x
<code>cor (x)</code>	Correlation matrix of x if it is a matrix or a data frame (1 if x is a vector)
<code>var (x, y) or cov (x, y)</code>	Covariance between x and y, or between the columns of x and those of y if they are matrices or data frames
<code>cor (x, y)</code>	Linear correlation between x and y, or correlation matrix if they are matrices or data frames
<code>round (x, n)</code>	Rounds the elements of x to n decimals
<code>log (x, base)</code>	Computes the logarithm of x with base "base"
<code>scale (x)</code>	If x is a matrix, centers and reduces the data. Without centering use the option <code>center = FALSE</code> . Without scaling use <code>scale = FALSE</code> (by default <code>center = TRUE</code> , <code>scale = TRUE</code>)
<code>pmin (x, y, ...)</code>	a vector which ith element is the minimum of x[i], y[i],...
<code>pmax (x, y, ...)</code>	a vector which ith element is the maximum of x[i], y[i],...
<code>cumsum (x)</code>	a vector which ith element is the sum from x[1] to x[i]
<code>cumprod (x)</code>	id. for the product
<code>cummin (x)</code>	id. for the minimum
<code>cummax (x)</code>	id. for the maximum
<code>Re (x)</code>	Real part of a complex number
<code>Im (x)</code>	Imaginary part of a complex number
<code>Mod (x)</code>	Modulus. <code>Abs (x)</code> is the same
<code>Arg (x)</code>	Angle in radians of the complex number
<code>Conj (x)</code>	Complex conjugate
<code>convolve (x, y)</code>	Compute the several kinds of convolutions of two sequences
<code>fft (x)</code>	Fast Fourier Transform of an array
<code>mvfft (x)</code>	FFT of each column of a matrix
<code>filter (x, filter)</code>	Applies linear filtering to a univariate time series or to each series separately of a multivariate time series

2.12 Matrix Operations

The following table summarizes basic operation functions. We will discuss this topic in detail in [Chap. 5](#) (Table 2.5).

```
mat1 <- cbind(c(1, -1/5), c(-1/3, 1))
mat1.inv <- solve(mat1)

mat1.identity <- mat1.inv %*% mat1
mat1.identity

##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1

b <- c(1, 2)
x <- solve (mat1, b)
x

## [1] 1.785714 2.357143
```

2.13 Advanced Data Processing

In this section, we will introduce some fancy functions that can save time remarkably.

apply(X, INDEX, FUN=) a vector or array or list of values obtained by applying a function FUN to margins (INDEX = 1 means row, INDEX = 2 means column) of X.

Table 2.5 Common R operators

Expression	Explanation
<code>t (x)</code>	Transpose
<code>diag (x)</code>	Diagonal
<code>%*%</code>	Matrix multiplication
<code>solve (a, b)</code>	Solves a %*% x = b for x
<code>solve (a)</code>	Matrix inverse of a
<code>rowsum (x)</code>	Sum of rows for a matrix-like object. <code>rowSums (x)</code> is a faster version
<code>colSums (x)</code>	id. for columns
<code>rowMeans (x)</code>	Fast version of row means
<code>colMeans (x)</code>	id. for columns

```
df1
##   a   b
## 1 1   1
## 2 1   1
## 3 7   NA
## 4 6   9
## 5 8   8

apply(df1, 2, mean, na.rm=T)
##      a      b
## 4.60 4.75
```

Note that we can add options for the FUN after the function.

lapply(X, FUN) apply FUN to each member of the list X. If X is a data frame, it will apply the FUN to each column and return a list.

```
lapply(df1, mean, na.rm=T)
## $a
## [1] 4.6
##
## $b
## [1] 4.75

lapply(list(a=c(1, 23, 5, 6, 1), b=c(9, 90, 999)), median)
## $a
## [1] 5
##
## $b
## [1] 90
```

tapply(X, INDEX, FUN=) apply FUN to each cell of a ragged array given by X with indexes equals to INDEX. Note that X is an atomic object, typically a vector.

```
v
## [1] 1 2 4 2 2 5 6 4 7 8 8

fac <- factor(rep(1:3, length = 11), levels = 1:3)
table(fac)

## fac
## 1 2 3
## 4 4 3

tapply(v, fac, sum)
## 1 2 3
## 17 16 16
```

by(data, INDEX, FUN) apply FUN to data frame data subsetted by INDEX.

```
by(df1, df1[, 1], sum)
## df1[, 1]: 1
## [1] 4
## -----
## df1[, 1]: 6
## [1] 15
## -----
## df1[, 1]: 7
## [1] NA
## -----
## df1[, 1]: 8
## [1] 16
```

This code applies the `sum` function to `df1` using column 1 as an index.

merge(a, b) merge two data frames by common columns or row names. We can use option `by` = to specify the index column.

```
df2<-data.frame(a=c(1, 1, 7, 6, 8), c=1:5)
df2
##   a   c
## 1 1   1
## 2 1   2
## 3 7   3
## 4 6   4
## 5 8   5

df3<-merge(df1, df2, by="a")
df3
##   a   b   c
## 1 1   1   1
## 2 1   1   2
## 3 1   1   1
## 4 1   1   2
## 5 6   9   4
## 6 7  NA   3
## 7 8   8   5
```

xtabs(a ~ b, data = x) a contingency table from cross-classifying factors.

```
DF <- as.data.frame(UCBAdmissions)
## 'DF' is a data frame with a grid of the factors and the counts
## in variable 'Freq'.
DF

##      Admit Gender Dept Freq
## 1  Admitted   Male    A  512
## 2  Rejected   Male    A  313
## 3  Admitted Female   A   89
...
## 23 Admitted Female    F   24
## 24 Rejected Female    F  317

## Nice for taking margins ...
xtabs(Freq ~ Gender + Admit, DF)

##      Admit
## Gender Admitted Rejected
##   Male      1198      1493
## Female      557      1278

## And for testing independence ...
summary(xtabs(Freq ~ ., DF))

## Call: xtabs(formula = Freq ~ ., data = DF)
## Number of cases in table: 4526
## Number of factors: 3
## Test for independence of all factors:
##  Chisq = 2000.3, df = 16, p-value = 0
```

aggregate(x, by, FUN) splits the data frame x into subsets, computes summary statistics for each, and returns the result in a convenient form. by is a list of grouping elements, that each have the same length as the variables in x.

```
list(rep(1:3, Length=7))

## [[1]]
## [1] 1 2 3 1 2 3 1

aggregate(df3, by=list(rep(1:3, Length=7)), sum)

##   Group.1  a  b  c
## 1        1 10 10  8
## 2        2  7 10  6
## 3        3  8 NA  4
```

The above code applied the function `sum` to data frame `df3` according to the index created by `list(rep(1:3, length=7))`.

stack(x, ...) transform data, stored as separate columns in a data frame or a list, into a single column and **unstack(x, ...)** is the inverse of `stack()`.

```

stack(df3)

##   values ind
## 1      1   a
## 2      1   a
## 3      1   a
...
## 20     3   c
## 21     5   c

unstack(stack(df3))

##   a   b   c
## 1 1  1  1
## 2 1  1  2
## 3 1  1  1
## 4 1  1  2
## 5 6  9  4
## 6 7 NA 3
## 7 8  8  5

```

reshape(x, ...) reshapes a data frame between "wide" format with repeated measurements in separate columns of the same record and "long" format with the repeated measurements in separate records. Use `direction = "wide"` or `direction = "long"`.

```

df4 <- data.frame(school = rep(1:3, each = 4), class = rep(9:10, 6),
                     time = rep(c(1, 1, 2, 2), 3), score = rnorm(12))
wide <- reshape(df4, idvar = c("school", "class"), direction = "wide")
wide

##   school class   score.1   score.2
## 1       1    9 -0.1575202 -1.415503816
## 2       1   10  0.5804452  1.754559537
## 5       2    9  0.1553872  1.693809827
## 6       2   10 -0.7540783  0.478035367
## 9       3    9 -0.6490757 -0.002922609
## 10      3   10 -0.2122064  0.276259031

Long <- reshape(wide, idvar = c("school", "class"), direction = "long")
Long

##       school class time   score.1
## 1.9.1     1    9    1 -0.157520208
## 1.10.1    1   10    1  0.580445243
## 2.9.1     2    9    1  0.155387189
## 2.10.1    2   10    1 -0.754078345
## 3.9.1     3    9    1 -0.649075721
## 3.10.1    3   10    1 -0.212206430
## 1.9.2     1    9    2 -1.415503816
## 1.10.2    1   10    2  1.754559537
## 2.9.2     2    9    2  1.693809827
## 2.10.2    2   10    2  0.478035367
## 3.9.2     3    9    2 -0.002922609
## 3.10.2    3   10    2  0.276259031

```

Notes

- The *x* in this function has to be longitudinal data.
- The call to `rnorm` used in `reshape` might generate different results for each call, unless `set.seed(1234)` is used to ensure reproducibility of random-number generation.

2.14 Strings

The following functions are useful for handling strings in R.

`paste(...)` concatenates vectors after converting to character. It has a few options. `Sep` = is the string to separate terms (a single space is the default). `collapse` = is an optional string to separate “collapsed” results.

```
a<- "today"
b<- "is a good day"
paste(a, b)

## [1] "today is a good day"

paste(a, b, sep= ", ")
## [1] "today, is a good day"
```

`substr(x, start, stop)` substrings in a character vector. It can also assign values (with the same length) to part of a string, as `substr(x, start, stop) <- value`.

```
a<- "When the going gets tough, the tough get going!"
substr(a, 10, 40)

## [1] "going gets tough, the tough get"
substr(a, 1, 9)<- "...."
a

## [1] ".....going gets tough, the tough get going!"
```

Note that characters at `start` and `stop` indexes are inclusive in the output.

`strsplit(x, split)` split x according to the substring split. Use `fixed = TRUE` for non-regular expressions.

```
strsplit("a.b.c", ".", fixed = TRUE)

## [[1]]
## [1] "a" "b" "c"
```

`grep(pattern, x)` searches for matches to pattern within x. It will return a vector of the indices of the elements of x that yielded a match. Use regular expression for `pattern` (unless `fixed = TRUE`). See `?regex` for details.

```
letters
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
## [18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
grep("[a-z]", letters)
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26
```

gsub(pattern, replacement, x) replacement of matches determined by regular expression matching. **sub()** is the same but only replaces the first occurrence.

```
a<-c("e", 0, "kj", 10, ";")
gsub("[a-z]", "Letters", a)
## [1] "Letters"      "0"           "Letters" "10"
## [5] ";"
sub("[a-z]", "Letters", a)
## [1] "Letters" "0"      "Letters" "10"      ";"
```

tolower(x) convert to lowercase. **toupper(x)** convert to uppercase.

match(x, table) a vector of the positions of first matches for the elements of x among table, with a short hand x %in% table, which returns a logical vector.

```
x<-c(1, 2, 10, 19, 29)
match(x, c(1, 10))
## [1] 1 NA 2 NA NA
x %in% c(1, 10)
## [1] TRUE FALSE TRUE FALSE FALSE
pmatch(x, table) partial matches for the elements of x among table.
pmatch("m", c("mean", "median", "mode")) # returns NA
## [1] NA
pmatch("med", c("mean", "median", "mode")) # returns 2
## [1] 2
```

The first one returns NA, and dependent on the R-version, possibly a warning, because all elements have the pattern "m".

nchar(x) number of characters in x.

Dates and Times

The class **Date** has dates without times. **POSIXct()** has dates and times, including time zones. Comparisons (e.g. **>**), **seq()**, and **difftime()** are useful. **?DateTimeClasses** gives more information. See also package **chron**.

as.Date(s) and **as.POSIXct(s)** convert to the respective class; **format(dt)** converts to a string representation. The default string format is 2001-02-21.

Table 2.6 R date formatting specifications

Formats	Explanations
<code>%a, %A</code>	Abbreviated and full weekday name.
<code>%b, %B</code>	Abbreviated and full month name.
<code>%d</code>	Day of the month (01 ... 31).
<code>%H</code>	Hours (00 ... 23).
<code>%I</code>	Hours (01 ... 12).
<code>%j</code>	Day of year (001 ... 366).
<code>%m</code>	Month (01 ... 12).
<code>%M</code>	Minute (00 ... 59).
<code>%p</code>	AM/PM indicator.
<code>%S</code>	Second as decimal number (00 ... 61).
<code>%U</code>	Week (00 ... 53); the first Sunday as day 1 of week 1.
<code>%w</code>	Weekday (0 ... 6, Sunday is 0).
<code>%W</code>	Week (00 ... 53); the first Monday as day 1 of week 1.
<code>%y</code>	Year without century (00 ... 99). Don't use.
<code>%Y</code>	Year with century.
<code>%z (output only)</code>	Offset from Greenwich; -0800 is 8 hours west of Greenwich Meridian.
<code>%Z (output only)</code>	Time zone as a character string (empty if not available).

These accept a second argument to specify a format for conversion. Some common formats are (Table 2.6):

Where leading zeros are shown they will be used on output but are optional on input. See `?strptime` for details.

2.15 Plotting

This is only an introduction for plotting functions in R. In Chap. 4, we will discuss visualization in more detail.

plot(x) plot of the values of x (on the y-axis) ordered on the x-axis.

plot(x, y) bivariate plot of x (on the x-axis) and y (on the y-axis).

hist(x) histogram of the frequencies of x.

barplot(x) histogram of the values of x. Use `horiz = FALSE` for horizontal bars.

dotchart(x) if x is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column).

pie(x) circular pie-chart.

boxplot(x) ‘box-and-whiskers’ plot.

sunflowerplot(x, y) id. than `plot()` but the points with similar coordinates are drawn as flowers which petal number represents the number of points.

stripplot(x) plot of the values of x on a line (an alternative to `boxplot()` for small sample sizes).

coplot(x~y | z) bivariate plot of x and y for each value or interval of values of z.

interaction.plot (f1, f2, y) if f1 and f2 are factors, plots the means of y (on the y-axis) with respect to the values of f1 (on the x-axis) and of f2 (different curves). The option fun allows choosing the summary statistic of y (by default fun = mean).

matplot(x, y) bivariate plot of the first column of x vs. the first one of y, the second one of x vs. the second one of y, etc.

fourfoldplot(x) visualizes, with quarters of circles, the association between two dichotomous variables for different populations (x must be an array with dim = c(2, 2, k), or a matrix with dim = c(2, 2) if k = 1).

assocplot(x) Cohen's Friendly graph shows the deviations from independence of rows and columns in a two dimensional contingency table.

mosaicplot(x) “mosaic” “graph of the residuals from a log-linear regression of a contingency table.

pairs(x) if x is a matrix or a data frame, draws all possible bivariate plots between the columns of x.

plot.ts(x) if x is an object of class “ts”, it plots x with respect to time. x may be multivariate but the series must have the same frequency and dates. Detailed examples are in Chap. 19: **Big Longitudinal Data Analysis**.

ts.plot(x) id. but if x is multivariate the series may have different dates and must have the same frequency.

qqnorm(x) quantiles of x with respect to the values expected under a normal law.

qqplot(x, y) quantiles of y with respect to the quantiles of x.

contour(x, y, z) contour plot (data are interpolated to draw the curves), x and y must be vectors and z must be a matrix so that `dim(z) = c(length(x), length(y))` (x and y may be omitted).

filled.contour(x, y, z) areas between the contours are colored, and a legend of the colors is drawn as well.

image(x, y, z) plotting actual data with colors.

persp(x, y, z) plotting actual data in perspective view.

stars(x) if x is a matrix or a data frame, draws a graph with segments or a star where each row of x is represented by a star and the columns are the lengths of the segments.

symbols(x, y, ...) draws, at the coordinates given by x and y, symbols (circles, squares, rectangles, stars, thermometers or “boxplots”) which sizes, colors, etc. are specified by supplementary arguments.

termplot(mod.obj) plot of the (partial) effects of a regression model (mod.obj).

The following parameters are common to many plotting functions (Table 2.7):

Table 2.7 Basic R plotting parameters

Parameters	Explanations
<code>add=FALSE</code>	If TRUE superposes the plot on the previous one (if it exists)
<code>axes=TRUE</code>	If FALSE does not draw the axes and the box
<code>type = "p"</code>	Specifies the type of plot, "p": Points, "l": Lines, "b": Points connected by lines, "o": Id. But the lines are over the points, "h": Vertical lines, "s": Steps, the data are represented by the top of the vertical lines, "S": Id. However, the data are represented at the bottom of the vertical lines
<code>xlim=</code> , <code>ylim=</code>	Specifies the lower and upper limits of the axes, for example with <code>xlim = c(1, 10)</code> or <code>xlim = range(x)</code>
<code>xlab=</code> , <code>ylab=</code>	Annotates the axes, must be variables of mode character
<code>main=</code>	Main title, must be a variable of mode character
<code>sub=</code>	Subtitle (written in a smaller font)

2.16 QQ Normal Probability Plot

Let's look at one simple example – quantile-quantile probability plot. Suppose $X \sim N(0, 1)$ and $Y \sim Cauchy$ represent the observed/raw and simulated/synthetic data for one feature (variable) in the data (Figs. 2.4, 2.5, 2.6 and 2.7).

```
X <- rnorm(1000)
Y <- rcauchy(1500)

# compare X to StdNormal distribution
qqnorm(X,
       main="Normal Q-Q Plot of the data",
       xLab="Theoretical Quantiles of the Normal",
       yLab="Sample Quantiles of the X (Normal) Data")
qqline(X)

qqplot(X, Y)

# Y against StdNormal
qqnorm(Y,
       main="Normal Q-Q Plot of the data",
       xLab="Theoretical Quantiles of the Normal",
       yLab="Sample Quantiles of the Y (Cauchy) Data", ylim= range(-4, 4))
# Why is the y-range specified here?
qqline(Y)

# Q-Q plot data (X) vs. simulation(Y)
myQQ <- function(x, y, ...) {
  #rang <- range(x, y, na.rm=T)
  rang <- range(-4, 4, na.rm=T)
  qqplot(x, y, xlim=rang, ylim=rang)
}
myQQ(X, Y) # where the Y is the newly simulated data for X
qqline(X)
```

Fig. 2.4 Quantile-quantile plot comparing the sample distribution to normal distribution

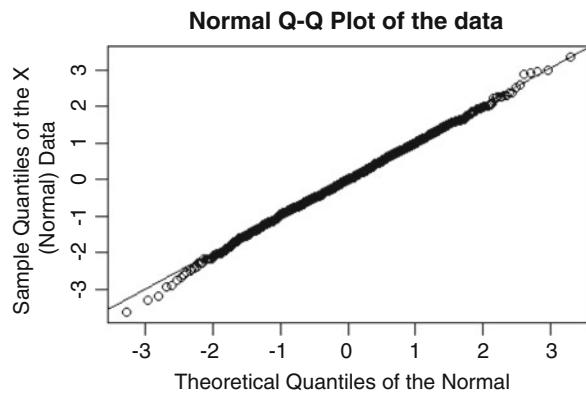


Fig. 2.5 Comparing a Cauchy sample distribution to Normal sample distribution via Q-Q plot

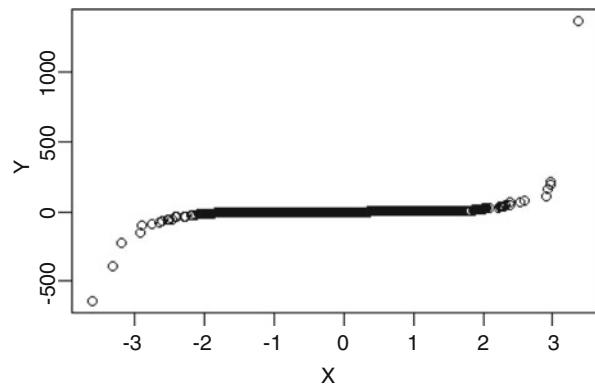


Fig. 2.6 Comparing Cauchy sample to Normal quantiles

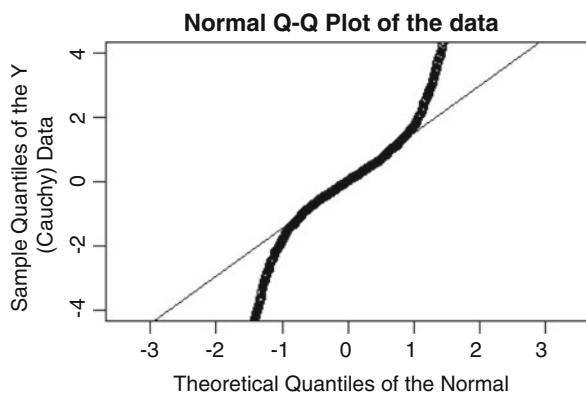
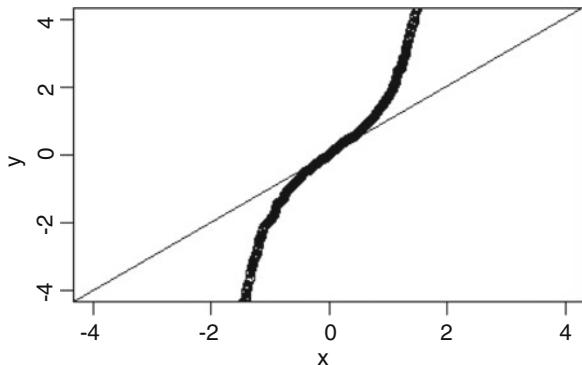


Fig. 2.7 Using isotropic scales to compare Cauchy sample to Normal quantiles



```

# Subsampling
x <- matrix(rnorm(100), ncol = 5)
y <- c(1, seq(19))
z <- cbind(x, y)
z.df <- data.frame(z)
z.df

##           V1          V2          V3          V4          V5   y
## 1 -0.5202336  0.5695642 -0.8104910 -0.775492348 1.8310536  1
## 2 -1.4370163 -3.0437691 -0.4895970 -0.018963095 2.2980451  1
## 3  1.1510882 -1.5345341 -0.5443915  1.176473324 -0.9079013  2
## 4  0.2937683 -1.1738992  1.1329062  0.050817201 -0.1975722  3
## 5  0.1011329  1.1382172 -0.3353099  1.980538873 -1.4902878  4
## 6 -0.3842767  1.7629568 -0.1734520  0.009448173  0.4166688  5
## 7 -0.1897151 -0.2928122  0.9917801  0.147767309 -0.3447306  6
## 8 -1.5184068 -0.6339424 -1.4102368  0.471592965  1.0748895  7
## 9 -0.6475764  0.3884220  1.5151532 -1.977356193 -0.9561620  8
## 10  0.1476949 -0.2219758  0.6255156 -0.755406330 -0.3411347  9
## 11  1.1927071 -0.2031697  0.6926743  1.263878207 -0.2628487 10
## 12  0.6117842 -0.3206093 -1.0544746  0.074048308 -0.3483535 11
## 13  1.7865743 -0.9457715 -0.2907310  1.520606318  2.3182403 12
## 14 -0.2075467  0.6440087  0.6277978 -1.670570757  0.1356807 13
## 15  0.2087459  1.2049360  1.2614003  1.102632278  0.4413631 14
## 16 -0.8663415 -0.4149625  1.3974565  0.432508163 -0.7408295 15
## 17 -0.4808447  0.6163081 -0.8693709 -0.830734957 -0.2094428 16
## 18 -0.3456697  2.5622196 -0.9398627  0.363765941 -1.4032376 17
## 19  1.1240451 -0.1887518 -0.6514363 -0.988661412 -1.2906608 18
## 20 -0.9783920  1.0246003 -0.6001832 -0.568181332  0.2374808 19

names(z.df)

## [1] "V1" "V2" "V3" "V4" "V5" "y"
# subsetting rows
z.sub <- subset(z.df, y > 2 & (y<10 | V1>0))
z.sub

```

```

##          V1          V2          V3          V4          V5  y
## 4  0.2937683 -1.1738992  1.1329062  0.050817201 -0.1975722 3
## 5  0.1011329  1.1382172 -0.3353099  1.980538873 -1.4902878 4
## 6 -0.3842767  1.7629568 -0.1734520  0.009448173  0.4166688 5
## 7 -0.1897151 -0.2928122  0.9917801  0.147767309 -0.3447306 6
## 8 -1.5184068 -0.6339424 -1.4102368  0.471592965  1.0748895 7
## 9 -0.6475764  0.3884220  1.5151532 -1.977356193 -0.9561620 8
## 10 0.1476949 -0.2219758  0.6255156 -0.755406330 -0.3411347 9
## 11 1.1927071 -0.2031697  0.6926743  1.263878207 -0.2628487 10
## 12 0.6117842 -0.3206093 -1.0544746  0.074048308 -0.3483535 11
## 13 1.7865743 -0.9457715 -0.2907310  1.520606318  2.3182403 12
## 15 0.2087459  1.2049360  1.2614003  1.102632278  0.4413631 14
## 19 1.1240451 -0.1887518 -0.6514363 -0.988661412 -1.2906608 18

z.sub1 <- z.df[z.df$y == 1, ]
z.sub1

##          V1          V2          V3          V4          V5 y
## 1 -0.5202336  0.5695642 -0.810491 -0.77549235 1.831054 1
## 2 -1.4370163 -3.0437691 -0.489597 -0.01896309 2.298045 1

z.sub2 <- z.df[z.df$y %in% c(1, 4), ]
z.sub2

##          V1          V2          V3          V4          V5 y
## 1 -0.5202336  0.5695642 -0.8104910 -0.77549235 1.831054 1
## 2 -1.4370163 -3.0437691 -0.4895970 -0.01896309 2.298045 1
## 5  0.1011329  1.1382172 -0.3353099  1.98053887 -1.490288 4

# subsetting columns
z.sub6 <- z.df[, 1:2]
z.sub6

##          V1          V2
## 1 -0.5202336  0.5695642
## 2 -1.4370163 -3.0437691
## 3  1.1510882 -1.5345341
## 4  0.2937683 -1.1738992
## 5  0.1011329  1.1382172
## 6 -0.3842767  1.7629568
## 7 -0.1897151 -0.2928122
## 8 -1.5184068 -0.6339424
## 9 -0.6475764  0.3884220
## 10 0.1476949 -0.2219758
## 11 1.1927071 -0.2031697
## 12 0.6117842 -0.3206093
## 13 1.7865743 -0.9457715
## 14 -0.2075467  0.6440087
## 15 0.2087459  1.2049360
## 16 -0.8663415 -0.4149625
## 17 -0.4808447  0.6163081
## 18 -0.3456697  2.5622196
## 19 1.1240451 -0.1887518
## 20 -0.9783920  1.0246003

```

2.17 Low-Level Plotting Commands

points(x, y) adds points (the option `type = "p"` can be used).

lines(x, y) a line plot of (x,y) pairs.

text(x, y, labels, ...) adds text given by labels at coordinates (x, y). Typical use: `plot(x, y, type = "n"); text(x, y, names)`.

mtext(text, side = 3, line = 0, ...) adds text given by text in the margin specified by side (see `axis()` below); line specifies the line from the plotting area.

segments(x0, y0, x1, y1) draws lines from points (x0, y0) to points (x1, y1).

arrows(x0, y0, x1, y1, angle = 30, code = 2) draw arrows between pairs of points. With arrows at points (x0, y0), if `code = 2`, or at point (x1, y1), if `code = 1`. Arrows are at both if `code = 3`. Angle controls the angle from the shaft of the arrow to the edge of the arrow head.

abline(a, b) draws a line of slope b and intercept a.

abline(h = y) draws a horizontal line at ordinate y.

abline(v = x) draws a vertical line at abscissa x.

abline(lm.obj) draws the regression line given by `lm.obj`. `Abline(h = 0, col = 2)` #color (col) is often used.

rect(x1, y1, x2, y2) draws a rectangle which left, right, bottom, and top limits are x1, x2, y1, and y2, respectively.

polygon(x, y) draws a polygon linking the points with coordinates given by x and y.

legend(x, y, legend) adds the legend at the point (x, y) with the symbols given by legend.

title() adds a plot title and optionally a subtitle.

axis(side, vect) adds an axis at the bottom (`side = 1`), on the left (`side = 2`), at the top (`side = 3`), or on the right (`side = 4`); `vect` (optional) gives the abscissa (or ordinates) where tick-marks are drawn.

rug(x) draws the data x on the x-axis as small vertical lines.

locator(n, type = "n", ...) returns the coordinates (x, y) after the user has clicked n times on the plot with the mouse; also draws symbols (`type = "p"`) or lines (`type = "l"`) with respect to optional graphic parameters (...); by default nothing is drawn (`type = "n"`).

2.18 Graphics Parameters

These can be set globally with `par(...)`. Many can be passed as parameters to plotting commands (Table 2.8).

adj controls text justification (`adj = 0` left-justified, `adj = 0.5` centered, `adj = 1` right-justified).

Table 2.8 Common plotting functions for displaying variable relationships subject to conditioning (trellis plots) available in the R lattice package

Expression	Explanation
<code>xyplot(y~x)</code>	Bivariate plots (with many functionalities).
<code>barchart(y~x)</code>	Histogram of the values of y with respect to those of x.
<code>dotplot(y~x)</code>	Cleveland dot plot (stacked plots line-by-line and column-by-column)
<code>densityplot(~x)</code>	Density functions plot
<code>histogram(~x)</code>	Histogram of the frequencies of x
<code>bwplot(y~x)</code>	“Box-and-whiskers” plot
<code>qqmath(~x)</code>	Quantiles of x with respect to the values expected under a theoretical distribution
<code>stripplot(y~x)</code>	Single dimension plot, x must be numeric, y may be a factor
<code>qq(y~x)</code>	Quantiles to compare two distributions, x must be numeric, y may be numeric, character, or factor but must have two “levels”
<code>splom(~x)</code>	Matrix of bivariate plots
<code>parallel(~x)</code>	Parallel coordinates plot
<code>Levelplot (z ~ x * y g1 * g2)</code>	Colored plot of the values of z at the coordinates given by x and y (x, y and z are all of the same length)
<code>wireframe (z ~ x * y g1 * g2)</code>	3d surface plot
<code>cloud (z ~ x * y g1 * g2)</code>	3d scatter plot

bg specifies the color of the background (ex.: `bg = "red"`, `bg = "blue"`, ...the list of the 657 available colors is displayed with `colors()`).

bty controls the type of box drawn around the plot. Allowed values are: "o", "l", "7", "c", "u" ou "]" (the box looks like the corresponding character). If `bty = "n"` the box is not drawn.

cex a value controlling the size of texts and symbols with respect to the default. The following parameters have the same control for numbers on the axes-`cex.axis`, the axis labels-`cex.lab`, the title-`cex.main`, and the subtitle-`cex.sub`.

col. controls the color of symbols and lines. Use color names: "red", "blue" see `colors()` or as "#RRGGBB"; see `rgb()`, `hsv()`, `gray()`, and `rainbow()`; as for cex there are: `col.axis`, `col.lab`, `col.main`, `col.sub`.

font an integer which controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics); as for cex there are: `font.axis`, `font.lab`, `font.main`, `font.sub`.

las an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical).

lty controls the type of lines, can be an integer or string (1: "solid", 2: "dashed", 3: "dotted", 4: "dotdash", 5: "longdash", 6: "twodash", or a string of up to eight characters (between "0" and "9") which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty = "44"` will have the same effect than `lty = 2`.

lwd a numeric which controls the width of lines, default = 1.

mar a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`.

mfcol a vector of the form `c(nr, nc)` which partitions the graphic window as a matrix of nr lines and nc columns, the plots are then drawn in columns.

mfrow id. but the plots are drawn by row.

pch controls the type of symbol, either an integer between 1 and 25, or any single character within "".

ts.plot(x) id. but if x is multivariate the series may have different dates by x and y.

ps an integer which controls the size in points of texts and symbols.

pty a character, which specifies the type of the plotting region, "s": square, "m": maximal.

tck a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tck = 1` a grid is drawn.

tcl a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl = -0.5`).

xaxt if `xaxt = "n"` the x-axis is set but not drawn (useful in conjunction with `axis(side = 1, ...)`).

yaxt if `yaxt = "n"` the y-axis is set but not drawn (useful in conjunction with `axis(side = 2, ...)`).

Lattice (Trellis) graphics.

In the normal Lattice formula, `y~x | g1*g2` has combinations of optional conditioning variables `g1` and `g2` plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also `data` = the data frame for the formula variables and `subset` = for subsetting. Use `panel` = to define a custom panel function (see `apropos("panel")` and `?lines`). Lattice functions return an object of class `trellis` and have to be printed to produce the graph. Use `print(xyplot(...))` inside functions where automatic printing doesn't work. Use `lattice.theme` and `lset` to change Lattice defaults.

2.19 Optimization and model Fitting

optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN")) general-purpose optimization; `par` is initial values, `fn` is function to optimize (normally minimize).

nlm(f, p) minimize function fusing a Newton-type algorithm with starting values `p`.

lm(formula) fit linear models; `formula` is typically of the form `response ~ termA + termB + ...`; use `I(x*y) + I(x^2)` for terms made of nonlinear components.

glm(formula, family=) fits generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error

distribution; `family` is a description of the error distribution and link function to be used in the model; see `?family`.

`nls(formula)` nonlinear least-squares estimates of the nonlinear model parameters.

`approx(x, y=)` linearly interpolate given data points; `x` can be an `xy` plotting structure.

`spline(x, y=)` cubic spline interpolation.

`loess(formula)` (locally weighted scatterplot smoothing) fit a polynomial surface using local fitting.

Many of the formula-based modeling functions have several common arguments:

`data` = the data frame for the formula variables, `subset` = a subset of variables used in the fit, `na.action` = action for missing values: `"na.fail"`, `"na.omit"`, or a function.

The following generics often apply to model fitting functions:

`predict(fit, ...)` predictions from fit based on input data.

`df.residual(fit)` returns the number of residual degrees of freedom.

`coef(fit)` returns the estimated coefficients (sometimes with their standard errors).

`residuals(fit)` returns the residuals.

`deviance(fit)` returns the deviance.

`fitted(fit)` returns the fitted values.

`logLik(fit)` computes the logarithm of the likelihood and the number of parameters.

`AIC(fit)` computes the Akaike information criterion (AIC).

2.20 Statistics

There are many R packages and functions for computing a wide spectrum of statistics. Below are some commonly used examples, and we will see many more throughout:

`aov(formula)` analysis of variance model.

`anova(fit, ...)` analysis of variance (or deviance) tables for one or more fitted model objects.

`density(x)` kernel density estimates of `x`.

Other functions include: `binom.test()`, `pairwise.t.test()`, `power.t.test()`, `prop.test()`, `t.test()`, ... use `help.search("test")` to see details.

2.21 Distributions

It's easy to generate random samples from different distributions. Remember to include `set.seed()` if you want to get reproducibility during exercises (Table 2.9).

Also, all of these functions can be used by replacing the letter `r` with `d`, `p` or `q` to get, respectively, the probability density (`dfunc(x, ...)`), the cumulative probability density (`pfunc(x, ...)`), and the value of quantile (`qfunc(p, ...)`, with $0 < p < 1$).

2.21.1 Programming

The standard setting for defining **new functions** is:

```
function.name<-function(x) { expr(an expression) return(value) },
```

where x is the parameter in the expression. A simple example of this is:

```
adding<-function(x=0, y=0){z<-x+y
return(z)}
adding(x=5, y=10)
## [1] 15
```

Table 2.9 Examples of R random number generators

Expression	Explanation
<code>rnorm(n, mean = 0, sd = 1)</code>	Gaussian (normal)
<code>rexp(n, rate = 1)</code>	Exponential
<code>rgamma(n, shape, scale = 1)</code>	Gamma
<code>rpois(n, lambda)</code>	Poisson
<code>rweibull(n, shape, scale = 1)</code>	Weibull
<code>rcauchy(n, location = 0, scale = 1)</code>	Cauchy
<code>rbeta(n, shape1, shape2)</code>	Beta
<code>rt(n, df)</code>	Student's (t)
<code>rf(n, df1, df2)</code>	Fisher's (F) (df1, df2)
<code>rchisq(n, df)</code>	Pearson rbinom(n, size, prob) binomial
<code>rgeom(n, prob)</code>	Geometric
<code>rhyper(nn, m, n, k)</code>	Hypergeometric
<code>rlogis(n, location = 0, scale = 1)</code>	Logistic
<code>rlnorm(n, meanlog = 0, sdlog = 1)</code>	Lognormal
<code>rnbinom(n, size, prob)</code>	Negative binomial
<code>runif(n, min = 0, max = 1)</code>	Uniform
<code>rwilcox(nn, m, n), rsignrank(nn, n)</code>	Wilcoxon's statistics

Conditions setting

```
if(cond) {expr}
```

or

```
if(cond) cons.expr else alt.expr
x<-10
if(x>10) z="T" else z="F"
z
## [1] "F"
```

Alternatively, `ifelse` represents a vectorized and extremely efficient conditional mechanism that provides one of the main advantages of R.

For loop

```
for(var in seq) expr
x<-c()
for(i in 1:10) x[i]=i
x
## [1] 1 2 3 4 5 6 7 8 9 10
```

Other loops

While loop: `while(cond) expr`.

Repeat: `repeat expr`.

Applied to innermost of nested loops: `break`, `next`.

Use braces `{ }` around statements.

`ifelse(test, yes, no)` a value with the same shape as test filled with elements from either yes or no.

`do.call(funname, args)` executes a function call from the name of the function and a list of arguments to be passed to it.

2.22 Data Simulation Primer

Before we demonstrate how to synthetically simulate data that closely resemble the characteristics of real observations from the same process. Start by importing some observed data for initial exploratory analytics.

Using the SOCR Health Evaluation and Linkage to Primary (HELP) Care Dataset we can extract some sample data: `00_Tiny_SOCR_HELP_Data_Simulation.csv` (Table 2.10, Figs. 2.8 and 2.9).

Table 2.10 A fragment of the SOCR Health Evaluation and Linkage to Primary (HELP) Care dataset

ID	i2	age	treat	homeless	pcs	mcs	cesd	...	female	Substance	racegrp
1	0	25	0	0	49	7	46	...	0	Cocaine	Black 2
3	39	36	0	0	76	9	33	...	0	Heroin	Black
100	81	22	0	0	37	17	19	...	0	Alcohol	Other

Fig. 2.8 Histogram of a sample of 200 random $\text{Normal}(m = 10, \text{sd} = 20)$ observations

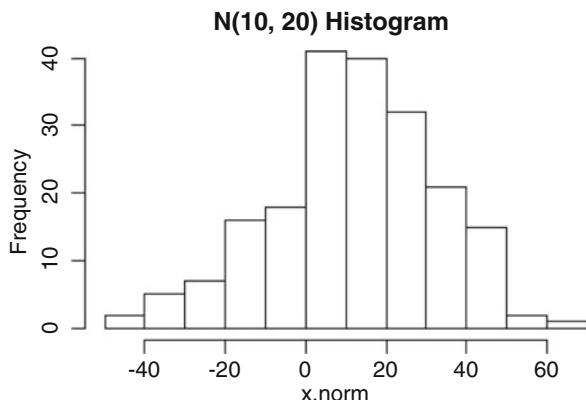
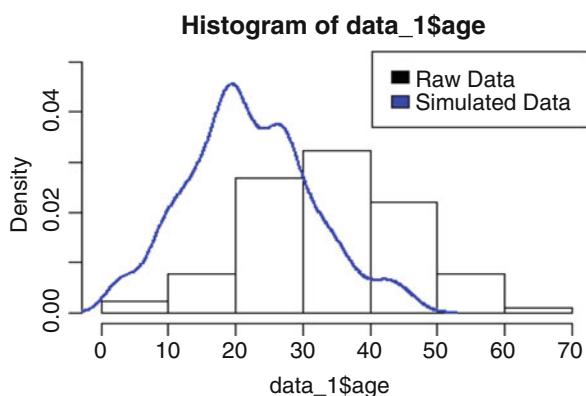


Fig. 2.9 Comparing the histogram (black) of the real ages of the PD patients to the synthetically generated/simulated ages (blue)



```

data_1 <- read.csv("https://umich.instructure.com/files/1628625/downLoad?downLoad_frd=1", as.is=T, header=T)
# data_1 = read.csv(file.choose( ))

attach(data_1)
# to ensure all variables are accessible within R, e.g., using "age" instead of data_1$age

```

```
# i2 maximum number of drinks (standard units) consumed per day (in the past 30 days range 0-184) see also i1
# treat randomization group (0=usual care, 1=HELP clinic)
# pcs SF-36 Physical Component Score (range 14-75)
# mcs SF-36 Mental Component Score(range 7-62)
# cesd Center for Epidemiologic Studies Depression scale (range 0-60)
# indtot Inventory of Drug Use Consequences (InDUC) total score (range 4-45)
# pss_fr perceived social supports (friends, range 0-14) see also dayslink
# drugrisk Risk-Assessment Battery(RAB) drug risk score (range0-21)
# satreat any BSAS substance abuse treatment at baseline (0=no, 1=yes)
```

```
summary(data_1)

##           ID          i2          age          treat
##  Min.   : 1.00   Min.   : 0.00   Min.   : 3.00   Min.   :0.0000
##  1st Qu.: 24.25  1st Qu.: 1.00   1st Qu.:27.00  1st Qu.:0.0000
##  Median : 50.50  Median : 15.50  Median :34.00  Median :0.0000
##  Mean   : 50.29  Mean   : 27.08  Mean   :34.31  Mean   :0.1222
##  3rd Qu.: 74.75  3rd Qu.: 39.00  3rd Qu.:43.00  3rd Qu.:0.0000
##  Max.   :100.00  Max.   :137.00  Max.   :65.00  Max.   :2.0000
##          homeless         pcs          mcs          cesd
##  Min.   :0.0000  Min.   : 6.00  Min.   : 0.00  Min.   : 0.00
##  1st Qu.:0.0000  1st Qu.:41.25  1st Qu.:20.25  1st Qu.:17.25
##  Median :0.0000  Median :48.50  Median :29.00  Median :30.00
##  Mean   :0.1444  Mean   :47.61  Mean   :30.49  Mean   :30.21
##  3rd Qu.:0.0000  3rd Qu.:57.00  3rd Qu.:39.75  3rd Qu.:43.00
##  Max.   :1.0000  Max.   :76.00  Max.   :93.00  Max.   :68.00
##          indtot         pss_fr        drugrisk        sexrisk
##  Min.   : 0.00  Min.   : 0.000  Min.   : 0.000  Min.   : 0.000
##  1st Qu.:31.25  1st Qu.: 2.000  1st Qu.: 0.000  1st Qu.: 1.250
##  Median :36.00  Median : 6.000  Median : 0.000  Median : 5.000
##  Mean   :37.03  Mean   : 6.533  Mean   : 2.578  Mean   : 4.922
##  3rd Qu.:45.00  3rd Qu.:10.000  3rd Qu.: 3.000  3rd Qu.: 7.750
##  Max.   :60.00  Max.   :20.000  Max.   :23.000  Max.   :13.000
##          satreat        female        substance        racegrp
##  Min.   :0.00000  Min.   :0.00000  Length:90      Length:90
##  1st Qu.:0.00000  1st Qu.:0.00000  Class :character  Class :character
##  Median :0.00000  Median :0.00000  Mode   :character  Mode   :character
##  Mean   :0.07778  Mean   :0.05556
##  3rd Qu.:0.00000  3rd Qu.:0.00000
##  Max.   :1.00000  Max.   :1.00000

x.norm <- rnorm(n=200, m=10, sd=20)
hist(x.norm, main='N(10, 20) Histogram')
```

```
mean(data_1$age)
## [1] 34.31111
sd(data_1$age)
## [1] 11.68947
```

Next, we will simulate new synthetic data to match the properties/characteristics of the observed data (using Uniform, Normal, and Poisson distributions):

```

# i2 [0: 184]
# age m=34, sd=12
# treat {0, 1}
# homeless {0, 1}
# pcs 14-75
# mcs 7-62
# cesd 0-60
# indtot 4-45
# pss_fr 0-14
# drugrisk 0-21
# sexrisk
# satreat (0=no, 1=yes)
# female (0=no, 1=yes)
# racegrp (black, white, other)

# Demographics variables
# Define number of subjects
NumSubj <- 282
NumTime <- 4

# Define data elements
# Cases
Cases <- c(2, 3, 6, 7, 8, 10, 11, 12, 13, 14, 17, 18, 20, 21, 22, 23, 24,
25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 37, 41, 42, 43, 44, 45, 53, 55, 58,
60, 62, 67, 69, 71, 72, 74, 79, 80, 85, 87, 90, 95, 97, 99, 100, 101, 106,
107, 109, 112, 120, 123, 125, 128, 129, 132, 134, 136, 139, 142, 147, 149,
153, 158, 160, 162, 163, 167, 172, 174, 178, 179, 180, 182, 192, 195, 201,
208, 211, 215, 217, 223, 227, 228, 233, 235, 236, 240, 245, 248, 250, 251,
254, 257, 259, 261, 264, 268, 269, 272, 273, 275, 279, 288, 289, 291, 296,
298, 303, 305, 309, 314, 318, 324, 325, 326, 328, 331, 332, 333, 334, 336,
338, 339, 341, 344, 346, 347, 350, 353, 354, 359, 361, 363, 364, 366, 367,
368, 369, 370, 371, 372, 374, 375, 376, 377, 378, 381, 382, 384, 385, 386,
387, 389, 390, 393, 395, 398, 400, 410, 421, 423, 428, 433, 435, 443, 447,
449, 450, 451, 453, 454, 455, 456, 457, 458, 459, 460, 461, 465, 466, 467,
470, 471, 472, 476, 477, 478, 479, 480, 481, 483, 484, 485, 486, 487, 488,
489, 492, 493, 494, 496, 498, 501, 504, 507, 510, 513, 515, 528, 530, 533,
537, 538, 542, 545, 546, 549, 555, 557, 559, 560, 566, 572, 573, 576, 582,
586, 590, 592, 597, 603, 604, 611, 619, 621, 623, 624, 625, 631, 633, 634,
635, 637, 640, 641, 643, 644, 645, 646, 647, 648, 649, 650, 652, 654, 656,
658, 660, 664, 665, 670, 673, 677, 678, 679, 680, 682, 683, 686, 687, 688,
689, 690, 692)

# Imaging Biomarkers
L_caudate_ComputeArea <- rpois(NumSubj, 600)
L_caudate_Volume <- rpois(NumSubj, 800)
R_caudate_ComputeArea <- rpois(NumSubj, 893)
R_caudate_Volume <- rpois(NumSubj, 1000)
L_putamen_ComputeArea <- rpois(NumSubj, 900)
L_putamen_Volume <- rpois(NumSubj, 1400)
R_putamen_ComputeArea <- rpois(NumSubj, 1300)
R_putamen_Volume <- rpois(NumSubj, 3000)
L_hippocampus_ComputeArea <- rpois(NumSubj, 1300)
L_hippocampus_Volume <- rpois(NumSubj, 3200)
R_hippocampus_ComputeArea <- rpois(NumSubj, 1500)
R_hippocampus_Volume <- rpois(NumSubj, 3800)
cerebellum_ComputeArea <- rpois(NumSubj, 16700)
cerebellum_Volume <- rpois(NumSubj, 14000)
L_lingual_gyrus_ComputeArea <- rpois(NumSubj, 3300)
L_lingual_gyrus_Volume <- rpois(NumSubj, 11000)

```

```

R_Lingual_gyrus_ComputeArea <- rpois(NumSubj, 3300)
R_Lingual_gyrus_Volume <- rpois(NumSubj, 12000)
L_fusiform_gyrus_ComputeArea <- rpois(NumSubj, 3600)
L_fusiform_gyrus_Volume <- rpois(NumSubj, 11000)
R_fusiform_gyrus_ComputeArea <- rpois(NumSubj, 3300)
R_fusiform_gyrus_Volume <- rpois(NumSubj, 10000)

Sex <- ifelse(runif(NumSubj)<.5, 0, 1)

Weight <- as.integer(rnorm(NumSubj, 80, 10))

Age <- as.integer(rnorm(NumSubj, 62, 10))

# Diagnostic labels (DX):
Dx <- c(rep("PD", 100), rep("HC", 100), rep("SWEDD", 82))

# Genetics traits
chr12_rs34637584_GT <- c(ifelse(runif(100)<.3, 0, 1), ifelse(runif(100)<.6, 0, 1), ifelse(runif(82)<.4, 0, 1)) # NumSubj Bernoulli trials

chr17_rs11868035_GT <- c(ifelse(runif(100)<.7, 0, 1), ifelse(runif(100)<.4, 0, 1), ifelse(runif(82)<.5, 0, 1)) # NumSubj Bernoulli trials

# Clinical
# rpois(NumSubj, 15) + rpois(NumSubj, 6)
UPDRS_part_I <- c(ifelse(runif(100)<.7, 0, 1) + ifelse(runif(100) < .7, 0, 1), ifelse(runif(100)<.6, 0, 1) + ifelse(runif(100)<.6, 0, 1), ifelse(runif(82)<.4, 0, 1) + ifelse(runif(82)<.4, 0, 1) )

UPDRS_part_II <- c(sample.int(20, 100, replace=T), sample.int(14, 100, replace=T),
sample.int(18, 82, replace=T) )

UPDRS_part_III <- c(sample.int(30, 100, replace=T), sample.int(20, 100, replace=T), sample.int(25, 82, replace=T) )

# Time: VisitTime - done automatically below in aggregator

# Data (putting all components together)
sim_PD_Data <- cbind(
  rep(Cases, each= NumTime), # Cases
  rep(L_caudate_ComputeArea, each= NumTime), # Imaging
  rep(Sex, each= NumTime), # Demographics
  rep(Weight, each= NumTime),
  rep(Age, each= NumTime),
  rep(Dx, each= NumTime), # Dx
  rep(chr12_rs34637584_GT, each= NumTime), # Genetics
  rep(chr17_rs11868035_GT, each= NumTime),
  rep(UPDRS_part_I, each= NumTime), # Clinical
  rep(UPDRS_part_II, each= NumTime),
  rep(UPDRS_part_III, each= NumTime),
  rep(c(0, 6, 12, 18), NumSubj) # Time
)
# Assign the column names
colnames(sim_PD_Data) <- c(
  "Cases",
  "L_caudate_Volume",
  "R_caudate_Volume",
  "L_fusiform_Volume",
  "R_fusiform_Volume",
  "Sex",
  "Weight",
  "Age",
  "Dx",
  "chr12_rs34637584_GT",
  "chr17_rs11868035_GT",
  "UPDRS_part_I",
  "UPDRS_part_II",
  "UPDRS_part_III",
  "VisitTime"
)

```

```

  "L_caudate_ComputeArea",
  "Sex", "Weight", "Age",
  "Dx", "chr12_rs34637584_GT", "chr17_rs11868035_GT",
  "UPDRS_part_I", "UPDRS_part_II", "UPDRS_part_III", "Time"
)
# some QC
summary(sim_PD_Data)

##      Cases      L_caudate_ComputeArea Sex      Weight      Age
## 10      : 4      594      : 36      0:592      77      : 72      59      : 68
## 100     : 4      607      : 36      1:536      83      : 60      58      : 56
## 101     : 4      618      : 32      76      : 56      61      : 56
## 106     : 4      581      : 28      78      : 56      60      : 52
## 107     : 4      585      : 28      70      : 44      67      : 52
## 109     : 4      599      : 28      75      : 44      65      : 48
## (Other):1104 (Other):940      (Other):796 (Other):796
##      Dx      chr12_rs34637584_GT chr17_rs11868035_GT UPDRS_part_I
##  HC   :400      0:464              0:592              0:412
##  PD   :400      1:664              1:536              1:520
##  SWEDD:328
##      UPDRS_part_II UPDRS_part_III Time
## 13      :108      1      : 80      0 :282
## 9       :100      13     : 68      12:282
## 5       : 88      19     : 60      18:282
## 14      : 76      7      : 60      6 :282
## 6       : 76      12     : 56
## 3       : 72      6      : 56
## (Other):608 (Other):748

dim(sim_PD_Data)
## [1] 1128 12

head(sim_PD_Data)

##      Cases L_caudate_ComputeArea Sex Weight Age Dx chr12_rs34637584_GT
## [1,] "2"   "618"               "0"  "59" "PD" "1"
## [2,] "2"   "618"               "0"  "59" "PD" "1"
## [3,] "2"   "618"               "0"  "59" "PD" "1"
## [4,] "2"   "618"               "0"  "59" "PD" "1"
## [5,] "3"   "621"               "0"  "61" "PD" "1"
## [6,] "3"   "621"               "0"  "61" "PD" "1"
##      chr17_rs11868035_GT UPDRS_part_I UPDRS_part_II UPDRS_part_III Time
## [1,] "0"      "1"              "2"      "4"      "0"
## [2,] "0"      "1"              "2"      "4"      "6"
## [3,] "0"      "1"              "2"      "4"      "12"
## [4,] "0"      "1"              "2"      "4"      "18"
## [5,] "1"      "1"              "13"     "6"      "0"
## [6,] "1"      "1"              "13"     "6"      "6"

hist(data_1$age, freq=FALSE, right=FALSE, ylim = c(0,0.05))
lines(density(as.numeric(as.data.frame(sim_PD_Data$Age))), lwd=2, col="blue")
Legend("topright", c("Raw Data", "Simulated Data"), fill=c("black", "blue"))

```

```
# Save Results
# Write out (save) the result to a file that can be shared
write.table(sim_PD_Data, "output_data.csv", sep=", ", row.names=FALSE, col.names=TRUE)
```

2.23 Appendix

2.23.1 HTML SOCR Data Import

SOCR Datasets can automatically be downloaded into the R environment using the following protocol, which uses the Parkinson's Disease dataset as an example:

```
library(rvest)

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_Bio
medBigMetadata")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...
pd_data <- html_table(html_nodes(wiki_url, "table"))[[1]]
head(pd_data); summary(pd_data)
##   Cases L_caudate_ComputeArea L_caudate_Volume R_caudate_ComputeArea
## 1     2                 597                 767                 855
## 2     2                 597                 767                 855
## 3     2                 597                 767                 855
## 4     2                 597                 767                 855
## 5     3                 604                 873                 935
## 6     3                 604                 873                 935
...
##   chr17_rs11868035_GT UPDRS_part_I UPDRS_part_II UPDRS_part_III Time
## 1                 0             1             12             1     0
## 2                 0             1             12             1     6
## 3                 0             1             12             1    12
## 4                 0             1             12             1    18
## 5                 1             0             19            22     0
## 6                 1             0             19            22     6
...
##   Cases      L_caudate_ComputeArea L_caudate_Volume
##  Min.   : 2.0  Min.   :525.0      Min.   :719.0
##  1st Qu.:158.0 1st Qu.:582.0      1st Qu.:784.0
##  Median :363.5 Median :600.0      Median :800.3
##  Mean   :346.1 Mean   :600.4      Mean   :800.3
##  3rd Qu.:504.0 3rd Qu.:619.0      3rd Qu.:819.0
##  Max.   :692.0  Max.   :667.0      Max.   :890.0
...
```

Also see: http://wiki.socr.umich.edu/index.php/SMHS_DataSimulation.

2.23.2 R Debugging

Most programs that give incorrect results are impacted by logical errors. When errors (bugs, exceptions) occur, we need explore deeper – this procedure to identify and fix bugs is “debugging”.

Common R tools for debugging include `traceback()`, `debug()`, `browser()`, `trace()` and `recover()`.

traceback(): Failing R functions report the error to the screen immediately the error. Calling `traceback()` will show the function where the error occurred. The `traceback()` function prints the list of functions that were called before the error occurred.

The function calls are printed in reverse order.

```
f1<-function(x) { r<- x-g1(x); r }
g1<-function(y) { r<-y*h1(y); r }
h1<-function(z) { r<-log(z); if(r<10) r^2 else   r^3}
f1(-1)
## Warning in log(z): NaNs produced
## Error in if (r < 10) r^2 else r^3: missing value where TRUE/FALSE needed
traceback()
3: h(y)
## Error in eval(expr, envir, enclos): could not find function "h"
2: g(x)
## Error in eval(expr, envir, enclos): could not find function "g"
1: f(-1)
## Error in eval(expr, envir, enclos): could not find function "f"
```

debug()

`traceback()` does not tell you where is the error. To find out which line causes the error, we may step through the function using `debug()`.

`debug(foo)` flags the function `foo()` for debugging. `Undebug(foo)` unflags the function.

When a function is flagged for debugging, each statement in the function is executed one at a time. After a statement is executed, the function suspends and user can interact with the R shell.

This allows us to inspect a function line-by-line.

An example computing the sum of squared errors, `SS`.

```

## compute sum of squares
SS<-function(mu, x) {
  d<-x-mu;
  d2<-d^2;
  ss<-sum(d2);
  ss }
set.seed(100);
x<-rnorm(100);
SS(1, x)

## to debug
debug(SS); SS(1, x)

## debugging in: SS(1, x)
## debug at <text>#2: {
##   d <- x - mu
##   d2 <- d^2
##   ss <- sum(d2)
##   ss
## }
## debug at <text>#3: d <- x - mu
## debug at <text>#4: d2 <- d^2
## debug at <text>#5: ss <- sum(d2)
## debug at <text>#6: ss
## exiting from: SS(1, x)

## [1] 202.5614519

```

In the debugging shell ("Browse[1] > "), users can:

- Enter **n** (next) executes the current line and prints the next one;
- Typing **c** (continue) executes the rest of the function without stopping;
- Enter **Q** quits the debugging;
- Enter **ls()** list all objects in the local environment;
- Enter an object name or **print()** tells the current value of an object.

Example:

```

debug(SS)
SS(1, x)

## debugging in: SS(1, x)
## debug at <text>#2: {
##   d <- x - mu
##   d2 <- d^2
##   ss <- sum(d2)
##   ss
## }
## debug at <text>#3: d <- x - mu
## debug at <text>#4: d2 <- d^2
## debug at <text>#5: ss <- sum(d2)
## debug at <text>#6: ss
## exiting from: SS(1, x)

## [1] 202.5614519

```

```

Browse[1]> n
debug: d <- x - mu ## the next command
Browse[1]> ls() ## current environment [1] "mu" "x" ## there is no d
Browse[1]> n ## go one step debug: d2 <- d^2 ## the next command
Browse[1]> ls() ## current environment [1] "d" "mu" "x" ## d has been created
Browse[1]> d[1:3] ## first three elements of d [1] -1.5021924 -0.8684688 -1.0789171
Browse[1]> hist(d) ## histogram of d
Browse[1]> where ## current position in call stack where 1: SS(1, x)
Browse[1]> n
debug: ss <- sum(d2)
Browse[1]> Q ## quit

undebug(ss)          ## remove debug Label, stop debugging process
ss(1, x)                ## now call SS again will without debugging

```

You can label a function for debugging while debugging another function.

```

f<-function(x)
{ r<-x-g(x);
  r }
g<-function(y)
{ r<-y*h(y);
  r }
h<-function(z)
{ r<-Log(z);
  if(r<10)
    r^2
  else
    r^3 }

debug(f)          ## ## If you only debug f, you will not go into g
f(-1)

## Warning in Log(z): Nans produced
## Error in if (r < 10) r^2 else r^3: missing value where TRUE/FALSE needed

Browse[1]> n
Browse[1]> n

```

But, we can also label g and h for debugging when we debug f.

```

f(-1)
Browse[1]> n
Browse[1]> debug(g)
Browse[1]> debug(h)
Browse[1]> n

```

Inserting a call to **browser()** in a function will pause the execution of a function at the point where **browser()** is called. This is similar to using **debug()**, except you can control where execution gets paused.

Example

```

h<-function(z) {
  browser()  ## a break point inserted here
  r<-log(z);
  if(r<10)
    r^2
  else
    r^3
}

f(-1)

## Error in if (r < 10) r^2 else r^3: missing value where TRUE/FALSE needed

Browse[1]> ls() Browse[1]> z
Browse[1]> n
Browse[1]> n
Browse[1]> ls()
Browse[1]> c

```

Calling **trace()** on a function allows inserting new code into a function. The syntax for trace() may be challenging.

```

as.list(body(h))
trace("h", quote(
  if(is.nan(r))
  {browser()})
  , at=3, print=FALSE)
f(1)
f(-1)
trace("h", quote(if(z<0) {z<-1}), at=2, print=FALSE)
f(-1)
untrace()

```

During the debugging process, **recover()** allows checking the status of variables in upper level functions. Recover() can be used as an error handler using **options()** (e.g. `options(error = recover)`). When functions throw exceptions, execution stops at point of failure. Browsing the function calls and examining the environment may indicate the source of the problem.

2.24 Assignments: 2. R Foundations

2.24.1 Confirm that You Have Installed R/RStudio

You should be able to download and load the Foundations of R code in RStudio and then run all the examples.

2.24.2 Long-to-Wide Data Format Translation

Load the SOCR Parkinson's Disease data in the long-format (http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_BiomedBigMetadata#Data_Table) and export it in the wide format. You can only do five variables you choose (not all), but note that there are several time observations for each subject. You can try using the `reshape` command.

2.24.3 Data Frames

Create a Data Frame of the SOCR Parkinson's Disease data and compute a summary of three features you select.

2.24.4 Data Stratification

Using the same SOCR Parkinson's Disease data:

- Extract the first 10 subjects
- Find the cases for which `L_caudate_ComputeArea < 600`
- Sort the subjects based on `L_caudate_Volume`
- Generate frequency and probability tables for `Gender` and `Age`
- Compute the mean `Age` and the correlation between `Age` and `Weight`
- Plot Histogram and density of `R_fusiform_gyrus_Volume` and scatterplot `L_fusiform_gyrus_Volume` and `R_fusiform_gyrus_Volume`.

Note: You don't have to apply these data filters sequentially, but this can also be done for deeper stratification.

2.24.5 Simulation

Generate 1,000 standard normal variables and 1,200 Cauchy distributed random variables and generate a quantile-quantile (Q-Q) probability plot of the two samples. Repeat this with 1,500 student t distributed random variables with `df = 20` and generate a quantile-quantile (Q-Q) probability plot.

2.24.6 *Programming*

Generate a function that computes the arithmetic average and compare it against the `mean()` function using the simulation data you generated in the last question.

References

Some R fundamentals: http://wiki.socr.umich.edu/index.php/SMHS_Usage_Rfundamentals
The Software Carpentry Foundation.
Programming with R: <http://swcarpentry.github.io/r-novice-inflammation>
R for Reproducible Scientific Analysis: <http://swcarpentry.github.io/r-novice-gapminder>
A very gentle stats intro using R Book (Verzani): <http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf>
Quick-R web examples: <http://www.statmethods.net/index.html>
R-tutor Introduction: <http://www.r-tutor.com/r-introduction>
R project Introduction: <http://cran.r-project.org/doc/manuals/r-release/R-intro.html>
UCLA ITS/IDRE R Resources: <https://stats.idre.ucla.edu/r/>

Chapter 3

Managing Data in R



In this Chapter, we will discuss strategies to `import` data and `export` results. Also, we are going to learn the basic tricks we need to know about processing different types of data. Specifically, we will illustrate common R data structures and strategies for loading (ingesting) and saving (regurgitating) data. In addition, we will (1) present some basic statistics, e.g., for measuring central tendency (mean, median, mode) or dispersion (variance, quartiles, range); (2) explore simple plots; (3) demonstrate the uniform and normal distributions; (4) contrast numerical and categorical types of variables; (5) present strategies for handling incomplete (missing) data; and (6) show the need for cohort-rebalancing when comparing imbalanced groups of subjects, cases or units.

3.1 Saving and Loading R Data Structures

Let's start by extracting the Edgar Anderson's Iris Data from the package `datasets`. The `iris` dataset quantifies morphologic shape variations of 50 Iris flowers of three related genera – *Iris setosa*, *Iris virginica* and *Iris versicolor*. Four shape features were measured from each sample – length and the width of the sepals and petals (in centimeters). These data were used by Ronald Fisher in his 1936 linear discriminant analysis paper (Fig. 3.1).

```
data()  
data(iris)  
class(iris)  
## [1] "data.frame"
```

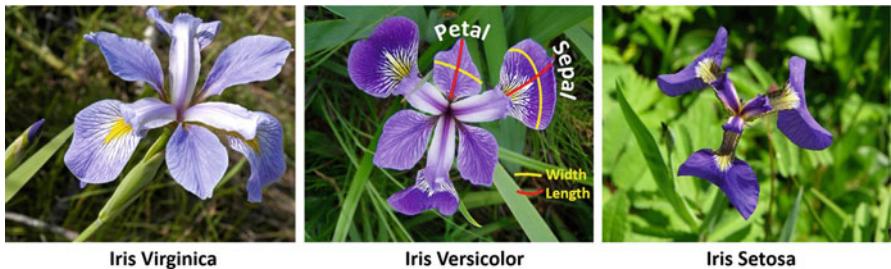


Fig. 3.1 Definitions of petal width and length for the three iris flower genera used in the example below

As an I/O (input/output) demonstration, after we load the `iris` data and examine its class type, we can save it into a file named "myData.RData" and then reload it back into R.

```
save(iris, file="myData.RData")
Load("myData.RData")
```

3.2 Importing and Saving Data from CSV Files

Importing the data from "CaseStudy07_WorldDrinkingWater_Data.csv" from these case-studies (https://umich.instructure.com/courses/38100/files/folder/Case_Studies) and saving it into the R dataset named "water" The variables in the dataset are as follows:

- **Time:** Years (1990, 1995, 2000, 2005, 2010, 2012)
- **Demographic:** Country (across the world)
- **Residence Area Type:** Urban, rural, or total
- **WHO Region**
- **Population using improved drinking water sources:** The percentage of the population using an improved drinking water source.
- **Population using improved sanitation facilities:** The percentage of the population using an improved sanitation facility.

Generally, the separator of a CSV file is comma. By default, we have option `sep = ", "` in the command `read.csv()`. Also, we can use `colnames()` to rename the column variables.

```

water <- read.csv('https://umich.instructure.com/files/399172/download?download_frd=1', header=T)
water[1:3, ]

##   Year..string. WHO.region..string. Country..string.
## 1      1990           Africa        Algeria
## 2      1990           Africa        Angola
## 3      1990           Africa        Benin
##   Residence.Area.Type..string.
## 1                  Rural
## 2                  Rural
## 3                  Rural
##   Population.using.improved.drinking.water.sources.....numeric.
## 1                               88
## 2                               42
## 3                               49
##   Population.using.improved.sanitation.facilities.....numeric.
## 1                               77
## 2                               7
## 3                               0

colnames(water)<-c("year", "region", "country", "residence_area", "improved_water", "sanitation_facilities")
water[1:3, ]

##   year region country residence_area improved_water sanitation_facilities
## 1 1990 Africa Algeria      Rural           88                  77
## 2 1990 Africa Angola      Rural           42                   7
## 3 1990 Africa Benin      Rural           49                   0

which.max(water$year);

## 913

# rowMeans(water[,5:6])
mean(water[,6], trim=0.08, na.rm=T)

## [1] 71.63629

```

This code loads CSV files that already include a header line listing the names of the variables. If we don't have a header in the dataset, we can use the `header = FALSE` option (https://umich.instructure.com/courses/38100/files/folder/Case_Studies). R will assign default names to the column variables of the dataset.

```

Simulation <- read.csv("https://umich.instructure.com/files/354289/download?download_frd=1", header = FALSE)
Simulation[1:3, ]

##   V1 V2 V3   V4      V5 V6 V7   V8      V9 V10      V11 V12
## 1 ID i2 age treat homeless pcs mcs cesd indtot pss_fr drugrisk sexrisk
## 2  1  0  25    0      0  49  7  46    37    0     1     6
## 3  2  18 31    0      0  48  34  17    48    0     0    11
##   V13    V14      V15    V16
## 1 satreat female substance racegrp
## 2      0      0 cocaine  black
## 3      0      0 alcohol white

```

To save a data frame to CSV files, we could use the `write.csv()` function. The option `file = "a/local/file/path"` allows us edit the saved file path.

```
write.csv(iris, file = "C:/Users/iris.csv") # Iris data
write.csv(water, file = "C:/Users/WHO_Water.csv") # World Drinking Water
```

3.3 Exploring the Structure of Data

We can use the command `str()` to explore the structure of a dataset. For instance, using the World Drinking Water dataset:

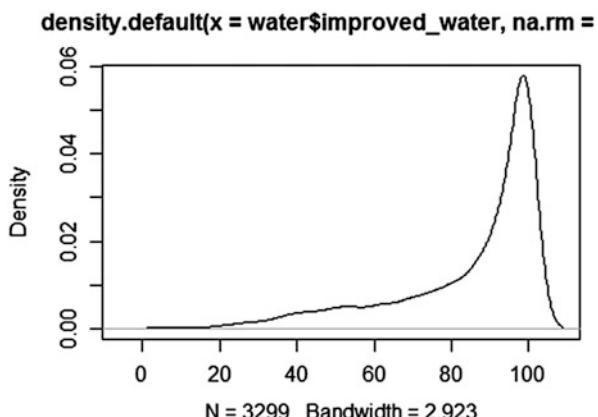
```
str(water)
## 'data.frame': 3331 obs. of 6 variables:
## $ year : int 1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
## $ region : Factor w/ 6 Levels "Africa","Americas",...: 1 1
## $ country : Factor w/ 192 Levels "Afghanistan",...: 3 5 19 2
## $ residence_area : Factor w/ 3 Levels "Rural","Total",...: 1 1 1 1
## $ improved_water : num 88 42 49 86 39 67 34 46 37 83 ...
## $ sanitation_facilities: num 77 7 0 22 2 42 27 12 4 11 ...
```

We can see that this (World Drinking Water) dataset has 3331 observations and 6 variables. The output also give us the class of each variable and first few elements in the variable.

3.4 Exploring Numeric Variables

Summary statistics for numeric variables in the dataset could be accessed by using the command `summary()` (Fig. 3.2).

Fig. 3.2 Density plot of the water improvement variable in the World Health Organization (WHO) water quality case-study.



```

summary(water$year)
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1990     1995    2005    2002    2010    2012

summary(water[c("improved_water", "sanitation_facilities")])
## improved_water sanitation_facilities
## Min. : 3.0 Min. : 0.00
## 1st Qu.: 77.0 1st Qu.: 42.00
## Median : 93.0 Median : 81.00
## Mean : 84.9 Mean : 68.87
## 3rd Qu.: 99.0 3rd Qu.: 97.00
## Max. :100.0 Max. :100.00
## NA's :32     NA's :135

plot(density(water$improved_water, na.rm = T))
# variables need not be continuous, we can still get intuition about their
distribution

```

The six summary statistics and NA's (missing data) are reported in the R output above.

3.5 Measuring the Central Tendency: Mean, Median, Mode

Mean and **median** are two frequent measurements of the central tendency. Mean is “the sum of all values divided by the number of values”. Median is the number in the middle of an ordered list of values. In R, `mean()` and `median()` functions provide us with these two measurements.

```

vec1<-c(40, 56, 99)
mean(vec1)
## [1] 65

mean(c(40, 56, 99))
## [1] 65

median(vec1)
## [1] 56

median(c(40, 56, 99))
## [1] 56

# install.packages("psych");
library("psych")
geometric.mean(vec1, na.rm=TRUE)
## [1] 60.52866

```

The **mode** is the value that occurs most often in the dataset. It is often used in categorical data, where mean and median are inappropriate measurements.

We can have one or more modes. In the water dataset, we have “Europe” and “Urban” as the modes for region and residence area, respectively. These two variables are unimodal, which has a single mode. For the year variable, we have two modes: 2000 and 2005. Both of the categories have 570 counts. The year variable is an example of a bimodal. We also have multimodal data that has two or more modes.

Mode is just one of the measures for the central tendency. The best way to use it is to compare the counts of the mode to other values. This help us to judge whether one or several categories dominates all others in the data. After that, we are able to analyze the meaning behind these common centrality measures.

In numeric datasets, the mode(s) represents the highest bin(s) in the histogram. In this way, we can also examine if the numeric data is multimodal.

More information about measures of centrality is available here (http://wiki.socr.umich.edu/index.php/AP_Statistics_Curriculum_2007_EDA_Center).

3.6 Measuring Spread: Quartiles and the Five-Number Summary

The five-number summary describes the spread of a dataset. They are:

- Minimum (Min.), representing the smallest value in the data
- First quantile/Q1 (1st Qu.), representing the 25th percentile, which splits off the lowest 25% of data from the highest 75%
- Median/Q2 (Median), representing the 50th percentile, which splits off the lowest 50% of data from the top 50%
- Third quantile/Q3 (3rd Qu.), representing the 75th percentile, which splits off the lowest 75% of data from the top 25%
- Maximum (Max.), representing the largest value in the data.

Min and Max can be obtained by using `min()` and `max()` respectively.

The difference between maximum and minimum is known as range. In R, the `range()` function gives us both the minimum and maximum. A combination of `range()` and `diff()` could do the trick of getting the actual range value.

```
range(water$year)
## [1] 1990 2012
diff(range(water$year))
## [1] 22
```

Q1 and Q3 are the 25th and 75th percentiles of the data. Median (Q2) is right in the middle of Q1 and Q3. The difference between Q3 and Q1 is called the

interquartile range (IQR). Within the IQR lies half of our data that has no extreme values.

In R, we use the `IQR()` to calculate the interquartile range. If we use `IQR()` for a data with NA's, the NA's are ignored by the function while using the option `na.rm = TRUE`.

```
IQR(water$year)
## [1] 15
summary(water$improved_water)
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.   NA's
##   3.0   77.0  93.0   84.9  99.0  100.0    32
IQR(water$improved_water, na.rm = T)
## [1] 22
```

Similar to the command `summary()` that we talked about earlier in this Chapter, the function `quantile()` could be used to obtain the five-number summary.

```
quantile(water$improved_water, na.rm = T)
##   0% 25% 50% 75% 100%
##   3   77  93   99  100
```

We can also calculate specific percentiles in the data. For example, if we want the 20th and 60th percentiles, we can do the following.

```
quantile(water$improved_water, probs = c(0.2, 0.6), na.rm = T)
## 20% 60%
## 71  97
```

Using the `seq()` function, we can generate percentiles that are evenly-spaced.

```
quantile(water$improved_water, seq(from=0, to=1, by=0.2), na.rm = T)
##   0% 20% 40% 60% 80% 100%
##   3   71  89   97  100  100
```

Let's reexamine the five-number summary for the `improved_water` variable. When we ignore the NA's, the difference between minimum and Q1 is 74 while the difference between Q3 and maximum is only 1. The interquartile range is 22%. Combining these facts, the first quarter is more widely spread than the middle 50% of values. The last quarter is the most condensed one that has only two percentages: 99% and 100%. Also, we can notice that the mean is smaller than the median. The mean is more sensitive to the extreme values than the median. We have a very small minimum that makes the range of first quartile very large. This extreme value impacts the mean more than the median.

3.7 Visualizing Numeric Variables: Boxplots

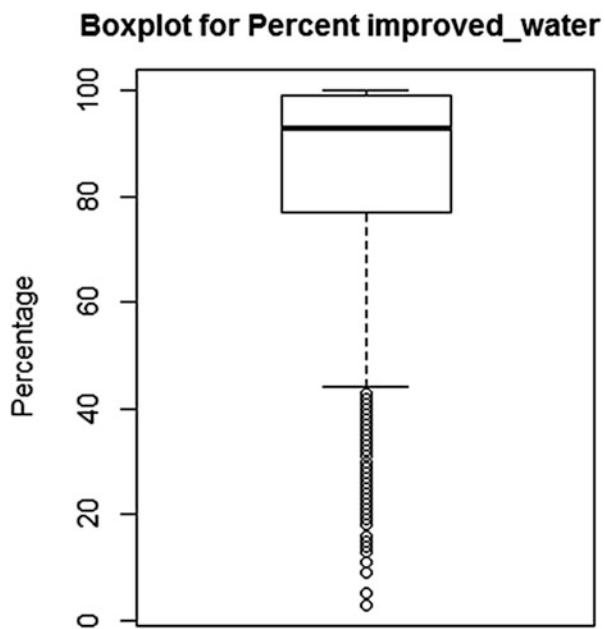
We can visualize the five-number summary by a boxplot (box-and-whiskers plot). With the `boxplot()` function we can specify the title (`main = ""`) and labels for x (`xlab = ""`) and y (`ylab = ""`) axes (Fig. 3.3).

```
boxplot(water$improved_water, main="BoxPlot for Percent improved_water",  
yLab="Percentage")
```

In the boxplot, we have five horizontal lines. Each represents the corresponding value in the five-number summary. The box in the middle represents the middle 50% of values. The bold line in the box is the median. Mean value is not illustrated on the graph.

Boxplots only allow the two ends to extend to a minimum or maximum of 1.5 times the IQR. Therefore, any value that falls outside of the $3 \times IQR$ range will be represented as circles or dots. They are considered as potential outliers. We can see that there are a lot of candidate outliers with small values on the low end of the graph.

Fig. 3.3 Boxplot of the water improvement variable in the WHO dataset



3.8 Visualizing Numeric Variables: Histograms

A histogram is another way to show the spread of a numeric variable (See Chap. 4 for additional information). It uses predetermined number of bins as containers for values to divide the original data. The height of the bins indicates frequency (Figs. 3.4 and 3.5).

```
hist(water$improved_water, main = "Histogram of Percent improved_water", xlab="Percentage")
```

```
hist(water$sanitation_facilities, main = "Histogram of Percent sanitation_facilities", xlab = "Percentage")
```

We could see that the shape of two graphs are somewhat similar. They are both left skewed patterns ($mean < median$). Other common skew patterns are shown in Fig. 3.6.

Fig. 3.4 Histogram plot of the water improvement data

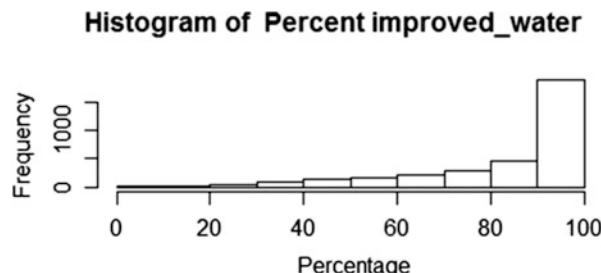


Fig. 3.5 Histogram plot of overall proportion of regions with sanitation facilities (WHO water dataset)

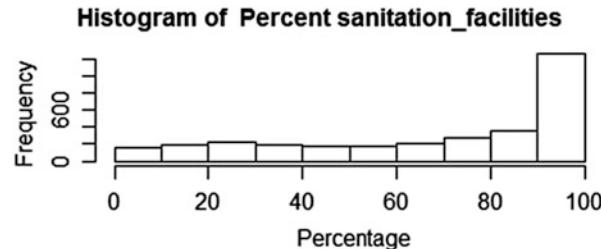


Fig. 3.6 Shape differences between skewed and symmetric distributions



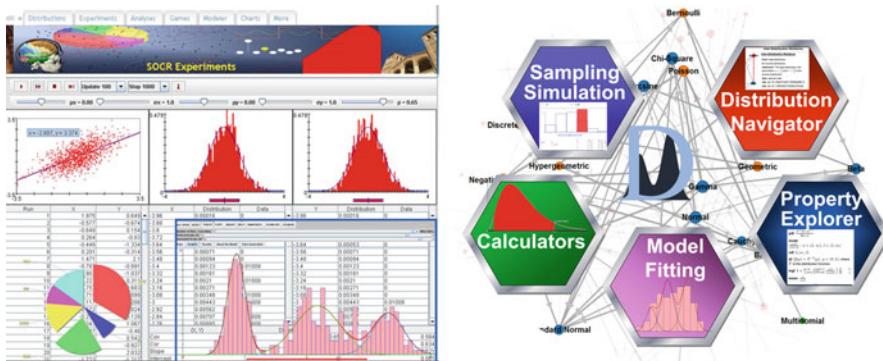


Fig. 3.7 Live visualization demonstrations using SOCR and Distributome resources

You can see the density plots of over 80 different probability distributions using the SOCR Java Distribution Calculators (<http://socr.umich.edu/html/dist/>) or the Distributome HTML5 Distribution Calculators (<http://www.distributome.org/tools.html>), Fig. 3.7.

3.9 Understanding Numeric Data: Uniform and Normal Distributions

If the data follows a uniform distribution, then all values are equally likely to occur in any interval of a fixed width. The histogram for a uniformly distributed dataset would have equal heights for each bin, see Fig. 3.9.

```
x <- rnorm(N, 0, 1)
hist(x, probability=T,
  col='lightblue', xlab=' ', ylab=' ', axes=F,
  main='Normal Distribution')
Lines(density(x, bw=0.4), col='red', Lwd=3)
```

Often, but not always, real world processes behave as normally distributed. A normal distribution would have a higher frequency for middle values and lower frequency for more extreme values. It has a symmetric and bell-curved shape just like in Fig. 3.8. Many parametric-based statistical approaches assume normality of the data. In cases where this parametric assumption is violated, variable transformations or distribution-free tests may be more appropriate.

Fig. 3.8 Overlay of a Normal distribution density (red) and a corresponding Normal sample histogram (blue)

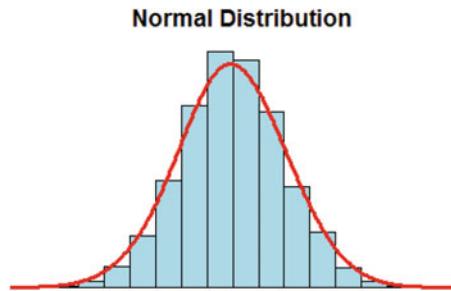
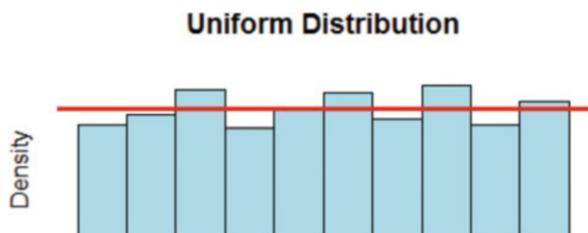


Fig. 3.9 Uniform distribution density and sample histogram plot



3.10 Measuring Spread: Variance and Standard Deviation

Distribution is a great way to characterize data using only a few parameters. For example, normal distribution can be defined by only two parameters: center and spread, or statistically by the mean and standard deviation.

A way to estimate the mean is to divide the sum of the data values by the total number of values. So, we have the following formula:

$$\text{Mean}(X) = \mu = \frac{1}{n} \sum_{i=1}^n x_i.$$

The variance is the average sum of squares and the standard deviation is a square root of the variance:

$$\text{Var}(X) = \sigma^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2$$

$$\text{StdDev}(X) = \sigma = \sqrt{\text{Var}(X)}.$$

Since the water dataset is non-normal, we use a new dataset, including the demographics of baseball players, to illustrate normal distribution properties. The "01_data.txt" dataset has following variables:

Fig. 3.10 Histogram plot of the players' weights, Major League Baseball (MLB) dataset

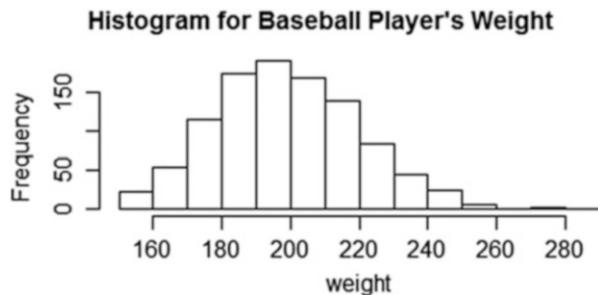
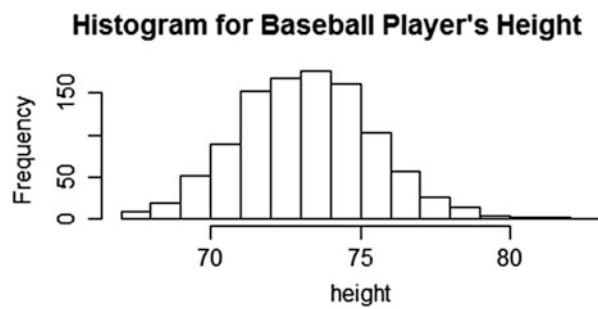


Fig. 3.11 Histogram plot of the players' heights, MLB data



- Name
- Team
- Position
- Height
- Weight
- Age

We can check the histogram for approximate normality of the players' weight and height (Figs. 3.10 and 3.11).

```
baseball<-read.table("https://umich.instructure.com/files/330381/download?do
wnload_frd=1", header=T)
hist(baseball$Weight, main = "Histogram for Baseball Player's Weight", xlab=
"weight")
```

```
hist(baseball$Height, main = "Histogram for Baseball Player's Height", xlab
="height")
```

These plots allow us to visually inspect the normality of the players' heights and weights. We could also obtain mean and standard deviation of the weight and height variables.

```
mean(baseball$Weight)
## [1] 201.7166
mean(baseball$Height)
## [1] 73.69729
var(baseball$Weight)
## [1] 440.9913
sd(baseball$Weight)
## [1] 20.99979
var(baseball$Height)
## [1] 5.316798
sd(baseball$Height)
## [1] 2.305818
```

Larger standard deviation, or variance, suggest the data is more spread out from the mean. Therefore, the weight variable is more spread than the height variable.

Given the first two moments (mean and standard deviation), we can easily estimate how extreme a specific value is. Assuming we have a normal distribution, the values follow a 68 – 95 – 99.7 rule. This means 68% of the data lies within the interval $[\mu - \sigma, \mu + \sigma]$; 95% of the data lies within the interval $[\mu - 2 * \sigma, \mu + 2 * \sigma]$ and 99.7% of the data lies within the interval $[\mu - 3 * \sigma, \mu + 3 * \sigma]$. The following graph plotted by R illustrates the 68 – 95 – 99.7% rule (Fig. 3.12).

Applying the 68-95-99.7 rule to our baseball weight variable, we know that 68% of our players weighted between 180.7 pounds and 222.7 pounds; 95% of the players weighted between 159.7 pounds and 243.7 pounds; And 99.7% of the players weighted between 138.7 pounds and 264.7 pounds.

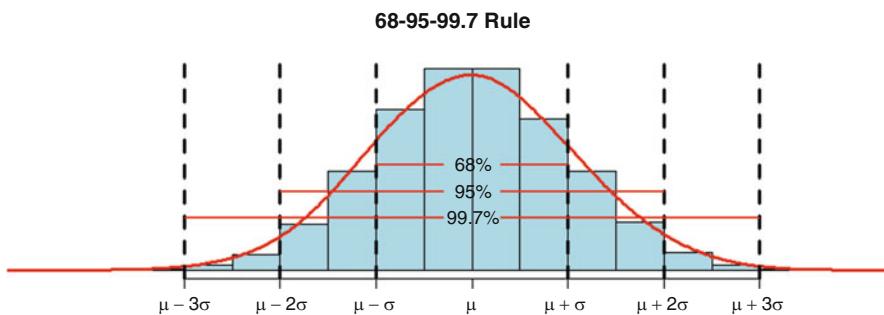


Fig. 3.12 68-95-99.7% rule for Normal distribution

3.11 Exploring Categorical Variables

Back to our water dataset, we can treat the year variable as a categorical rather than a numeric variable. Since the year variable only has six distinctive values, it is reasonable to treat it as a categorical feature, where each value is a category that could apply to multiple WHO regions. Moreover, region and residence area variables are also categorical.

Different from numeric variables, the categorical variables are better examined by tables rather than summary statistics. A one-way tables represent a single categorical variable. It gives us the counts of different categories. The `table()` function can create one-way tables for our water dataset:

```
water <- read.csv('https://umich.instructure.com/files/399172/download?downl
oad_frd=1', header=T)
table(water$Year)

##
## 1990 1995 2000 2005 2010 2012
## 520 561 570 570 556 554

table(water$WHO.region)

##
##           Africa           Americas  Eastern Mediterranean
##           797                  613                  373
##           Europe      South-East Asia  Western Pacific
##           910                  191                  447

table(water$Residence.Area)

##
## Rural Total Urban
## 1095 1109 1127
```

Given that we have a total of 3331 observations, the WHO region table tells us that about 27% (910/3331) of the areas examined in the study are in Europe.

R can directly give us table proportions when using the `prop.table()` function. The proportion values can be transformed as percentages.

```
year_table<-table(water$Year..string.)
prop.table(year_table)

##
##      1990      1995      2000      2005      2010      2012
## 0.1561093 0.1684179 0.1711198 0.1711198 0.1669168 0.1663164

year_pct<-prop.table(year_table)*100
round(year_pct, digits=1)

##
## 1990 1995 2000 2005 2010 2012
## 15.6 16.8 17.1 17.1 16.7 16.6
```

3.12 Exploring Relationships Between Variables

So far, the methods and statistics that we have seen are univariate. Sometimes, we want to examine the relationship between two or multiple variables. For example, did the percentage of the population that uses improved drinking-water sources increase over time? To address such problems, we need to look at bivariate or multivariate relationships.

Visualizing Relationships: scatterplots

Let's look at a bivariate case first. A scatterplot is a good way to visualize bivariate relationships. We have the x-axis and y-axis each representing one of the variables. Each observation is illustrated on the graph by a dot. If the graph shows a clear pattern, rather than a cluster of random dots, the two variables may be correlated with each other.

In R, we can use the `plot()` function to create scatterplots. We have to define the variables for the x and y-axes. The labels in the graph are editable (Fig. 3.13).

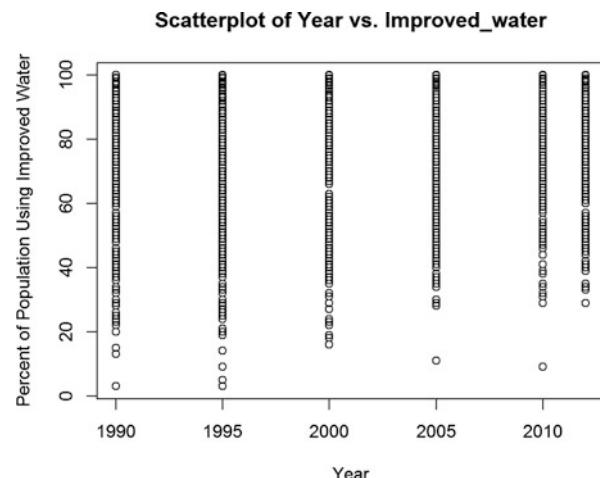
```
plot.window(c(400,1000), c(500,1000))
plot(x=water$year, y=water$improved_water,
      main= "Scatterplot of Year vs. Improved_water",
      xlab= "Year",
      ylab= "Percent of Population Using Improved Water")
```

We can see from the scatterplot that there appears to be a pattern.

Examining Relationships: two-way cross-tabulations

Scatterplot is a useful tool to examine the relationship between two variables where at least one of them is numeric. When both variables are nominal, two-way

Fig. 3.13 Scatterplot of the percent of world population using improved water quality (WHO dataset)



cross-tabulation would be a better choice (also called crosstab or contingency table).

The function `CrossTable()` is available in R under the package `gmodels`. Let's install it first.

```
#install.packages("gmodels", repos = "http://cran.us.r-project.org")
library(gmodels)
```

We are interested in investigating the relationship between World Health Organization (WHO) region and residence area type in the water study. We might want to know if there is a difference in terms of residence area type between the African WHO region and all other WHO regions.

To address this problem, we need to create an indicator variable for the African WHO region first.

```
water$africa<-water$WHO.region=="Africa"
```

Let's revisit the `table()` function to see how many WHO regions are in Africa.

```
table(water$africa)
## FALSE  TRUE
## 2534    797
```

Now, let's create a two-way cross-tabulation using `CrossTable()`.

```
CrossTable(x=water$Residence.Area, y=water$africa)

##
##   Cell Contents
##   |-----|
##   |           N |
##   | Chi-square contribution |
##   |           N / Row Total |
##   |           N / Col Total |
##   |           N / Table Total |
##   |-----|
## 
## Total Observations in Table:  3331
##
##
##           | water$africa
## water$Residence.Area | FALSE | TRUE | Row Total |
## -----|-----|-----|-----|
##   Rural |     828 |   267 | 1095 |
##           | 0.030 | 0.096 |   |
##           | 0.756 | 0.244 | 0.329 |
##           | 0.327 | 0.335 |   |
##           | 0.249 | 0.080 |   |
## -----|-----|-----|-----|
```

##	<i>Total</i>	845	264	1109	
##		0.002	0.007		
##		0.762	0.238	0.333	
##		0.333	0.331		
##		0.254	0.079		
##	<hr/>				
##	<i>Urban</i>	861	266	1127	
##		0.016	0.050		
##		0.764	0.236	0.338	
##		0.340	0.334		
##		0.258	0.080		
##	<hr/>				
##	<i>Column Total</i>	2534	797	3331	
##		0.761	0.239		
##	<hr/>				

Each cell in the table contains five numbers. The first one N gives us the count that fall into its corresponding category. The Chi-square contribution yields information about the cell's contribution in the Pearson's Chi-squared test for independence between two variables. This number measures the probability that the differences in cell counts are due to chance alone.

The numbers of interest include *Col Total* and *Row Total*. In this case, these numbers represent the marginal distributions for residence area type among African regions and the regions in the rest of the world. We can see that the numbers are very close between African and non-African regions for each type of residence area. Therefore, we can conclude that African WHO regions do not have a difference in terms of residence area types compared to the rest of the world.

3.13 Missing Data

In the previous sections, we simply ignored the incomplete observations in our water dataset (`na.rm = TRUE`). Is this an appropriate strategy to handle incomplete data? Could the missingness pattern of those incomplete observations be important? It is possible that the arrangement of the missing observations may reflect an important factor that was not accounted for in our statistics or our models.

Missing Completely at Random (MCAR) is an assumption about the probability of missingness being equal for all cases; **Missing at Random (MAR)** assumes the probability of missingness has a known but random mechanism (e.g., different rates for different groups); **Missing not at Random (MNAR)** suggest a missingness mechanism linked to the values of predictors and/or response, e.g., some participants may drop out of a drug trial when they have side-effects.

There are a number of strategies to impute missing data. The expectation maximization (EM) algorithm provides one example for handling missing data. The SOCR EM tutorial, activity, and documentations provide the theory, applications and practice for effective (multidimensional) EM parameter estimation.

Fig. 3.14 Schematic data representation indexing data values by case (rows) and feature (columns)

$$\begin{array}{c} \text{Data Elements: } j \text{ index} \\ \left[\begin{array}{cccccc} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i_1,1} & x_{i_1,2} & \cdots & \cdots & x_{i_1,j} & \cdots \\ x_{i_2,1} & x_{i_2,2} & \cdots & \cdots & x_{i_2,j} & \cdots \\ \vdots & & \ddots & x_{i,j} & & \vdots \\ \cdots & & & & \ddots & \ddots \end{array} \right] = X \\ \text{Cases: } i \text{ index} \\ i_1 \dots i_l \end{array}$$

The simplest way to handle incomplete data is to substitute each missing value with its (feature or column) average. When the missingness proportion is small, the effect of substituting the means for the missing values will have little effect on the mean, variance, or other important statistics of the data. Also, this will preserve those non-missing values of the same observation or row, see Fig. 3.14.

```

m1<-mean(water$Population.using.improved.drinking, na.rm = T)
m2<-mean(water$Population.using.improved.sanitation, na.rm = T)
water_imp<-water
for(i in 1:3331){
  if(is.na(water_imp$Population.using.improved.drinking[i])){
    water_imp$Population.using.improved.drinking[i]=m1
  }
  if(is.na(water_imp$Population.using.improved.sanitation[i])){
    water_imp$Population.using.improved.sanitation=m2
  }
}
summary(water_imp)

## Year..string.           WHO.region..string.           Country..string.
## Min.   :1990   Africa           :797   Albania      : 18
## 1st Qu.:1995   Americas        :613   Algeria      : 18
## Median :2005   Eastern Mediterranean:373   Andorra      : 18
## Mean   :2002   Europe          :910   Angola       : 18
## 3rd Qu.:2010   South-East Asia  :191   Antigua and Barbuda: 18
## Max.   :2012   Western Pacific  :447   Argentina    : 18
##                                         (Other)      :3223
## Residence.Area.Type..string.
## Rural:1095
## Total:1109
## Urban:1127
## Population.using.improved.drinking.water.sources.....numeric.
## Min.   : 3.0
## 1st Qu.: 77.0
## Median : 93.0
## Mean   : 84.9
## 3rd Qu.: 99.0
## Max.   :100.0
## NA's   :32

```

```

## Population.using.improved.sanitation.facilities.....numeric.
## Min. : 0.00
## 1st Qu.: 42.00
## Median : 81.00
## Mean : 68.87
## 3rd Qu.: 97.00
## Max. :100.00
## NA's :135
## africa Population.using.improved.sanitation
## Mode :Logical Min. :68.87
## FALSE:2534 1st Qu.:68.87
## TRUE :797 Median :68.87
## NA's :0 Mean :68.87
## 3rd Qu.:68.87
## Max. :68.87
##
## Population.using.improved.drinking
## Min. : 3.0
## 1st Qu.: 77.0
## Median : 93.0
## Mean : 84.9
## 3rd Qu.: 99.0
## Max. :100.0

```

A more sophisticated way of resolving missing data is to use a model (e.g., linear regression) to predict the missing feature and impute its missing values. This is called the predictive mean matching approach. This method is good for data with multivariate normality. However, a disadvantage of it is that it can only predict one value at a time, which is very time consuming. Also, the multivariate normality assumption might not be satisfied and there may be important multivariate relations that are not accounted for. We are using the `mi` package to demonstrate predictive mean matching.

Let's install the `mi` package first.

```

# install.packages("mi", repos = "http://cran.us.r-project.org")
library(mi)

```

Then, we need to get the missing information matrix. We are using the imputation method `pmm` (predictive mean matching approach) for both missing variables.

```

mdf<-missing_data.frame(water)
head(mdf)

## Year..string. WHO.region..string. Country..string.
## 1 1990 Africa Algeria
## 2 1990 Africa Angola
## 3 1990 Africa Benin
## 4 1990 Africa Botswana
## 5 1990 Africa Burkina Faso
## 6 1990 Africa Burundi

```

```

##  Residence.Area.Type..string.
## 1          Rural
## 2          Rural
## 3          Rural
## 4          Rural
## 5          Rural
## 6          Rural
##
##  Population.using.improved.drinking.water.sources.....numeric.
## 1          88
## 2          42
## 3          49
## 4          86
## 5          39
## 6          67
##  Population.using.improved.sanitation.facilities.....numeric. africa
## 1          77  TRUE
## 2           7  TRUE
## 3           0  TRUE
## 4          22  TRUE
## 5           2  TRUE
## 6          42  TRUE
##  missing_Population.using.improved.drinking.water.sources.....numeric.
## 1          FALSE
## 2          FALSE
## 3          FALSE
## 4          FALSE
## 5          FALSE
## 6          FALSE
##  missing_Population.using.improved.sanitation.facilities.....numeric.
## 1          FALSE
## 2          FALSE
## 3          FALSE
## 4          FALSE
## 5          FALSE
## 6          FALSE
show(mdf)

## Object of class missing_data.frame with 3331 observations on 7 variables
##
## There are 3 missing data patterns
##
## Append '@patterns' to this missing_data.frame to access the corresponding
## pattern for every observation or perhaps use table()
##
##
## type
## Year..string.
continuous
## WHO.region..string.                      unordered-categorical
## Country..string.                         unordered-categorical
## Residence.Area.Type..string.               unordered-categorical
## Population.using.improved.drinking.water.sources.....numeric.
continuous
## Population.using.improved.sanitation.facilities.....numeric.
continuous
## africa
binary

```

```

##                                         missing
## Year..string.                           0
## WHO.region..string.                     0
## Country..string.                       0
## Residence.Area.Type..string.           0
## Population.using.improved.drinking.water.sources.....numeric. 32
## Population.using.improved.sanitation.facilities.....numeric. 135
## africa                                    0
##                                         method
## Year..string.                            <NA>
...
## africa
<NA>

mdf<-change(mdf, y="Population.using.improved.drinking", what = "imputation_
method", to="pmm")
mdf<-change(mdf, y="Population.using.improved.sanitation", what = "imputatio_
n_method", to="pmm")

```

Notes

- Converting the input `data.frame` to a `missing_data.frame` allows us to include in the DF enhanced metadata about each variable, which is essential for the subsequent modeling, interpretation, and imputation of the initial missing data.
- `show()` displays all missing variables and their class-labels (e.g., continuous), along with meta-data. The `missing_data.frame` constructor suggests the most appropriate classes for each missing variable; however, the user often needs to correct, modify, or change these meta-data, using `change()`.
- Use the `change()` function to change/correct meta-data in the constructed `missing_data.frame` object which may be incorrectly reported by `show(mfd)`.
- To get a sense of the raw data, look at the `summary`, `image`, or `hist` of the `missing_data.frame`.
- The `mi` vignettes provide many useful examples of handling missing data.

Next, we can perform the initial imputation. Here we imputed three times, which will create three different datasets with slightly different imputed values.

```
imputations<-mi(mdf, n.iter=10, n.chains=3, verbose=T)
```

Next, we need to extract several multiply imputed `data.frames` from `imputations` object. Finally, we can compare the summary statistics between the original dataset and the imputed datasets.

```

data.frames <- complete(imputations, 3)
summary(water)

## Year..string.           WHO.region..string.           Country..string.
.
## Min.   :1990   Africa           :797    Albania        : 18
## 1st Qu.:1995   Americas        :613    Algeria        : 18
## Median  :2005   Eastern Mediterranean:373  Andorra        : 18
## Mean    :2002   Europe          :910    Angola         : 18

```

```

## 3rd Qu.:2010   South-East Asia      :191    Antigua and Barbuda: 18
## Max.     :2012   Western Pacific    :447    Argentina            : 18
## (Other)          :3223
## Residence.Area.Type..string.
## Rural:1095
## Total:1109
## Urban:1127
...
## missing_Population.using.improved.sanitation.facilities.....numeric.
## Mode :Logical
## FALSE:3196
## TRUE  :135
## NA's  :0

```

This is just a brief introduction for handling incomplete datasets. In later Chapters, we will discuss more about missing data with different imputation methods and how to evaluate the complete imputed results.

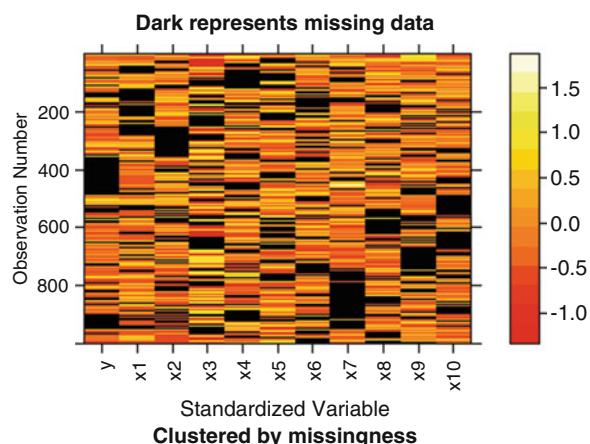
3.13.1 Simulate Some Real Multivariate Data

Suppose we would like to generate a synthetic dataset:

$$\text{sim_data} = \{y, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10\}.$$

Then, we can introduce a method that takes a dataset and a desired proportion of missingness and wipes out the same proportion of the data, i.e., introduces random patterns of missingness. Note that there are already R functions that automate the introduction of missingness, e.g., `missForest::prodNA()`; however, writing such method from scratch is also useful. Figure 3.15 shows the results of introducing 30% missingness in the simulated data.

Fig. 3.15 Incomplete data image plot illustrating the pattern of data missingness



```

set.seed(123)
# create MCAR missing-data generator
create.missing <- function (data, pct.mis = 10)
{
  n <- nrow(data)
  J <- ncol(data)
  if (length(pct.mis) == 1) {
    if(pct.mis>= 0 & pct.mis <=100) {
      n.mis <- rep((n * (pct.mis/100)), J)
    }
    else {
      warning("Percent missing values should be an integer between
              0 and 100! Exiting");
      break
    }
  }
  else {
    if (length(pct.mis) < J)
      stop("The Length of the missing-vector is not equal to the number
            of columns in the data! Exiting!")
    n.mis <- n * (pct.mis/100)
  }
  for (i in 1:ncol(data)) {
    if (n.mis[i] == 0) { # if column has no missing do nothing.
      data[, i] <- data[, i]
    }
    else {
      data[sample(1:n, n.mis[i], replace = FALSE), i] <- NA
      # For each given column (i), sample the row indices (1:n),
      # a number of indices to replace as "missing", n.mis[i], "NA",
      # without replacement
    }
  }
  return(as.data.frame(data))
}

```

Next, let's synthetically generate (simulate) 1,000 cases including all 11 features in the data ($\{y, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10\}$).

```

n <- 1000; u1 <- rbinom(n, 1, .5); v1<-log(rnorm(n, 5, 1)); x1 <- u1*exp(v1)
u2 <- rbinom(n, 1, .5); v2 <- log(rnorm(n, 5, 1)); x2 <- u2*exp(v2)
x3 <- rbinom(n,1,prob=0.45); x4<-ordered(rep(seq(1, 5), n)[sample(1:n, n)]);
x5 <- rep(Letters[1:10], n)[sample(1:n, n)]; x6 <- trunc(runif(n, 1, 10));
x7 <- rnorm(n); x8 <- factor(rep(seq(1, 10), n)[sample(1:n, n)]);
x9 <- runif(n,0.1,0.99); x10 <- rpois(n, 4); y<-x1 + x2 + x7 + x9 + rnorm(n)

# package the simulated data as a data frame object
sim_data <- cbind.data.frame(y, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10)

# randomly create missing values
sim_data_30pct_missing <- create.missing(sim_data, pct.mis=30);
head(sim_data_30pct_missing); summary(sim_data_30pct_missing)

```

```

##          y      x1      x2  x3   x4      x5  x6      x7      x8      x9
## 1      NA      NA 0.00000  0   1      h  8      NA      3      NA
## 2 11.449223      NA 5.236938  0   1      i NA      NA     10 0.2639489
## 3 -1.188296 0.000000 0.000000  0   5      a  3 -1.1469495 <NA> 0.4753195
## 4      NA      NA      NA  0 <NA>      e  6 1.4810186  10 0.6696932
## 5 4.267916 3.490833 0.000000  0 <NA> <NA> NA 0.9161912 <NA> 0.9578455
## 6      NA 0.000000 4.384732  1 <NA>      a NA      NA     10 0.6095176
##      x10
## 1   1
## 2   2
## 3   NA
## 4   3
## 5   8
## 6   6

##          y      x1      x2      x3
##  Min. :-3.846  Min. :0.000  Min. :0.000  Min. :0.0000
##  1st Qu.: 2.410  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.0000
##  Median : 5.646  Median :0.000  Median :3.068  Median :0.0000
##  Mean   : 5.560  Mean   :2.473  Mean   :2.545  Mean   :0.4443
##  3rd Qu.: 8.503  3rd Qu.:4.958  3rd Qu.:4.969  3rd Qu.:1.0000
##  Max.   :16.487  Max.   :8.390  Max.   :8.421  Max.   :1.0000
##  NA's   :300    NA's   :300    NA's   :300    NA's   :300
##      x4      x5      x6      x7      x8
##  1   :138    c   : 80  Min. :1.00  Min. :-2.5689  3   : 78
##  2   :129    h   : 76  1st Qu.:3.00  1st Qu.:-0.6099  7   : 77
##  3   :147    b   : 74  Median :5.00  Median : 0.0202  5   : 75
##  4   :144    a   : 73  Mean   :4.93  Mean   : 0.0435  4   : 73
##  5   :142    j   : 72  3rd Qu.:7.00  3rd Qu.: 0.7519  1   : 70
##  NA's :300  (Other):325  Max.  :9.00  Max.  : 3.7157 (Other):327
##           NA's   :300  NA's   :300  NA's   :300  NA's   :300
##      x9      x10
##  Min. :0.1001  Min.  : 0.000
##  1st Qu.:0.3206 1st Qu.: 2.000
##  Median :0.5312  Median : 4.000
##  Mean   :0.5416  Mean   : 3.929
##  3rd Qu.:0.7772 3rd Qu.: 5.000
##  Max.   :0.9895  Max.   :11.000
##  NA's   :300    NA's   :300

# install.packages("mi")
# install.packages("betareg")
library("betareg"); library("mi")

# get show the missing information matrix
mdf <- missing_data.frame(sim_data_30pct_missing)
show(mdf)

## Object of class missing_data.frame with 1000 observations on 11 variables
##
## There are 542 missing data patterns
##
## Append '@patterns' to this missing_data.frame to access the corresponding
## pattern for every observation or perhaps use table()
##
##          type missing method  model
## y      continuous      300     ppd  linear

```

```

## x1      continuous    300  ppd  linear
## x2      continuous    300  ppd  linear
## x3      binary        300  ppd  logit
## x4      ordered-categorical 300  ppd  ologit
## x5      unordered-categorical 300  ppd  mLogit
## x6      continuous    300  ppd  linear
## x7      continuous    300  ppd  linear
## x8      unordered-categorical 300  ppd  mLogit
## x9      proportion    300  ppd  betareg
## x10     continuous   300  ppd  linear
##
##      family      link transformation
## y      gaussian identity standardize
## x1     gaussian identity standardize
## x2     gaussian identity standardize
## x3     binomial    logit      <NA>
## x4     multinomial logit      <NA>
## x5     multinomial logit      <NA>
## x6     gaussian identity standardize
## x7     gaussian identity standardize
## x8     multinomial logit      <NA>
## x9     binomial    logit      identity
## x10    gaussian identity standardize

# mdf@patterns  # to get the textual missing pattern
image(mdf)  # remember the visual pattern of this MCAR

```

The histogram plots display the distributions of:

- The **observed** data (in **blue color**),
- The **imputed** data (in **red color**), and
- The **completed** values (observed plus imputed, in **gray color**) (Figs. 3.16, 3.17 and 3.18).

```

# Next, try to impute the missing values.

# Get the Graph Parameters (plotting canvas/margins)
# set to plot the histograms for the 3 imputation chains
# mfcol=c(nr, nc). Subsequent histograms are drawn as nr-by-nc arrays on
# the graphics device by columns (mfcol), or rows (mfrow)
# oma: oma=c(bottom, left, top, right) giving the size of the outer
# margins in lines of text
# mar=c(bottom, left, top, right) gives the number of lines of margin
# to be specified on the four sides of the plot.
# tcl=length of tick marks as a fraction of the height of a line of
# text (default=0.5)
par(mfcol=c(5, 5), oma=c(1, 1, 0, 0), mar=c(1, 1, 1, 0), tcl=-0.1,
mgp=c(0, 0, 0))
imputations <- mi(sim_data_30pct_missing, n.iter=5, n.chains=3, verbose=TRUE)
hist(imputations)

```

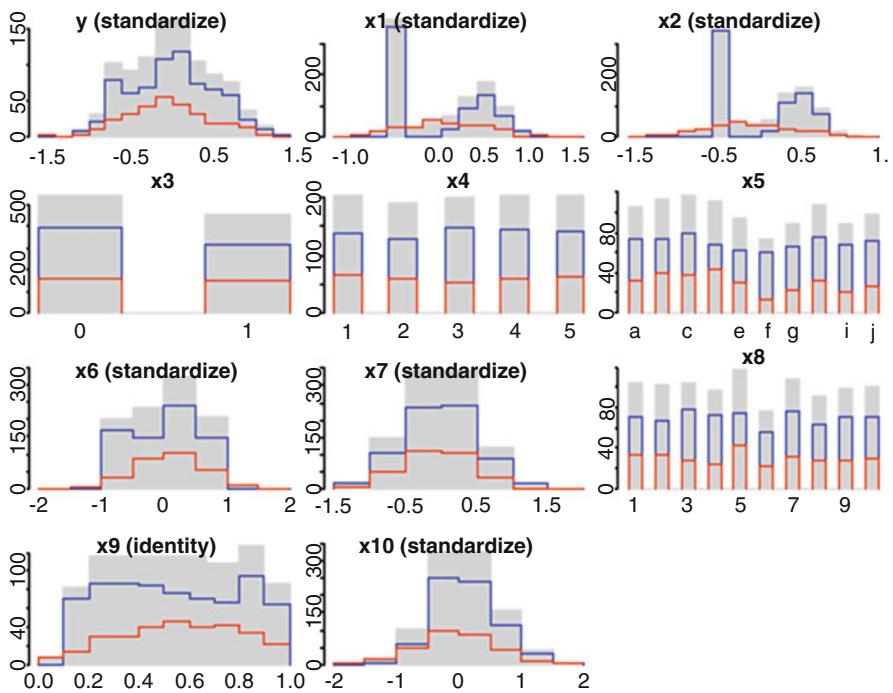


Fig. 3.16 Imputation chain 1: Histogram plots comparing the initially observed (blue), imputed (red), and imputed complete (gray) data

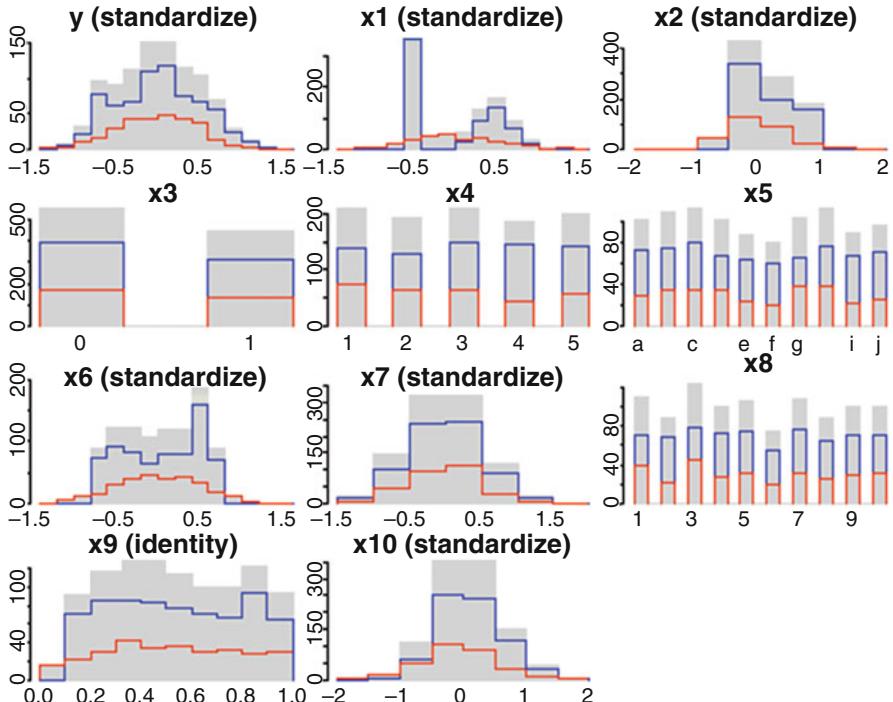


Fig. 3.17 Imputation chain 2: Histogram plots comparing the initially observed (blue), imputed (red), and imputed complete (gray) data

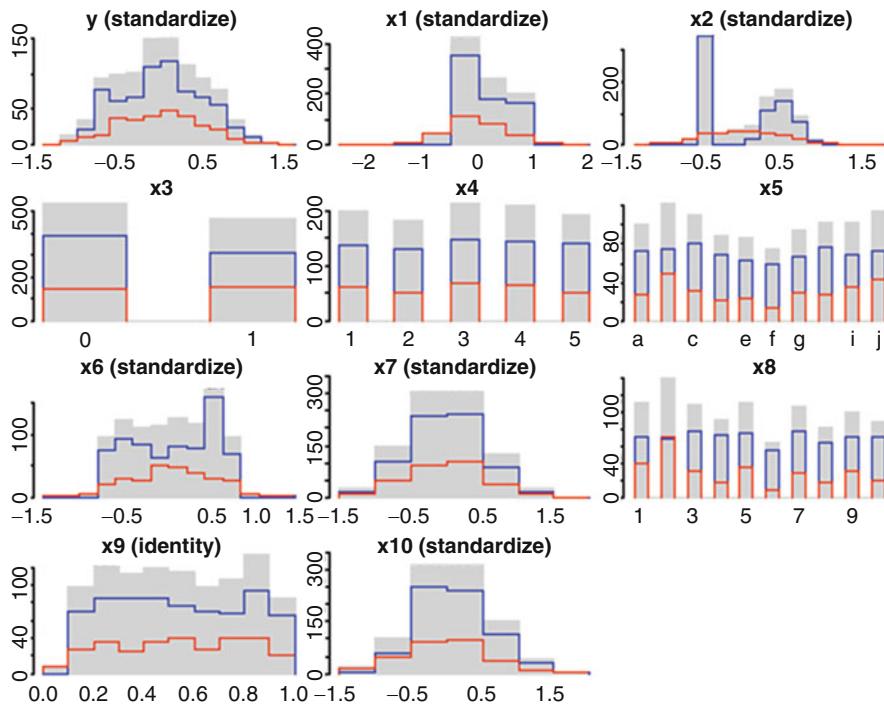


Fig. 3.18 Imputation chain 3: Histogram plots comparing the initially observed (blue), imputed (red), and imputed complete (gray) data

```

# Extracts several multiply imputed data.frames from "imputations" object
data.frames <- complete(imputations, 3)

# Compare the summaries for the original data (prior to introducing missing
# values) with missing data and the re-completed data following imputation
summary(sim_data);summary(sim_data_30pct_missing);summary(data.frames[[1]]);

##          y              x1              x2              x3              x4
##  Min. :-3.846  Min. :0.000  Min. :0.000  Min. :0.000  1:200
##  1st Qu.: 2.489  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.000  2:200
##  Median : 5.549  Median :0.000  Median :2.687  Median :0.000  3:200
##  Mean   : 5.562  Mean   :2.472  Mean   :2.516  Mean   :0.431  4:200
##  3rd Qu.: 8.325  3rd Qu.:4.996  3rd Qu.:5.007  3rd Qu.:1.000  5:200
##  Max.   :16.487  Max.   :8.390  Max.   :8.421  Max.   :1.000
##          x5              x6              x7              x8              x9
##  Min. :0.000  Min. :0.000  Min. :0.000  Min. :0.000  Min. :0.000
##  1st Qu.: 0.000  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.000
##  Median : 0.000  Median :0.000  Median :0.000  Median :0.000  Median :0.000
##  Mean   : 0.000  Mean   :0.000  Mean   :0.000  Mean   :0.000  Mean   :0.000
##  3rd Qu.: 0.000  3rd Qu.:0.000  3rd Qu.:0.000  3rd Qu.:0.000  3rd Qu.:0.000
##  Max.   :0.000  Max.   :0.000  Max.   :0.000  Max.   :0.000  Max.   :0.000
##          x10
##  Min. :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.000
##  3rd Qu.:0.000
##  Max.   :0.000
##          y              x1              x2              x3
##  Min. :-3.846  Min. :0.000  Min. :0.000  Min. :0.0000
##  1st Qu.: 2.410  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.0000
##  Median : 5.646  Median :0.000  Median :3.068  Median :0.0000
##  Mean   : 5.560  Mean   :2.473  Mean   :2.545  Mean   :0.4443
##  3rd Qu.: 8.503  3rd Qu.:4.958  3rd Qu.:4.969  3rd Qu.:1.0000
##  Max.   :16.487  Max.   :8.390  Max.   :8.421  Max.   :1.0000
##          x4              x5              x6              x7              x8
##  Min. :0.0000  Min. :0.000  Min. :0.000  Min. :0.000  Min. :0.000
##  1st Qu.:0.0000  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.000  1st Qu.:0.000
##  Median :0.0000  Median :0.000  Median :0.000  Median :0.000  Median :0.000
##  Mean   :0.0000  Mean   :0.000  Mean   :0.000  Mean   :0.000  Mean   :0.000
##  3rd Qu.:0.0000  3rd Qu.:0.000  3rd Qu.:0.000  3rd Qu.:0.000  3rd Qu.:0.000
##  Max.   :0.0000  Max.   :0.000  Max.   :0.000  Max.   :0.000  Max.   :0.000
##          x9
##  Min. :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.0000
##  3rd Qu.:0.0000
##  Max.   :0.0000
##          x10
##  Min. :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.0000
##  3rd Qu.:0.0000
##  Max.   :0.0000

```

```

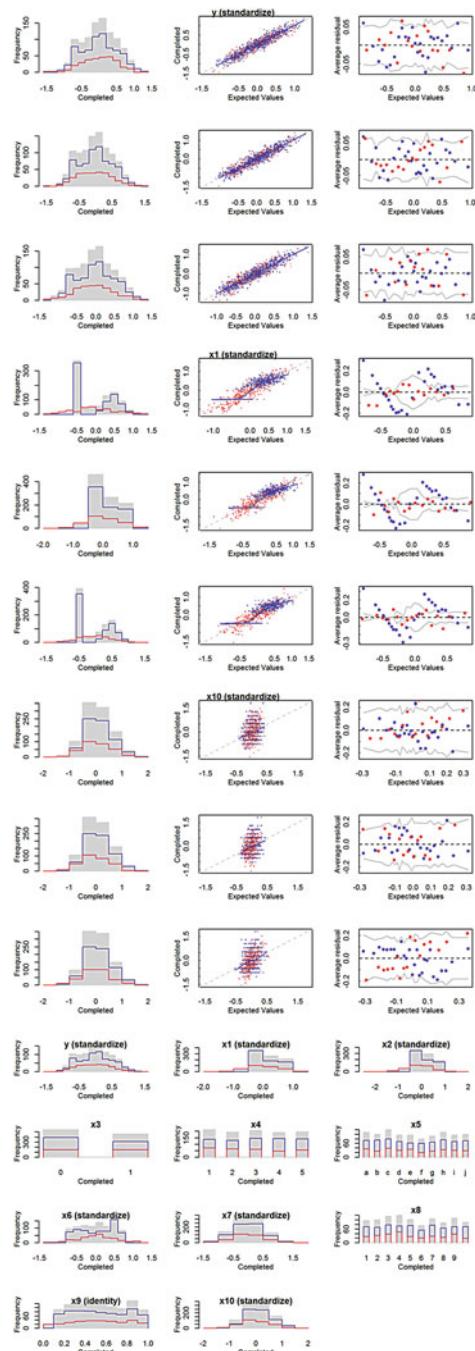
##  missing_x10
##  Mode :Logical
##  FALSE:700
##  TRUE :300
##  NA's :0
Lapply(data.frames, summary)

## $`chain:1`
##   y           x1          x2          x3          x4
##  Min.   :-6.852   Min.   :-3.697   Min.   :-4.920   0:545   1:203
##  1st Qu.: 2.475   1st Qu.: 0.000   1st Qu.: 0.000   1:455   2:189
##  Median : 5.470   Median : 2.510   Median : 1.801   3:201
##  Mean   : 5.458   Mean   : 2.556   Mean   : 2.314   4:202
##  3rd Qu.: 8.355   3rd Qu.: 4.892   3rd Qu.: 4.777   5:205
##  Max.   :16.487   Max.   :10.543   Max.   : 8.864
##
...
##  missing_x10
##  Mode :Logical
##  FALSE:700
##  TRUE :300
##  NA's :0
##
## $`chain:2`
##   y           x1          x2          x3          x4
##  Min.   :-4.724   Min.   :-4.744   Min.   :-5.740   0:558   1:211
##  1st Qu.: 2.587   1st Qu.: 0.000   1st Qu.: 0.000   1:442   2:193
##  Median : 5.669   Median : 2.282   Median : 2.135   3:211
##  Mean   : 5.528   Mean   : 2.486   Mean   : 2.452   4:187
##  3rd Qu.: 8.367   3rd Qu.: 4.884   3rd Qu.: 4.782   5:198
##  Max.   :17.054   Max.   :10.445   Max.   :10.932
...
## $`chain:3`
##   y           x1          x2          x3          x4
##  Min.   :-5.132   Min.   :-8.769   Min.   :-3.643   0:538   1:200
##  1st Qu.: 2.414   1st Qu.: 0.000   1st Qu.: 0.000   1:462   2:182
##  Median : 5.632   Median : 2.034   Median : 2.610   3:215
##  Mean   : 5.537   Mean   : 2.417   Mean   : 2.530   4:211
##  3rd Qu.: 8.434   3rd Qu.: 4.836   3rd Qu.: 4.812   5:192
##  Max.   :16.945   Max.   :10.335   Max.   :11.683
...
##  missing_x10
##  Mode :Logical
##  FALSE:700
##  TRUE :300
##  NA's :0

```

Let's check the imputation convergence (details provided below) (Figs. 3.19 and 3.20).

Fig. 3.19 Plots of the imputation iterations for the simulated dataset



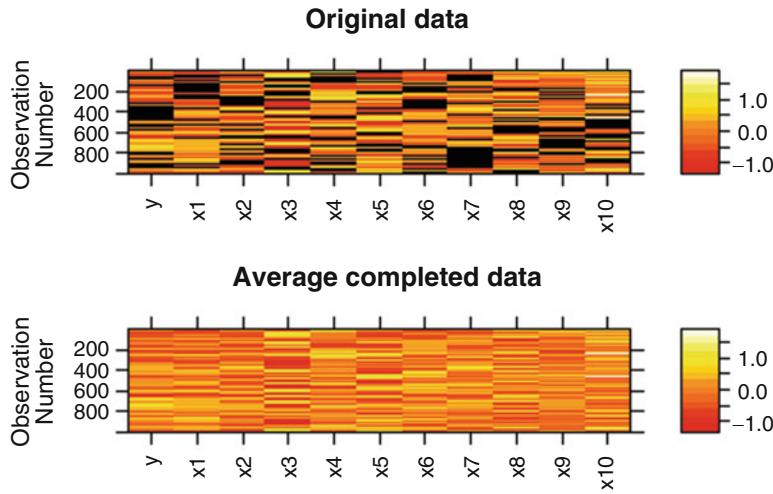


Fig. 3.20 Comparison of the missingness patterns in the raw (top) and imputed (bottom) datasets

```

round(mipply(imputations, mean, to.matrix = TRUE), 3)

##          chain:1 chain:2 chain:3
## y        -0.013  -0.004  -0.003
## x1        0.016  0.003  -0.011
## x2       -0.045  -0.018  -0.003
## x3        1.455  1.442  1.462
## x4        3.017  2.968  3.013
## x5        5.321  5.406  5.480
## x6        0.023  0.004  0.005
## x7       -0.015  -0.005  -0.006
## x8        5.431  5.409  5.202
## x9        0.548  0.536  0.541
## x10      -0.015  -0.020  -0.009
## missing_y  0.300  0.300  0.300
## missing_x1  0.300  0.300  0.300
## missing_x2  0.300  0.300  0.300
## missing_x3  0.300  0.300  0.300
## missing_x4  0.300  0.300  0.300
## missing_x5  0.300  0.300  0.300
## missing_x6  0.300  0.300  0.300
## missing_x7  0.300  0.300  0.300
## missing_x8  0.300  0.300  0.300
## missing_x9  0.300  0.300  0.300
## missing_x10 0.300  0.300  0.300

Rhats(imputations, statistic = "moments")
# assess the convergence of MI algorithm

##   mean_y   mean_x1   mean_x2   mean_x3   mean_x4   mean_x5   mean_x6
## 1.0235026 1.1125720 1.1565542 0.9460979 1.0543446 1.3207898 0.9855947
##   mean_x7   mean_x8   mean_x9   mean_x10      sd_y      sd_x1      sd_x2
## 1.0023935 0.9438358 1.0192697 0.9927675 0.9658852 1.6248062 1.0025950
##   sd_x3     sd_x4     sd_x5     sd_x6     sd_x7     sd_x8     sd_x9
## 0.9463044 1.0706666 1.4470270 1.2510790 0.9008732 1.2865944 1.0195947
##   sd_x10
## 1.1760195

plot(imputations);hist(imputations);image(imputations);summary(imputations)

```

```

## $y
## $y$is_missing
## missing
## FALSE TRUE
## 700 300
##
## $y$imputed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -1.55100 -0.36930 -0.01107 -0.02191 0.30080 1.43600
##
## $y$observed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -1.17500 -0.39350  0.01069  0.00000 0.36770 1.36500
##
## $x1
## $x1$is_missing
## missing
## FALSE TRUE
## 700 300
##
## $x1$imputed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -2.16800 -0.353600 -0.023620 0.008851 0.379800 1.556000
##
## $x1$observed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -0.4768 -0.4768 -0.4768  0.0000 0.4793 1.1410
...
## $x10$observed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -1.01800 -0.49980  0.01851  0.00000 0.27760 1.83200

```

Finally, pool over the $m = 3$ completed datasets when we fit the “model”. In order to estimate a linear regression model, we pool from across the three chains. Figure 3.21 shows the distribution of a simple bivariate linear model ($y = x_1 + x_2$).

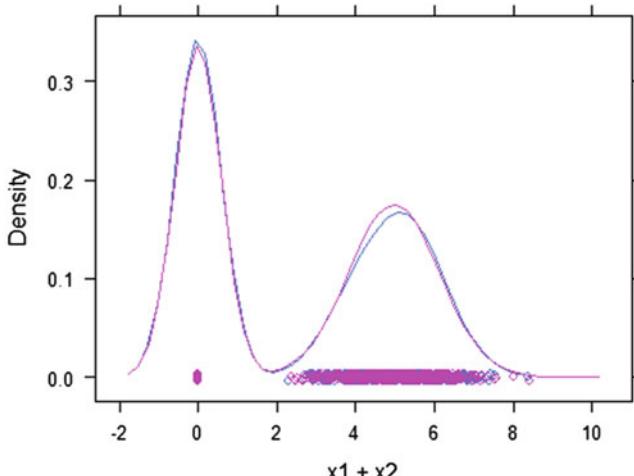


Fig. 3.21 Density plots comparing the observed and imputed outcome variable y

```

model_results<-pool(y~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10, data=imputations, m=3)
display(model_results); summary(model_results)

## bayesglm(formula = y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10, data = imputations, m = 3)
##          coef.est  coef.se
## (Intercept) 0.77      0.84
## x1          0.94      0.05
## x2          0.97      0.04
## x31         -0.27      0.37
## x4.L         0.21      0.21
## x4.Q         -0.09      0.16
## x4.C         0.03      0.24
## x4^4         0.25      0.20
## x5b          0.03      0.42
## x5c          -0.41      0.26
## x5d          -0.22      0.86
## x5e          0.11      0.56
## x5f          -0.13      0.55
## x5g          -0.27      0.67
## x5h          -0.17      0.66
## x5i          -0.69      0.81
## x5j          0.21      0.28
## x6          -0.04      0.07
## x7          0.98      0.09
## x82         0.44      0.39
## x83         0.40      0.20
## x84         -0.14      0.62
## x85         0.20      0.30
## x86         0.19      0.25
## x87         0.19      0.38
## x88         0.51      0.34
## x89         0.25      0.26
## x810        0.17      0.48
## x9          0.88      0.71
## x10         -0.06      0.05
## n = 970, k = 30
## residual deviance = 2056.5, null deviance = 15851.5 (difference=13795.0)
## overdispersion parameter = 2.1
## residual sd is sqrt(overdispersion) = 1.46

## Call:
## pool(formula = y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 +
##       x10, data = imputations, m = 3)
## 

## Deviance Residuals:
##    Min      1Q  Median      3Q      Max
## -2.8821 -0.6925 -0.0005  0.6859  3.7035
## 

## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 0.76906   0.83558   0.920  0.440149    
## x1          0.94250   0.04535  20.781  0.000388 ***  
## x2          0.97495   0.03517  27.721 2.01e-05 ***  
## x31         -0.27349   0.37377  -0.732  0.533696    
## x4.L         0.21116   0.21051   1.003  0.378488    
## x4.Q         -0.08567   0.15627  -0.548  0.602349    
## x4.C         0.02957   0.24490   0.121  0.911557    

```

```

## x4^4      0.24987  0.19504  1.281 0.271639
## x5b      0.03327  0.41563  0.080 0.940649
## x5c     -0.41124  0.25525 -1.611 0.129304
## x5d     -0.21576  0.86290 -0.250 0.824194
## x5e      0.11334  0.56396  0.201 0.854842
## x5f     -0.13162  0.55187 -0.238 0.827734
## x5g     -0.27014  0.67022 -0.403 0.719913
## x5h     -0.16951  0.66294 -0.256 0.818576
## x5i     -0.68619  0.80975 -0.847 0.477639
## x5j      0.20681  0.27823  0.743 0.473891
## x6      -0.04009  0.07306 -0.549 0.633836
## x7      0.98130  0.08527 11.508 0.000197 ***
## x82      0.43774  0.38574  1.135 0.322775
## x83      0.40307  0.20475  1.969 0.049445 *
## x84     -0.13651  0.62307 -0.219 0.843284
## x85      0.19905  0.29973  0.664 0.528335
## x86      0.18662  0.24702  0.755 0.452036
## x87      0.18792  0.38029  0.494 0.647992
## x88      0.51106  0.34272  1.491 0.192478
## x89      0.25125  0.26340  0.954 0.356132
## x810     0.17383  0.47841  0.363 0.740434
## x9       0.87514  0.71484  1.224 0.334593
## x10     -0.05722  0.05035 -1.136 0.331688
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 2.120095)
##
## Null deviance: 15851.5 on 999 degrees of freedom
## Residual deviance: 2056.5 on 970 degrees of freedom
## AIC: 3616.9
## Number of Fisher Scoring iterations: 7

# Report the summaries of the imputations
data.frames <- complete(imputations, 3)      # extract the first 3 chains
lapply(data.frames, summary)

## $`chain:1`#
##      y          x1          x2          x3          x4
##  Min. : -6.852  Min. : -3.697  Min. : -4.920 0:545 1:203
##  1st Qu.: 2.475  1st Qu.: 0.000  1st Qu.: 0.000 1:455 2:189
##  Median : 5.470  Median : 2.510  Median : 1.801 3:201
##  Mean   : 5.458  Mean   : 2.556  Mean   : 2.314 4:202
##  3rd Qu.: 8.355  3rd Qu.: 4.892  3rd Qu.: 4.777 5:205
##  Max.   :16.487  Max.   :10.543  Max.   : 8.864
##
##      x5          x6          x7          x8
##  c   :118  Min. : -4.291  Min. : -2.73138 5   :117
##  b   :113  1st Qu.: 3.000  1st Qu.: -0.61765 7   :109
##  d   :111  Median : 5.000  Median : 0.00085 3   :105
##  h   :108  Mean   : 5.051  Mean   : 0.01486 1   :104
##  a   :105  3rd Qu.: 7.000  3rd Qu.: 0.71796 2   :102
##  j   : 99  Max.   :13.284  Max.   : 3.71572 10  :100
##  (Other):346
##      x9          x10         missing_y      missing_x1
##  Min. :0.0073  Min. : -1.930  Mode :Logical  Mode :Logical
##  1st Qu.:0.3383 1st Qu.: 2.115  FALSE:700   FALSE:700
##  Median :0.5417  Median : 4.000  TRUE :300    TRUE :300

```

```

##  Mean    :0.5476  Mean    : 3.870  NA's :0          NA's :0
##  3rd Qu.:0.7635 3rd Qu.: 5.000
##  Max.   :0.9975  Max.   :11.000
##
##  missing_x2    missing_x3    missing_x4    missing_x5
##  Mode :Logical  Mode :Logical  Mode :Logical  Mode :Logical
##  FALSE:700     FALSE:700     FALSE:700     FALSE:700
##  TRUE :300      TRUE :300      TRUE :300      TRUE :300
##  NA's :0        NA's :0        NA's :0        NA's :0
##
##  missing_x6    missing_x7    missing_x8    missing_x9
##  Mode :Logical  Mode :Logical  Mode :Logical  Mode :Logical
##  FALSE:700     FALSE:700     FALSE:700     FALSE:700
##  TRUE :300      TRUE :300      TRUE :300      TRUE :300
##  NA's :0        NA's :0        NA's :0        NA's :0
##
##  missing_x10
##  Mode :Logical
##  FALSE:700
##  TRUE :300
##  NA's :0
##
## $`chain:2`  

##      y          x1          x2          x3          x4
##  Min. :-4.724  Min. :-4.744  Min. :-5.740  0:558  1:211
##  1st Qu.: 2.587 1st Qu.: 0.000  1st Qu.: 0.000  1:442  2:193
##  Median : 5.669 Median : 2.282  Median : 2.135
##  Mean   : 5.528 Mean   : 2.486  Mean   : 2.452  4:187
##  3rd Qu.: 8.367 3rd Qu.: 4.884  3rd Qu.: 4.782  5:198
##  Max.   :17.054 Max.   :10.445  Max.   :10.932
##
##      x5          x6          x7          x8
##  c   :114  Min.   :-1.498  Min.   :-2.65008  3   :123
##  h   :114  1st Qu.: 3.000  1st Qu.:-0.58182  1   :110
##  b   :109  Median : 5.000  Median : 0.02262  7   :108
##  g   :104  Mean    : 4.948  Mean    : 0.03298  5   :106
##  a   :103  3rd Qu.: 7.000  3rd Qu.: 0.71906  10  :101
##  d   :102  Max.   :12.954  Max.   : 3.71572  4   :100
##  (Other):354
##      x9          x10         missing_y      missing_x1
##  Min. :0.0132  Min. :-1.954  Mode :Logical  Mode :Logical
##  1st Qu.:0.3200 1st Qu.: 2.097  FALSE:700    FALSE:700
##  Median :0.5269 Median : 4.000  TRUE :300     TRUE :300
##  Mean   :0.5357 Mean   : 3.851  NA's :0       NA's :0
##  3rd Qu.:0.7612 3rd Qu.: 5.000
##  Max.   :0.9954 Max.   :11.000
##
##  missing_x2    missing_x3    missing_x4    missing_x5
##  Mode :Logical  Mode :Logical  Mode :Logical  Mode :Logical
##  FALSE:700     FALSE:700     FALSE:700     FALSE:700
##  TRUE :300      TRUE :300      TRUE :300      TRUE :300
##  NA's :0        NA's :0        NA's :0        NA's :0
##
##  missing_x6    missing_x7    missing_x8    missing_x9
##  Mode :Logical  Mode :Logical  Mode :Logical  Mode :Logical
##  FALSE:700     FALSE:700     FALSE:700     FALSE:700
##  TRUE :300      TRUE :300      TRUE :300      TRUE :300
##  NA's :0        NA's :0        NA's :0        NA's :0

```

```

## missing_x10
## Mode :Logical
## FALSE:700
## TRUE :300
## NA's :0
##
## $`chain:3` 
##      y          x1          x2          x3          x4
## Min. :-5.132  Min. :-8.769  Min. :-3.643  0:538  1:200
## 1st Qu.: 2.414  1st Qu.: 0.000  1st Qu.: 0.000  1:462  2:182
## Median : 5.632  Median : 2.034  Median : 2.610          3:215
## Mean   : 5.537  Mean   : 2.417  Mean   : 2.530          4:211
## 3rd Qu.: 8.434  3rd Qu.: 4.836  3rd Qu.: 4.812          5:192
## Max.  :16.945  Max.  :10.335  Max.  :11.683
##
##      x5          x6          x7          x8
## b   :123  Min. :-2.223  Min. :-2.76469  2   :139
## j   :115  1st Qu.: 3.000  1st Qu.:-0.64886  5   :111
## c   :111  Median : 5.000  Median : 0.03266  1   :110
## h   :103  Mean   : 4.957  Mean   : 0.03220  3   :109
## i   :103  3rd Qu.: 7.000  3rd Qu.: 0.71341  7   :106
## a   :100  Max.  :11.785  Max.  : 3.71572  9   :100
## (Other):345          (Other):325
##      x9          x10      missing_y      missing_x1
## Min. :0.007236  Min. :-1.522  Mode :Logical  Mode :Logical
## 1st Qu.:0.320579 1st Qu.: 2.224  FALSE:700    FALSE:700
## Median :0.531962  Median : 4.000  TRUE :300    TRUE :300
## Mean   :0.541147  Mean   : 3.894  NA's :0     NA's :0
## 3rd Qu.:0.772802 3rd Qu.: 5.000
## Max.  :0.992118  Max.  :11.000
##
## missing_x2      missing_x3      missing_x4      missing_x5
## Mode :Logical  Mode :Logical  Mode :Logical  Mode :Logical
## FALSE:700      FALSE:700      FALSE:700      FALSE:700
## TRUE :300      TRUE :300      TRUE :300      TRUE :300
## NA's :0        NA's :0        NA's :0        NA's :0
##
## missing_x6      missing_x7      missing_x8      missing_x9
## Mode :Logical  Mode :Logical  Mode :Logical  Mode :logical
## FALSE:700      FALSE:700      FALSE:700      FALSE:700
## TRUE :300      TRUE :300      TRUE :300      TRUE :300
## NA's :0        NA's :0        NA's :0        NA's :0
##
## missing_x10
## Mode :Logical
## FALSE:700
## TRUE :300
## NA's :0

coef(summary(model_results))[, 1:2] # get the model coef's and their SE's
##                   Estimate Std. Error
## (Intercept)  0.76906403 0.83558319
## x1          0.94250085 0.04535482
## x2          0.97494755 0.03517050
## x31         -0.27348764 0.37377108
## x4.L         0.21116072 0.21050885
## x4.Q         -0.08566591 0.15626753
## x4.C         0.02957084 0.24489775

```

```

## x4^4      0.24986739 0.19503550
## x5b      0.03327092 0.41562660
## x5c      -0.41123612 0.25525293
## x5d      -0.21576243 0.86289626
## x5e      0.11334262 0.56396149
## x5f      -0.13161632 0.55187362
## x5g      -0.27013537 0.67021765
## x5h      -0.16951449 0.66293826
## x5i      -0.68618715 0.80974757
## x5j      0.20681081 0.27822872
## x6      -0.04008539 0.07305886
## x7      0.98130349 0.08526896
## x82      0.43773548 0.38574473
## x83      0.40306683 0.20475089
## x84      -0.13651311 0.62306988
## x85      0.19905219 0.29973411
## x86      0.18661601 0.24702280
## x87      0.18792270 0.38028640
## x88      0.51105644 0.34272121
## x89      0.25124560 0.26340027
## x810     0.17382802 0.47841078
## x9      0.87514342 0.71483501
## x10     -0.05721504 0.05035334

library("lattice")
densityplot(y ~ x1 + x2, data=imputations)

```

This plot, Fig. 3.21, allows us to compare the density of observed data and imputed data—these should be similar (though not identical) under MAR assumptions.

3.13.2 TBI Data Example

Next, we will see an example using the traumatic brain injury (TBI) dataset.

```

# Load the (raw) data from the table into a plain text file "08_EpiBioSData_
Incomplete.csv"
TBI_Data <- read.csv("https://umich.instructure.com/files/720782/download?do
wnload_frd=1", na.strings=c("", ".", "NA"))
summary(TBI_Data)

##           id          age         sex      mechanism
##  Min.   : 1.00   Min.   :16.00  Female: 9  Bike_vs_Auto: 4
##  1st Qu.:12.25  1st Qu.:23.00  Male   :37   Blunt      : 4
##  Median :23.50  Median :33.00          Fall      :13
##  Mean   :23.50  Mean   :36.89          GSW       : 2
##  3rd Qu.:34.75  3rd Qu.:47.25          MCA       : 7
##  Max.   :46.00  Max.   :83.00          MVA       :10
##                               Peds_vs_Auto: 6
##           field.gcs      er.gcs      icu.gcs      worst.gcs
##  Min.   : 3   Min.   : 3.000   Min.   : 0.000   Min.   : 0.0
##  1st Qu.: 3   1st Qu.: 4.000   1st Qu.: 3.000   1st Qu.: 3.0

```

```

## Median : 7  Median : 7.500  Median : 6.000  Median : 3.0
## Mean   : 8  Mean   : 8.182  Mean   : 6.378  Mean   : 5.4
## 3rd Qu.:12 3rd Qu.:12.250 3rd Qu.: 8.000 3rd Qu.: 7.0
## Max.   :15  Max.   :15.000  Max.   :14.000  Max.   :14.0
## NA's   :2   NA's   :2     NA's   :1     NA's   :1
##      X6m.gose      X2013.gose      skull.fx      temp.injury
## Min.  :2.000  Min.  :2.000  Min.  :0.0000  Min.  :0.000
## 1st Qu.:3.000 1st Qu.:5.000 1st Qu.:0.0000 1st Qu.:0.000
## Median :5.000  Median :7.000  Median :1.0000  Median :1.000
## Mean   :4.805  Mean   :5.804  Mean   :0.6087  Mean   :0.587
## 3rd Qu.:6.000 3rd Qu.:7.000 3rd Qu.:1.0000 3rd Qu.:1.000
## Max.   :8.000  Max.   :8.000  Max.   :1.0000  Max.   :1.000
## NA's   :5
##      surgery      spikes.hr      min.hr      max.hr
## Min.  :0.0000  Min.  : 1.280  Min.  : 0.000  Min.  : 12.00
## 1st Qu.:0.0000 1st Qu.: 5.357 1st Qu.: 0.000 1st Qu.: 35.25
## Median :1.0000  Median :18.170  Median : 0.000  Median : 97.50
## Mean   :0.6304  Mean   :52.872  Mean   : 3.571  Mean   :241.89
## 3rd Qu.:1.0000 3rd Qu.:57.227 3rd Qu.: 0.000 3rd Qu.:312.75
## Max.   :1.0000  Max.   :294.000  Max.   :42.000  Max.   :1199.00
## NA's   :18     NA's   :18     NA's   :18     NA's   :18
##      acute.sz      late.sz      ever.sz
## Min.  :0.0000  Min.  :0.0000  Min.  :0.000
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.000
## Median :0.0000  Median :1.0000  Median :1.000
## Mean   :0.1739  Mean   :0.5652  Mean   :0.587
## 3rd Qu.:0.0000 3rd Qu.:1.0000 3rd Qu.:1.000
## Max.   :1.0000  Max.   :1.0000  Max.   :1.000
## 
```

1. Convert to a `missing_data.frame` (Fig. 3.22)

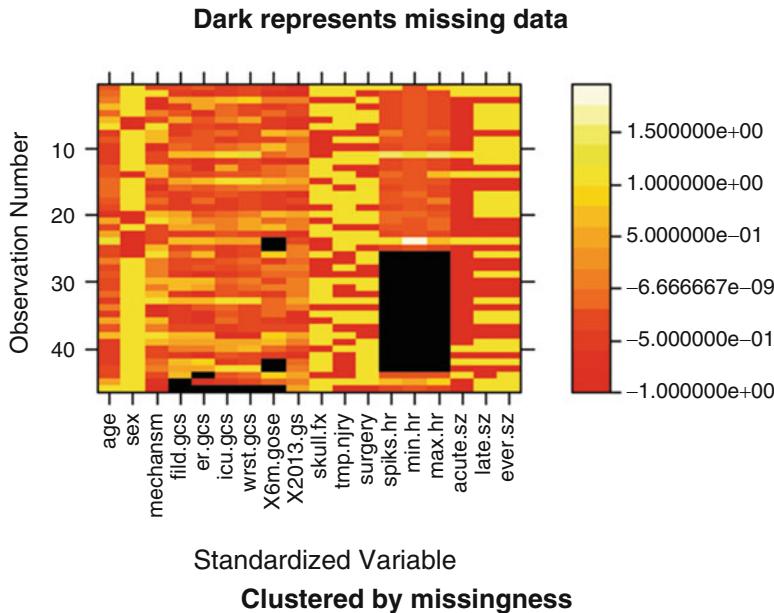


Fig. 3.22 Missing data pattern for the TBI case-study

```

# Get information matrix of the data
# library("betareg"); library("mi")
mdf <- missing_data.frame(TBI_Data) # warnings about missingness patterns

## NOTE: The following pairs of variables appear to have the same missingness pattern.
## Please verify whether they are in fact logically distinct variables.
## [,1]      [,2]
## [1,] "icu.gcs" "worst.gcs"

show(mdf); mdf@patterns; image(mdf)

## Object of class missing_data.frame with 46 observations on 19 variables
##
## There are 7 missing data patterns
##
## Append '@patterns' to this missing_data.frame to access the corresponding pattern for every observation or perhaps use table()
##
##          type missing method  model
## id      irrelevant 0 <NA> <NA>
## age     continuous 0 <NA> <NA>
## sex     binary    0 <NA> <NA>
## mechanism  unordered-categorical 0 <NA> <NA>
## field.gcs  continuous 2 ppd linear
## er.gcs    continuous 2 ppd Linear
## icu.gcs   continuous 1 ppd linear
## worst.gcs  continuous 1 ppd linear
## X6m.gose  continuous 5 ppd linear
## X2013.gose continuous 0 <NA> <NA>
## skull.fx  binary    0 <NA> <NA>
## temp.injury binary    0 <NA> <NA>
## surgery   binary    0 <NA> <NA>
## spikes.hr  continuous 18 ppd Linear
## min.hr    continuous 18 ppd linear
## max.hr    continuous 18 ppd linear
## acute.sz   binary    0 <NA> <NA>
## late.sz   binary    0 <NA> <NA>
## ever.sz   binary    0 <NA> <NA>
##
##          family  Link transformation
## id       <NA> <NA> <NA>
## age     <NA> <NA> standardize
## sex     <NA> <NA> <NA>
## mechanism <NA> <NA> <NA>
## field.gcs gaussian identity standardize
## er.gcs   gaussian identity standardize
## icu.gcs  gaussian identity standardize
## worst.gcs gaussian identity standardize
## X6m.gose gaussian identity standardize
## X2013.gose <NA> <NA> standardize
## skull.fx  <NA> <NA> <NA>
## temp.injury <NA> <NA> <NA>
## surgery   <NA> <NA> <NA>
## spikes.hr  gaussian identity standardize
## min.hr    gaussian identity standardize
## max.hr    gaussian identity standardize
## acute.sz   <NA> <NA> <NA>
## late.sz   <NA> <NA> <NA>

```

```

## ever.sz      <NA>      <NA>      <NA>
## [1] spikes.hr, min.hr, max.hr
## [2] field.gcs
## [3] nothing
## [4] nothing
## [5] nothing
## [6] nothing
## [7] spikes.hr, min.hr, max.hr
## [8] nothing
## [9] nothing
## [10] nothing
## [11] nothing
## [12] nothing
## [13] spikes.hr, min.hr, max.hr
## [14] nothing
## [15] spikes.hr, min.hr, max.hr
## [16] spikes.hr, min.hr, max.hr
## [17] nothing
## [18] spikes.hr, min.hr, max.hr
## [19] spikes.hr, min.hr, max.hr
## [20] spikes.hr, min.hr, max.hr
## [21] X6m.gose, spikes.hr, min.hr, max.hr
## [22] nothing
## [23] spikes.hr, min.hr, max.hr
## [24] spikes.hr, min.hr, max.hr
## [25] spikes.hr, min.hr, max.hr
## [26] spikes.hr, min.hr, max.hr
## [27] spikes.hr, min.hr, max.hr
## [28] X6m.gose
## [29] spikes.hr, min.hr, max.hr
## [30] nothing
## [31] X6m.gose, spikes.hr, min.hr, max.hr
## [32] spikes.hr, min.hr, max.hr
## [33] nothing
## [34] nothing
## [35] nothing
## [36] nothing
## [37] field.gcs, er.gcs, icu.gcs, worst.gcs, X6m.gose
## [38] er.gcs
## [39] nothing
## [40] nothing
## [41] nothing
## [42] spikes.hr, min.hr, max.hr
## [43] nothing
## [44] nothing
## [45] nothing
## [46] X6m.gose
## 7 Levels: nothing field.gcs X6m.gose er.gcs ... field.gcs, er.gcs, icu.gcs, worst.gcs, X6m.gose

```

2. Configuring the imputation process.

```

# mi::change() method changes the family imputation method,
# size, type, and so forth of a missing variable. It's called
# before calling mi to affect how the conditional expectation of each
# missing variable is modeled.
mdf <- change(mdf, y = "spikes.hr", what = "transformation", to="identity")

```

3. Examine the missingness patterns.

```
summary(mdf); hist(mdf);

##      id          age          sex      mechanism
##  Min.   : 1.00   Min.   :16.00   Female: 9   Bike_vs_Auto: 4
##  1st Qu.:12.25  1st Qu.:23.00   Male   :37    Blunt      : 4
##  Median :23.50  Median :33.00
##  Mean   :23.50  Mean   :36.89
##  3rd Qu.:34.75  3rd Qu.:47.25
##  Max.   :46.00  Max.   :83.00
## 
##      field.gcs      er.gcs      icu.gcs      worst.gcs
##  Min.   : 3   Min.   : 3.000   Min.   : 0.000   Min.   : 0.0
##  1st Qu.: 3   1st Qu.: 4.000   1st Qu.: 3.000   1st Qu.: 3.0
##  Median : 7   Median : 7.500   Median : 6.000   Median : 3.0
##  Mean   : 8   Mean   : 8.182   Mean   : 6.378   Mean   : 5.4
##  3rd Qu.:12   3rd Qu.:12.250   3rd Qu.: 8.000   3rd Qu.: 7.0
##  Max.   :15   Max.   :15.000   Max.   :14.000   Max.   :14.0
##  NA's   : 2   NA's   : 2       NA's   : 1       NA's   : 1
##      X6m.gose      X2013.gose      skull.fx      temp.injury
##  Min.   :2.000   Min.   :2.000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:3.000   1st Qu.:5.000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :5.000   Median :7.000   Median :1.0000   Median :1.000
##  Mean   :4.805   Mean   :5.804   Mean   :0.6087   Mean   :0.587
##  3rd Qu.:6.000   3rd Qu.:7.000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :8.000   Max.   :8.000   Max.   :1.0000   Max.   :1.000
##  NA's   : 5
##      surgery      spikes.hr      min.hr      max.hr
##  Min.   :0.0000   Min.   : 1.280   Min.   : 0.000   Min.   : 12.00
##  1st Qu.:0.0000   1st Qu.: 5.357   1st Qu.: 0.000   1st Qu.: 35.25
##  Median :1.0000   Median :18.170   Median : 0.000   Median : 97.50
##  Mean   :0.6304   Mean   :52.872   Mean   : 3.571   Mean   :241.89
##  3rd Qu.:1.0000   3rd Qu.:57.227   3rd Qu.: 0.000   3rd Qu.:312.75
##  Max.   :1.0000   Max.   :294.000   Max.   :42.000   Max.   :1199.00
##  NA's   :18
##      acute.sz      late.sz      ever.sz
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.0000   Median :1.0000   Median :1.000
##  Mean   :0.1739   Mean   :0.5652   Mean   :0.587
##  3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.000
## 

## 

image(mdf)
```

4. Perform the initial imputation (Fig. 3.23).

```
imputations <- mi(mdf, n.iter=10, n.chains=5, verbose=TRUE)
hist(imputations)
```

5. Extracts several multiply imputed `data.frames` from the "imputations" object.

```
data.frames <- complete(imputations, 5)
```

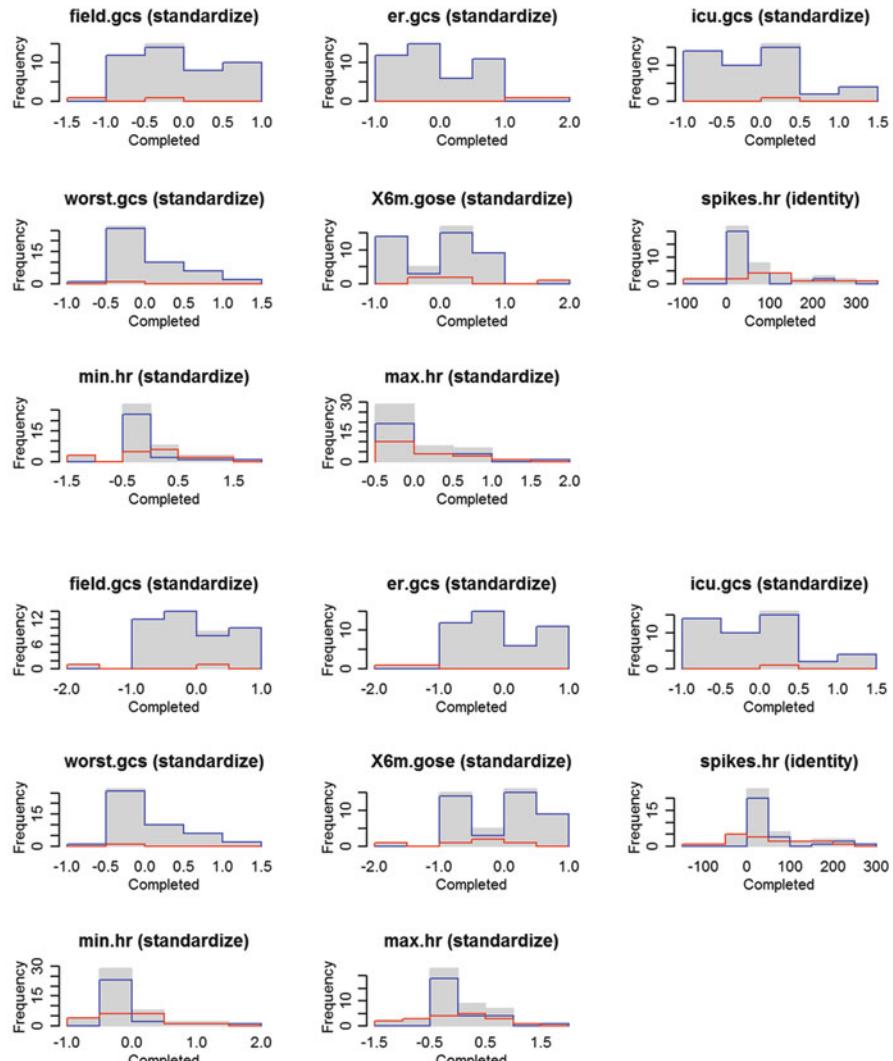


Fig. 3.23 Validation plots for the original, imputed and complete TBI datasets

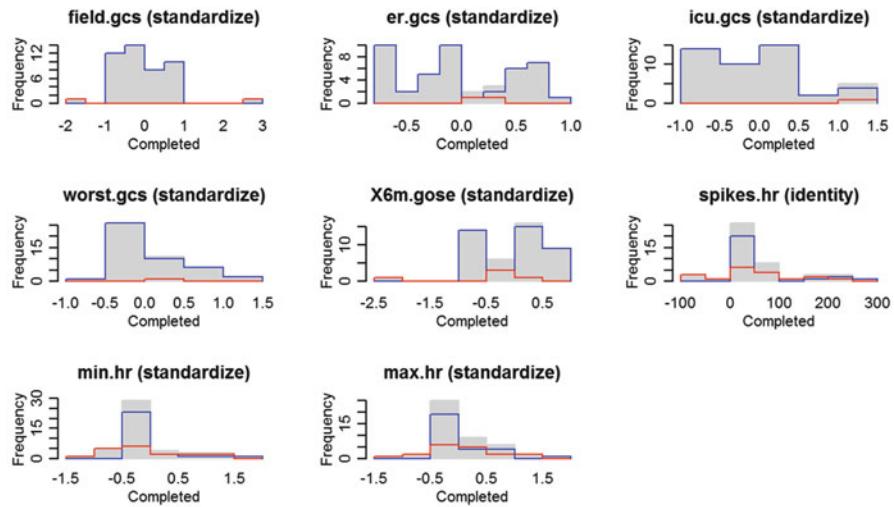


Fig. 3.23 (continued)

6. Report a list of "summaries" for each element (imputation instance).

```
lapply(data.frames, summary)

## $`chain:1`  

##      id          age          sex      mechanism  

##  Min. :1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4  

##  1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4  

##  Median :23.50 Median :33.00          Fall      :13  

##  Mean   :23.50 Mean  :36.89          GSW       : 2  

##  3rd Qu.:34.75 3rd Qu.:47.25          MCA       : 7  

##  Max.  :46.00  Max. :83.00          MVA       :10  

##                                         Peds_vs_Auto: 6  

##      field.gcs          er.gcs          icu.gcs          worst.gcs  

##  Min. :-3.424  Min. : 3.000  Min. : 0.000  Min. : 0.000  

##  1st Qu.: 3.000 1st Qu.: 4.250  1st Qu.: 3.000  1st Qu.: 3.000  

##  Median : 6.500  Median : 8.000  Median : 6.000  Median : 3.000  

##  Mean   : 7.593  Mean  : 8.442  Mean  : 6.285  Mean  : 5.494  

##  3rd Qu.:12.000 3rd Qu.:13.000 3rd Qu.: 7.750 3rd Qu.: 7.750  

##  Max.  :15.000  Max. :15.000  Max. :14.000  Max. :14.000  

##  

##      X6m.gose      X2013.gose      skull.fx temp.injury surgery  

##  Min. :2.000  Min. :2.000  0:18    0:19    0:17  

##  1st Qu.:3.000 1st Qu.:5.000  1:28    1:27    1:29  

##  Median :5.000  Median :7.000  

##  Mean   :5.031  Mean  :5.804  

##  3rd Qu.:6.815 3rd Qu.:7.000  

##  Max.  :8.169  Max. :8.000
```

```

##   spikes.hr      min.hr      max.hr      acute.sz  late.sz
## Min.   :-86.914  Min.   :-11.697  Min.   :-153.94  0:38    0:20
## 1st Qu.: 3.953   1st Qu.: 0.000   1st Qu.: 42.25   1: 8     1:26
## Median : 28.125  Median : 0.000   Median : 211.49
## Mean   : 59.108  Mean   : 7.133   Mean   : 282.79
## 3rd Qu.:113.615 3rd Qu.:11.329   3rd Qu.: 390.63
## Max.   :294.000  Max.   :43.706   Max.   :1199.00
##
## ever.sz missing_field.gcs missing_er.gcs missing_icu.gcs
## 0:19   Mode :logical      Mode :logical      Mode :logical
## 1:27   FALSE:44          FALSE:44          FALSE:45
## TRUE  :2               TRUE  :2           TRUE  :1
## NA's  :0               NA's  :0           NA's  :0
##
## missing_worst.gcs missing_X6m.gose missing_spikes.hr missing_min.hr
## Mode :logical      Mode :logical      Mode :logical      Mode :logical
## FALSE:45          FALSE:41          FALSE:28          FALSE:28
## TRUE  :1           TRUE  :5            TRUE  :18          TRUE  :18
## NA's  :0           NA's  :0           NA's  :0           NA's  :0
##
## missing_max.hr
## Mode :logical
## FALSE:28
## TRUE  :18
## NA's  :0
##
## $`chain:2`
##      id      age      sex      mechanism
## Min.   : 1.00  Min.   :16.00  Female: 9  Bike_vs_Auto: 4
## 1st Qu.:12.25 1st Qu.:23.00  Male   :37   Blunt      : 4
## Median :23.50  Median :33.00
## Mean   :23.50  Mean   :36.89  Fall      :13
## 3rd Qu.:34.75 3rd Qu.:47.25  GSW       : 2
## Max.   :46.00  Max.   :83.00  MCA       : 7
##                                     MVA       :10
##                                     Peds_vs_Auto: 6
##      field.gcs      er.gcs      icu.gcs      worst.gcs
## Min.   :-3.324  Min.   : 3.000  Min.   : 0.00  Min.   : 0.000
## 1st Qu.: 3.000  1st Qu.: 4.000  1st Qu.: 3.00  1st Qu.: 3.000
## Median : 6.500  Median : 7.000  Median : 6.00  Median : 3.000
## Mean   : 7.658  Mean   : 8.046  Mean   : 6.24  Mean   : 5.466
## 3rd Qu.:12.000  3rd Qu.:12.000 3rd Qu.: 7.75  3rd Qu.: 7.750
## Max.   :15.000  Max.   :15.000  Max.   :14.00  Max.   :14.000
##
##      X6m.gose      X2013.gose      skull_fx temp.injury surgery
## Min.   :-3.196  Min.   : 2.000  0:18    0:19    0:17
## 1st Qu.: 3.000  1st Qu.: 5.000  1:28    1:27    1:29
## Median : 5.000  Median : 7.000
## Mean   : 4.755  Mean   : 5.804
## 3rd Qu.: 6.117  3rd Qu.: 7.000
## Max.   : 8.000  Max.   : 8.000
##
##   spikes.hr      min.hr      max.hr      acute.sz  late.sz
## Min.   :-138.854  Min.   :-30.603  Min.   :-432.95  0:38    0:20
## 1st Qu.: 5.518   1st Qu.: 0.000   1st Qu.: 28.75   1: 8     1:26
## Median : 34.522  Median : 0.000   Median : 97.50
## Mean   : 61.310  Mean   : 2.329   Mean   : 209.53
## 3rd Qu.: 97.394  3rd Qu.: 4.306   3rd Qu.: 306.98

```

```

##  Max.   : 294.000   Max.   : 42.000   Max.   :1336.12
##
##  ever.sz missing_field.gcs missing_er.gcs missing_icu.gcs
##  0:19   Mode :logical      Mode :logical      Mode :logical
##  1:27   FALSE:44          FALSE:44          FALSE:45
##          TRUE :2           TRUE :2           TRUE :1
##          NA's :0           NA's :0           NA's :0
##
##  missing_worst.gcs missing_X6m.gose missing_spikes.hr missing_min.hr
##  Mode :Logical      Mode :logical      Mode :logical      Mode :logical
##  FALSE:45          FALSE:41          FALSE:28          FALSE:28
##  TRUE :1           TRUE :5           TRUE :18          TRUE :18
##  NA's :0           NA's :0           NA's :0           NA's :0
##
##  missing_max.hr
##  Mode :Logical
##  FALSE:28
##  TRUE :18
##  NA's :0
##
##  $`chain:3`
##    id      age      sex      mechanism
##  Min.   : 1.00  Min.   :16.00  Female: 9  Bike_vs_Auto: 4
##  1st Qu.:12.25 1st Qu.:23.00  Male   :37   Blunt   : 4
##  Median :23.50 Median :33.00          Fall   :13
##  Mean   :23.50 Mean   :36.89          GSW   : 2
##  3rd Qu.:34.75 3rd Qu.:47.25          MCA   : 7
##  Max.   :46.00  Max.   :83.00          MVA   :10
##                                     Peds_vs_Auto: 6
##    field.gcs      er.gcs      icu.gcs      worst.gcs
##  Min.   : 3.000  Min.   : 3.000  Min.   : 0.000  Min.   : 0.000
##  1st Qu.: 3.250 1st Qu.: 4.191  1st Qu.: 3.000  1st Qu.: 3.000
##  Median : 7.000 Median : 7.500  Median : 6.000  Median : 3.000
##  Mean   : 8.218  Mean   : 8.513  Mean   : 6.325  Mean   : 5.304
##  3rd Qu.:12.000 3rd Qu.:12.750 3rd Qu.: 7.750  3rd Qu.: 7.000
##  Max.   :17.978  Max.   :26.831  Max.   :14.000  Max.   :14.000
##
##    X6m.gose      X2013.gose    skull.fx temp.injury surgery
##  Min.   :2.000  Min.   :2.000  0:18   0:19   0:17
##  1st Qu.:3.000  1st Qu.:5.000  1:28   1:27   1:29
##  Median :5.000  Median :7.000
##  Mean   :4.892  Mean   :5.804
##  3rd Qu.:6.000  3rd Qu.:7.000
##  Max.   :8.000  Max.   :8.000
##
##    spikes.hr      min.hr      max.hr      acute.sz late.sz
##  Min.   :-40.459  Min.   :-27.222  Min.   :-236.6  0:38   0:20
##  1st Qu.: 5.518  1st Qu.:  0.000  1st Qu.: 37.5  1: 8   1:26
##  Median :34.864  Median :  0.000  Median :195.6
##  Mean   :65.781  Mean   : 2.619  Mean   :281.8
##  3rd Qu.:100.137 3rd Qu.: 5.681  3rd Qu.:476.1
##  Max.   :294.000  Max.   :42.000  Max.   :1199.0
##
##  ever.sz missing_field.gcs missing_er.gcs missing_icu.gcs
##  0:19   Mode :logical      Mode :logical      Mode :logical
##  1:27   FALSE:44          FALSE:44          FALSE:45
##          TRUE :2           TRUE :2           TRUE :1

```

```

##          NA's :0          NA's :0          NA's :0
##
##  missing_worst.gcs missing_X6m.gose missing_spikes.hr missing_min.hr
##  Mode :Logical      Mode :Logical      Mode :Logical      Mode :Logical
##  FALSE:45          FALSE:41          FALSE:28          FALSE:28
##  TRUE :1           TRUE :5           TRUE :18          TRUE :18
##  NA's :0           NA's :0           NA's :0           NA's :0
##
##  missing_max.hr
##  Mode :Logical
##  FALSE:28
##  TRUE :18
##  NA's :0
##
## $`chain:4`:
##          id          age          sex      mechanism
##  Min. : 1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4
##  1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4
##  Median :23.50 Median :33.00          FALL      :13
##  Mean   :23.50 Mean   :36.89          GSW       : 2
##  3rd Qu.:34.75 3rd Qu.:47.25          MCA      : 7
##  Max.   :46.00  Max.   :83.00          MVA      :10
##                                         Peds_vs_Auto: 6
##
##          field.gcs      er.gcs      icu.gcs      worst.gcs
##  Min.   : 3.000  Min.   :-7.960  Min.   :-1.610  Min.   :-4.339
##  1st Qu.: 3.250 1st Qu.: 4.000  1st Qu.: 3.000  1st Qu.: 3.000
##  Median : 7.000 Median : 7.000  Median : 6.000  Median : 3.000
##  Mean   : 8.001 Mean   : 7.746  Mean   : 6.204  Mean   : 5.188
##  3rd Qu.:12.000 3rd Qu.:12.000 3rd Qu.: 7.750  3rd Qu.: 7.000
##  Max.   :15.000 Max.   :15.000  Max.   :14.000  Max.   :14.000
##
##          X6m.gose      X2013.gose  skull.fx temp.injury surgery
##  Min.   :2.000  Min.   :2.000  0:18    0:19    0:17
##  1st Qu.:3.000 1st Qu.:5.000  1:28    1:27    1:29
##  Median :5.000 Median :7.000
##  Mean   :5.095 Mean   :5.804
##  3rd Qu.:6.997 3rd Qu.:7.000
##  Max.   :8.930 Max.   :8.000
##
##          spikes.hr      min.hr      max.hr      acute.sz late.sz
##  Min.   :-106.439 Min.   :-28.5276 Min.   :-536.00  0:38    0:20
##  1st Qu.:  4.577 1st Qu.:  0.0000 1st Qu.: 32.21  1: 8    1:26
##  Median : 29.593 Median :  0.0000 Median : 98.15
##  Mean   : 52.290 Mean   : -0.1032 Mean   : 197.81
##  3rd Qu.: 84.794 3rd Qu.:  0.1667 3rd Qu.:333.46
##  Max.   :294.000 Max.   : 42.0000 Max.   :1199.00
##
##          ever.sz missing_field.gcs missing_er.gcs missing_icu.gcs
##  0:19    Mode :Logical      Mode :Logical      Mode :Logical
##  1:27    FALSE:44          FALSE:44          FALSE:45
##  TRUE :2           TRUE :2           TRUE :1
##  NA's :0           NA's :0           NA's :0
##
##  missing_worst.gcs missing_X6m.gose missing_spikes.hr missing_min.hr
##  Mode :Logical      Mode :Logical      Mode :Logical      Mode :Logical
##  FALSE:45          FALSE:41          FALSE:28          FALSE:28
##  TRUE :1           TRUE :5           TRUE :18          TRUE :18

```

```

##  NA's :0          NA's :0          NA's :0          NA's :0
## 
##  missing_max.hr
##  Mode :Logical
##  FALSE:28
##  TRUE :18
##  NA's :0
## 
##  $`chain:5` 
##  id          age          sex          mechanism
##  Min.   : 1.00  Min.   :16.00  Female: 9  Bike_vs_Auto: 4
##  1st Qu.:12.25 1st Qu.:23.00  Male   :37   Blunt      : 4
##  Median :23.50 Median :33.00          Fall      :13
##  Mean   :23.50 Mean   :36.89          GSW       : 2
##  3rd Qu.:34.75 3rd Qu.:47.25          MCA       : 7
##  Max.   :46.00  Max.   :83.00          MVA       :10
##                                         Peds_vs_Auto: 6
##  field.gcs      er.gcs      icu.gcs      worst.gcs
##  Min.   : 3.000  Min.   :-15.73  Min.   : 0.000  Min.   :-2.742
##  1st Qu.: 3.250 1st Qu.: 4.00  1st Qu.: 3.000 1st Qu.: 3.000
##  Median : 7.000 Median : 7.32  Median : 6.000  Median : 3.000
##  Mean   : 8.473 Mean   : 7.65  Mean   : 6.439  Mean   : 5.223
##  3rd Qu.:12.750 3rd Qu.:12.00 3rd Qu.: 8.000 3rd Qu.: 7.000
##  Max.   :20.172 Max.   :15.00  Max.   :14.000  Max.   :14.000
## 
##  X6m.gose      X2013.gose  skull.fx temp.injury surgery
##  Min.   : 2.000  Min.   :2.000  0:18    0:19    0:17
##  1st Qu.: 3.000 1st Qu.:5.000 1:28    1:27    1:29
##  Median : 5.000 Median :7.000
##  Mean   : 4.972 Mean   :5.804
##  3rd Qu.: 6.000 3rd Qu.:7.000
##  Max.   :11.481 Max.   :8.000
## 
##  spikes.hr      min.hr      max.hr      acute.sz late.sz
##  Min.   :-74.552 Min.   :-11.877 Min.   :-570.4 0:38   0:20
##  1st Qu.:  5.518 1st Qu.: -1.924 1st Qu.: 37.5 1: 8   1:26
##  Median : 32.297 Median : 0.000  Median : 175.3
##  Mean   : 54.268 Mean   : 1.022  Mean   : 253.7
##  3rd Qu.: 71.288 3rd Qu.: 0.000 3rd Qu.: 432.1
##  Max.   :294.000 Max.   :42.000 Max.   :1199.0
## 
##  ever.sz missing_field.gcs missing_er.gcs missing_icu.gcs
##  0:19   Mode :Logical      Mode :Logical      Mode :Logical
##  1:27   FALSE:44          FALSE:44          FALSE:45
##  TRUE  :2               TRUE  :2           TRUE  :1
##  NA's :0               NA's :0           NA's :0
## 
##  missing_worst.gcs missing_X6m.gose missing_spikes.hr missing_min.hr
##  Mode :Logical      Mode :Logical      Mode :Logical      Mode :Logical
##  FALSE:45          FALSE:41          FALSE:28          FALSE:28
##  TRUE  :1           TRUE  :5           TRUE  :18          TRUE  :18
##  NA's :0           NA's :0           NA's :0           NA's :0
## 
##  missing_max.hr
##  Mode :Logical
##  FALSE:28
##  TRUE :18
##  NA's :0

```

7. Cast the imputed numbers as integers.

```
# (not necessary, but may be useful)
indx <- sapply(data.frames[[5]], is.numeric) # get the indices of
# numeric columns
data.frames[[5]][indx] <- lapply(data.frames[[5]][indx], function(x)
  as.numeric(as.integer(x)))
# cast each value as integer data.frames[[5]]$spikes.hr
```

8. Save results out.

```
write.csv(data.frames[[5]], "C:\\Users\\User\\Desktop\\TBI_MIData.csv")
```

9. Complete Data analytics functions.

```
# library("mi")
# lm.mi(); glm.mi(); polr.mi(); bayesglm.mi(); bayespolr.mi(); lmer.mi(); gl
mer.mi()
```

10. Fit a linear model for one multiply imputed chain.

```
# Also see Step (9)
##Linear regression for each imputed data set - 5 regression models are fit
fit_lm1 <- glm(ever.sz ~ surgery + worst.gcs + factor(sex) + age, data.frame
s$`chain:1`, family = "binomial"); summary(fit_lm1); display(fit_lm1)

## Call:
## glm(formula = ever.sz ~ surgery + worst.gcs + factor(sex) + age,
##      family = "binomial", data = data.frames$`chain:1`)
## 

## Deviance Residuals:
##      Min        1Q     Median        3Q       Max
## -1.7000  -1.2166   0.8222   1.0007   1.3871
## 

## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)          0.249780  1.356397  0.184   0.854
## surgery1            0.947392  0.685196  1.383   0.167
## worst.gcs          -0.068734  0.097962 -0.702   0.483
## factor(sex)Male   -0.329313  0.842761 -0.391   0.696
## age                 0.004453  0.019431  0.229   0.819
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
## Null deviance: 62.371 on 45 degrees of freedom
## Residual deviance: 60.046 on 41 degrees of freedom
## AIC: 70.046
## 
## Number of Fisher Scoring iterations: 4
```

```

## glm(formula = ever.sz ~ surgery + worst.gcs + factor(sex) + age,
##   family = "binomial", data = data.frames$`chain:1`)
##   coef.est coef.se
## (Intercept) 0.25 1.36
## surgery1 0.95 0.69
## worst.gcs -0.07 0.10
## factor(sex)Male -0.33 0.84
## age 0.00 0.02
## ---
## n = 46, k = 5
## residual deviance = 60.0, null deviance = 62.4 (difference = 2.3)

```

11. Fit the appropriate model and pool the results.

```

# (estimates over MI chains)
model_results <- pool(ever.sz ~ surgery + worst.gcs + factor(sex) + age,
family = "binomial", data=imputations, m=5)
display (model_results); summary (model_results)

## bayesglm(formula = ever.sz ~ surgery + worst.gcs + factor(sex) +
##   age, data = imputations, m = 5, family = "binomial")
##   coef.est coef.se
## (Intercept) 0.46 1.34
## surgery1 0.94 0.66
## worst.gcs -0.09 0.10
## factor(sex)Male -0.33 0.77
## age 0.00 0.02
## n = 41, k = 5
## residual deviance = 59.3, null deviance = 62.4 (difference = 3.0)

##
## Call:
## pool(formula = ever.sz ~ surgery + worst.gcs + factor(sex) +
##   age, data = imputations, m = 5, family = "binomial")
## 

## Deviance Residuals:
##   Min     1Q Median     3Q    Max 
## -1.6796 -1.1802  0.8405  1.0225  1.3824 
## 

## Coefficients:
##   Estimate Std. Error z value Pr(>|z|)    
## (Intercept) 0.459917  1.344700  0.342   0.732    
## surgery1   0.938109  0.661646  1.418   0.156    
## worst.gcs  -0.089340  0.098293 -0.909   0.363    
## factor(sex)Male -0.332875  0.770327 -0.432   0.666    
## age        0.001582  0.019685  0.080   0.936    
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
## Null deviance: 62.371 on 45 degrees of freedom
## Residual deviance: 59.343 on 41 degrees of freedom
## AIC: 69.343
## 
## Number of Fisher Scoring iterations: 6.6

```

12. Report the summaries of the imputations.

```

data.frames <- complete(imputations, 3)      # extract the first 3 chains
lapply(data.frames, summary)
## $`chain:1`  

##   id      age      sex      mechanism  

##   Min. :1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4  

##   1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4  

##   Median :23.50 Median :33.00          Fall      :13  

##   Mean   :23.50  Mean  :36.89          GSW       : 2  

##   3rd Qu.:34.75 3rd Qu.:47.25          MCA       : 7  

##   Max.  :46.00  Max.  :83.00          MVA       :10  

##                                         Peds_vs_Auto: 6  

...  

##   missing_max.hr  

##   Mode :Logical  

##   FALSE:28  

##   TRUE :18  

##   NA's :0  

## $`chain:2`  

##   id      age      sex      mechanism  

##   Min. :1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4  

##   1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4  

##   Median :23.50 Median :33.00          Fall      :13  

##   Mean   :23.50  Mean  :36.89          GSW       : 2  

##   3rd Qu.:34.75 3rd Qu.:47.25          MCA       : 7  

##   Max.  :46.00  Max.  :83.00          MVA       :10  

##                                         Peds_vs_Auto: 6  

...  

##   missing_max.hr  

##   Mode :Logical  

##   FALSE:28  

##   TRUE :18  

##   NA's :0  

##  

## $`chain:3`  

##   id      age      sex      mechanism  

##   Min. :1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4  

##   1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4  

##   Median :23.50 Median :33.00          Fall      :13  

##   Mean   :23.50  Mean  :36.89          GSW       : 2  

##   3rd Qu.:34.75 3rd Qu.:47.25          MCA       : 7  

##   Max.  :46.00  Max.  :83.00          MVA       :10  

##                                         Peds_vs_Auto: 6  

...  

##   missing_max.hr  

##   Mode :Logical  

##   FALSE:28  

##   TRUE :18  

##   NA's :0

```

13. Validation:

Next, we can verify whether enough iterations were conducted. One validation criteria demands that the mean of each completed variable is similar to the corresponding mean of the complete data (Fig. 3.24).

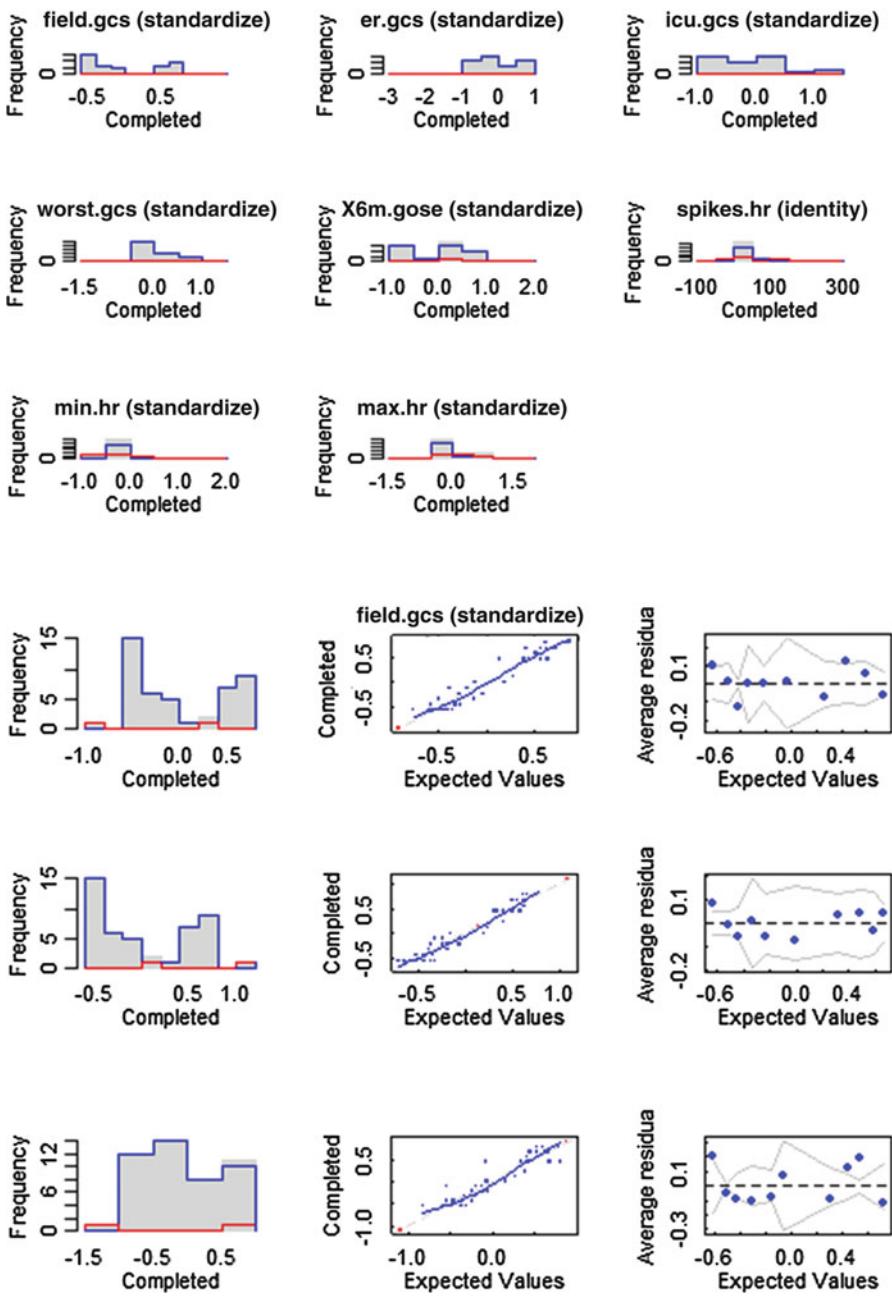


Fig. 3.24 TBI data imputation quality plots

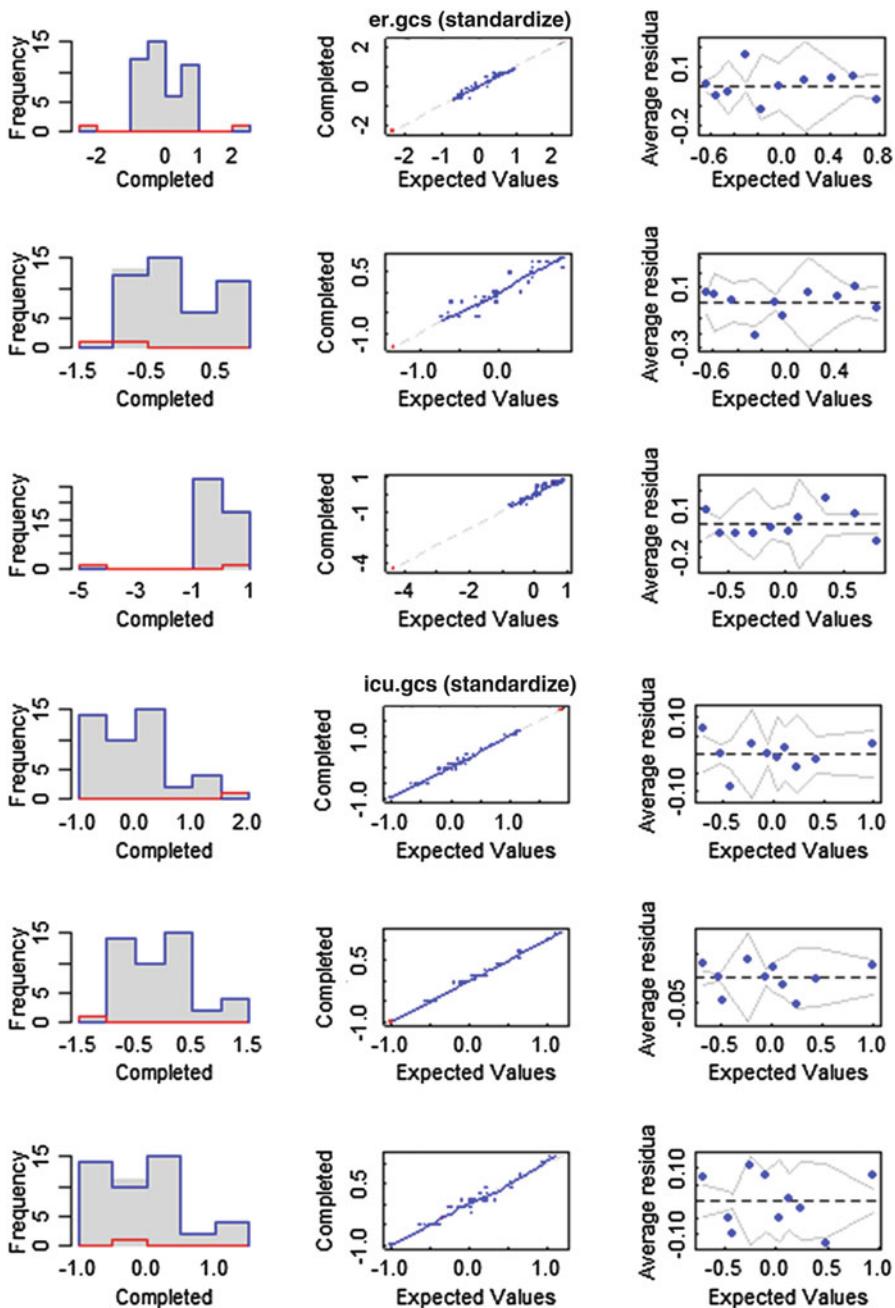


Fig. 3.24 (continued)

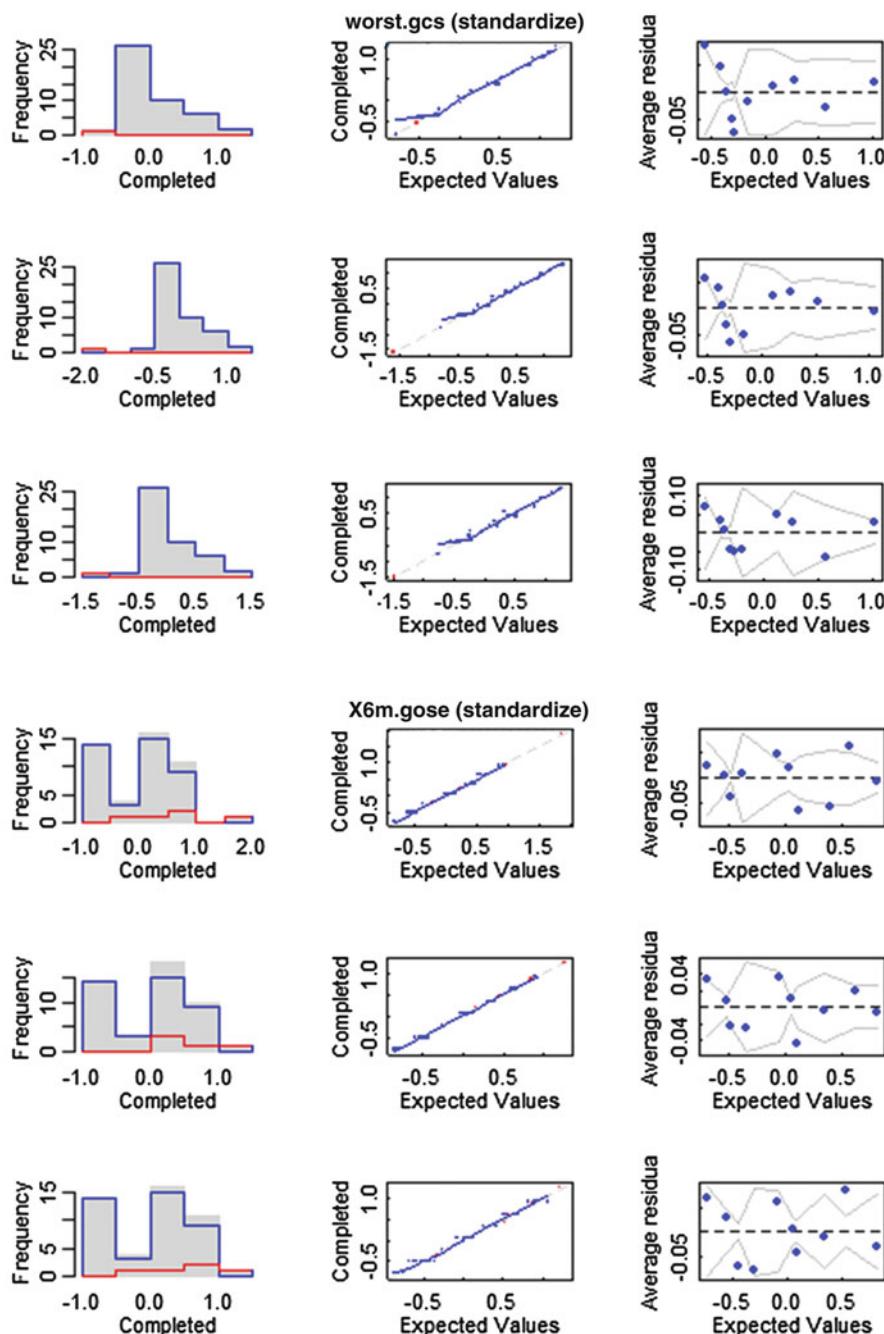


Fig. 3.24 (continued)

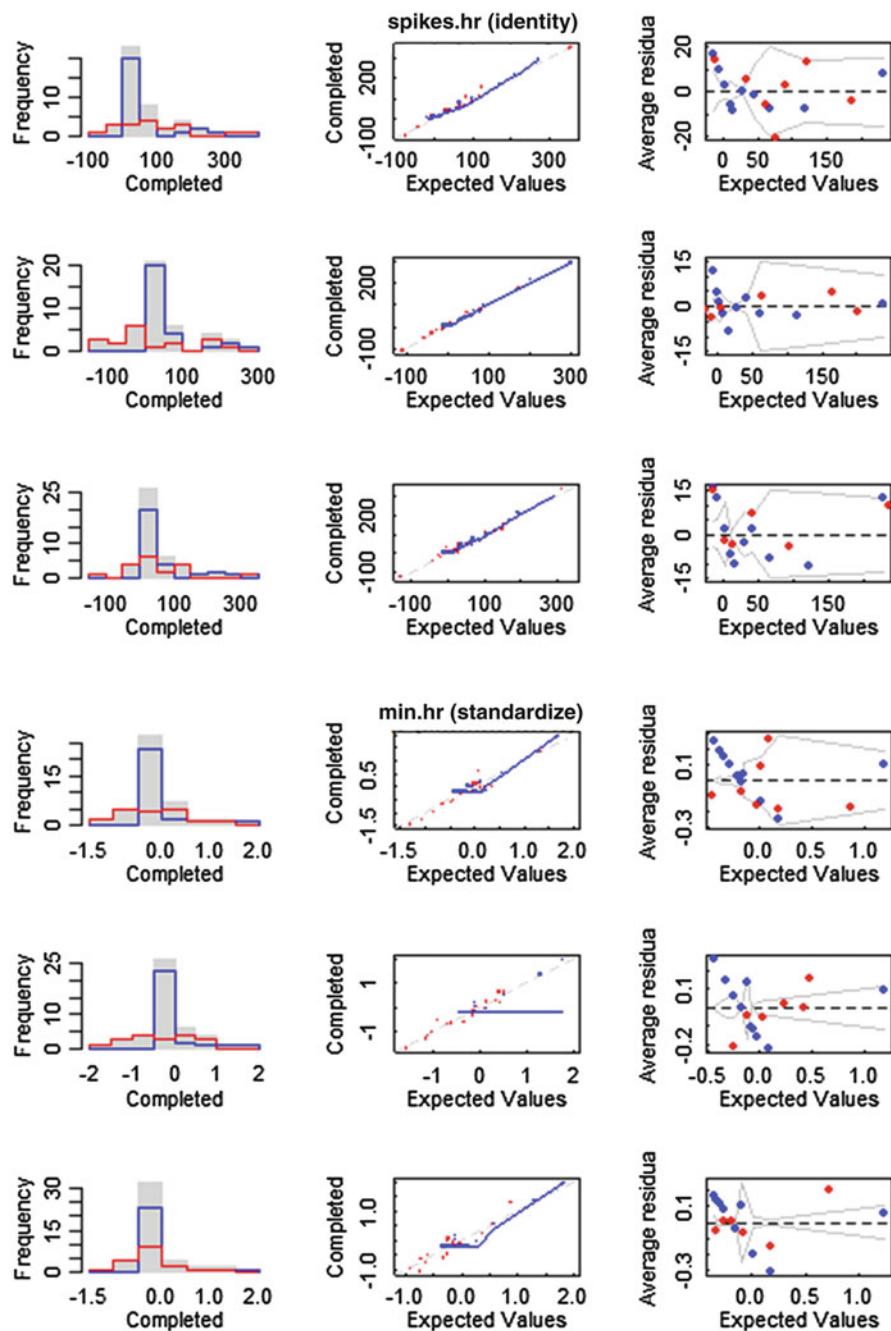


Fig. 3.24 (continued)

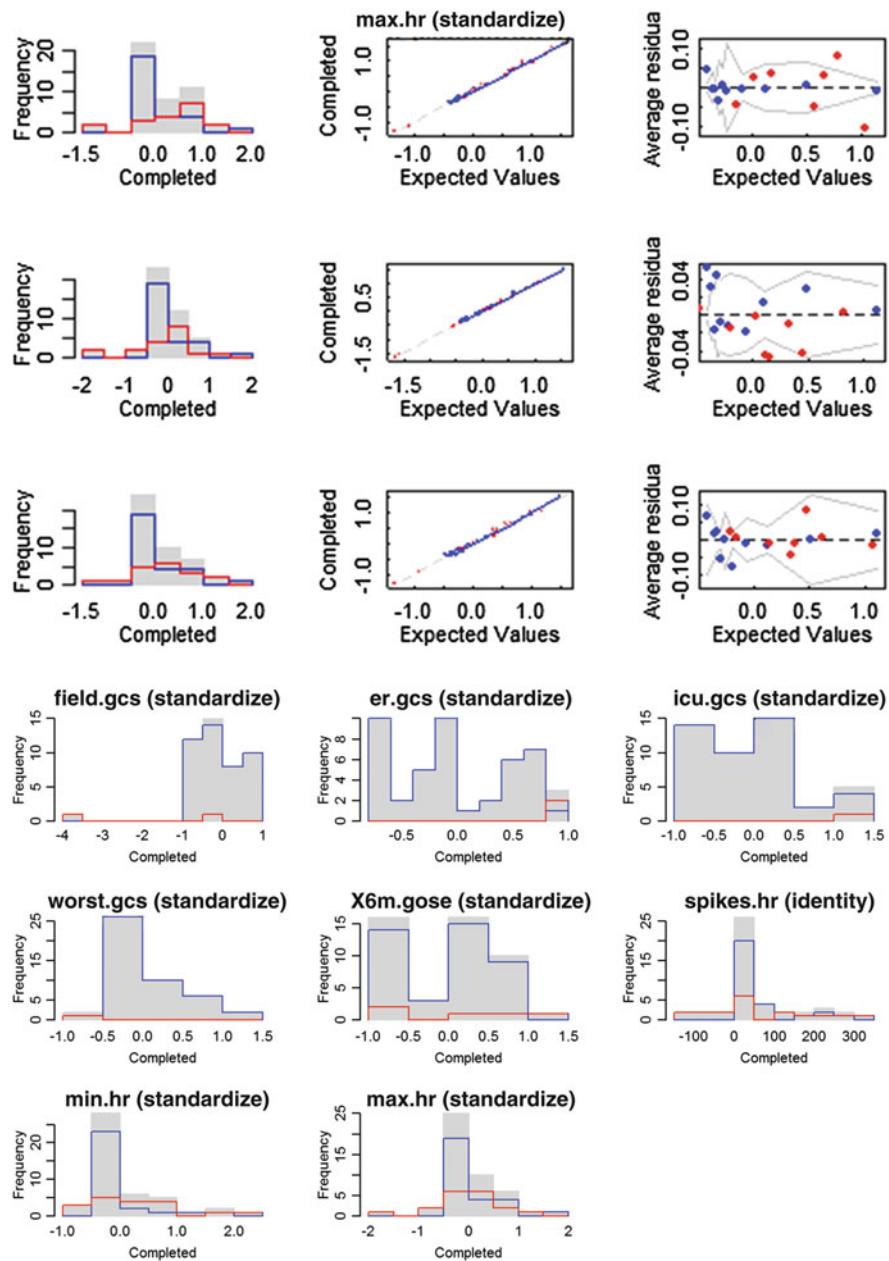


Fig. 3.24 (continued)

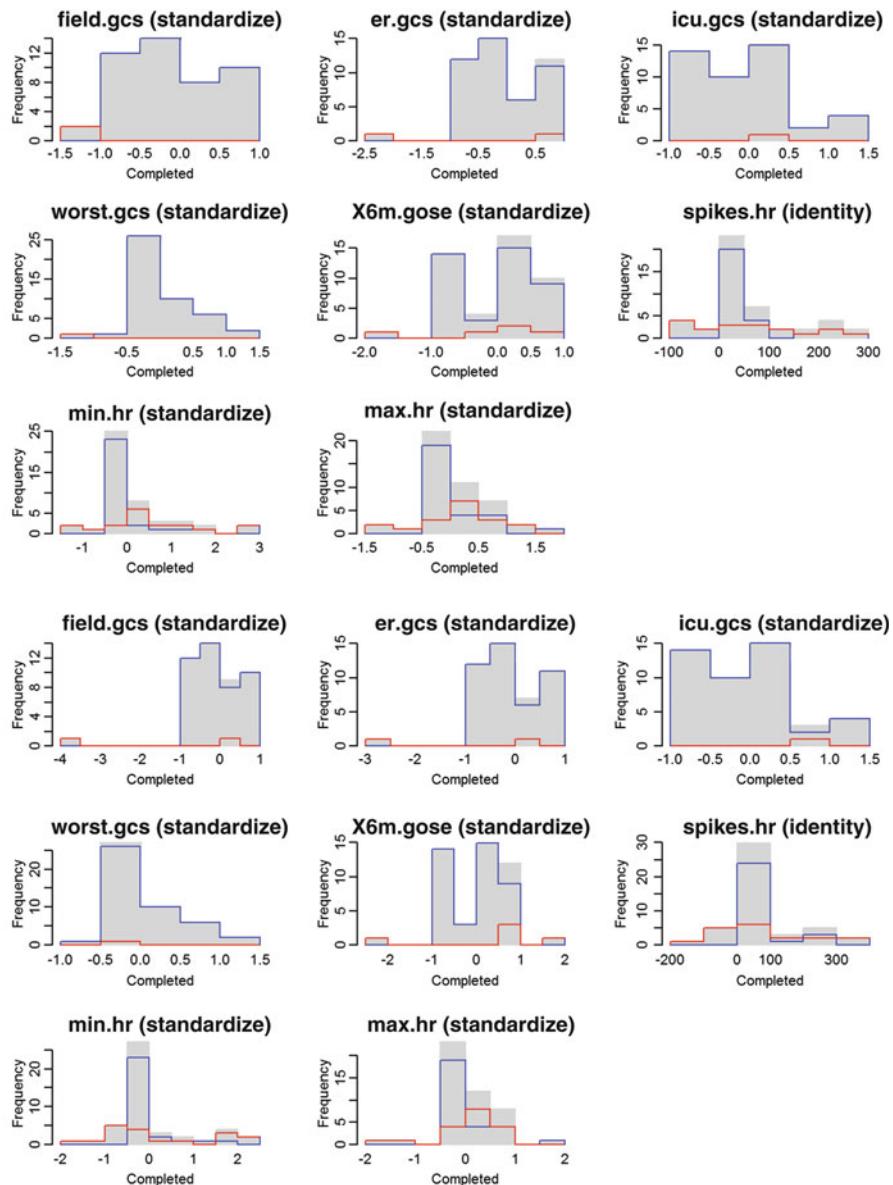


Fig. 3.24 (continued)

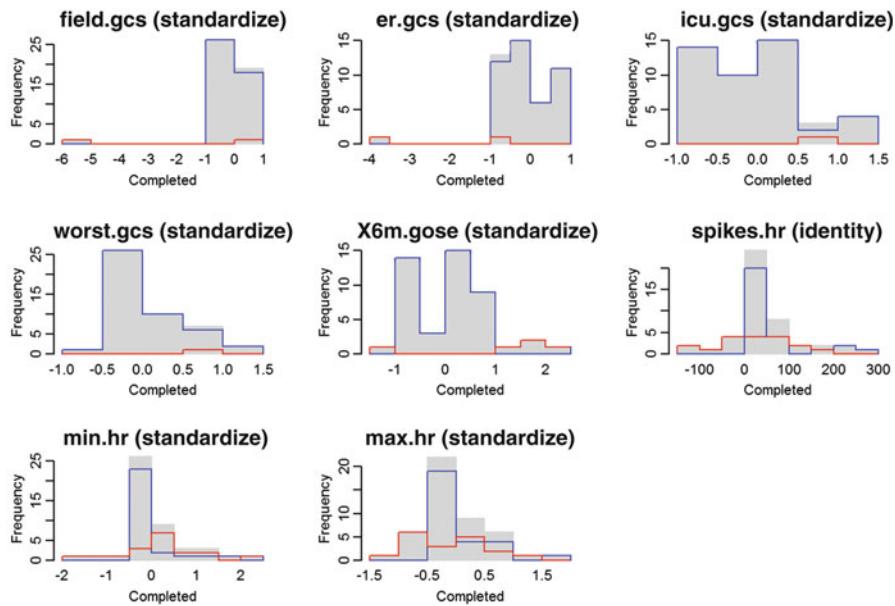


Fig. 3.24 (continued)

```

# should be similar for each of the k chains (in this case k=5).
# mippy is wrapper for sapply invoked on mi-class objects to

# compute the col means
round(mippy(imputations, mean, to.matrix = TRUE), 3)

##                               chain:1 chain:2 chain:3 chain:4 chain:5
## id                         23.500  23.500  23.500  23.500  23.500
## age                        0.000   0.000   0.000   0.000   0.000
## sex                        1.804   1.804   1.804   1.804   1.804
..
## missing_max.hr             0.391   0.391   0.391   0.391   0.391

# Rhat convergence statistics compares the variance between chains to the variance
# across chains.
# Rhat Values ~ 1.0 indicate likely convergence,
# Rhat Values > 1.1 indicate that the chains should be run longer
# (use large number of iterations)
Rhats(imputations, statistic = "moments") # assess the convergence of MI algorithm

## mean_field.gcs  mean_er.gcs  mean_icu.gcs  mean_worst.gcs  mean_X6m.gose
## 1.858663      2.200902     1.484120     2.360286      1.752000
## mean_spikes.hr mean_min.hr   mean_max.hr   sd_field.gcs   sd_er.gcs
## 1.972090      1.393025     1.743846     1.291126      1.479967
## sd_icu.gcs    sd_worst.gcs  sd_X6m.gose  sd_spikes.hr   sd_min.hr
## 1.417884      1.861408     1.355365     1.514089      1.723325
## sd_max.hr     sd_mean.hr
## 1.497625

```

```
# When convergence is unstable, we can continue the iterations for
# all chains, e.g.
imputations <- mi(imputations, n.iter=20) # add additional 20 iterations

# To plot the produced mi results, for all missing_variables we can generate
# a histogram of the observed, imputed, and completed data.
# We can compare of the completed data to the fitted values

# implied by the model for the completed data, by plotting binned residuals.
# hist function works similarly as plot.
# image function gives a sense of the missingness patterns in the data

plot(imputations); hist(imputations)

image(imputations); summary(imputations)
```

```
## $id
## $id$is_missing
## [1] "all values observed"
##
## $id$observed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.00 12.25 23.50 23.50 34.75 46.00
##
## 
## $age
## $age$is_missing
## [1] "all values observed"
##
## $age$observed
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## -0.6045 -0.4019 -0.1126 0.0000 0.2997 1.3340
##
## 
## $sex
## $sex$is_missing
## [1] "all values observed"
##
## $sex$observed
##
## 1 2
## 9 37
...
## $Late.sz$observed
##
## 1 2
## 20 26
## $ever.sz
## $ever.sz$is_missing
## [1] "all values observed"
##
## $ever.sz$observed
##
## 1 2
## 19 27
```

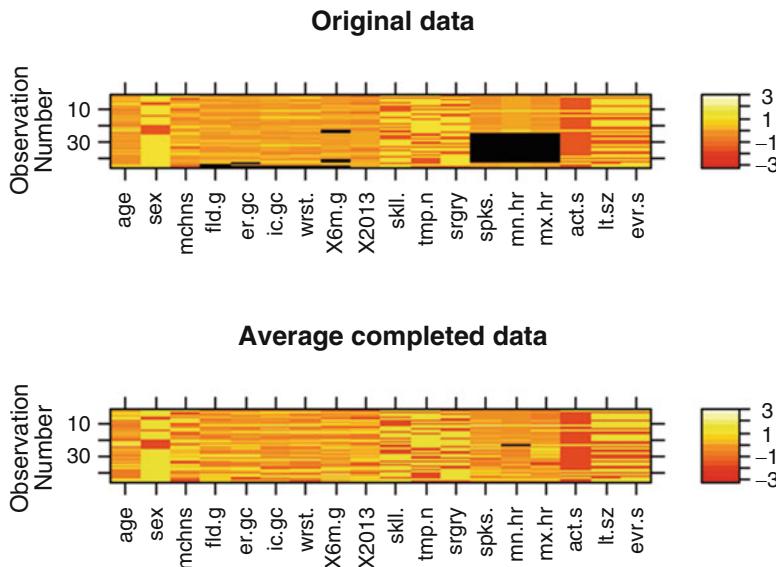


Fig. 3.25 Comparison of the missing data patterns in the original (top) and the completed (bottom) TBI sets

14. Pool over them $m = 5$ imputed chains when fitting the "linear model". We can pool from across the five chains in order to estimate the final linear model (Fig. 3.25).

```
# regression model and impact of various predictors
model_results <- pool(ever.sz ~ surgery + worst.gcs + factor(sex) + age,
data = imputations, m=5); display (model_results); summary (model_results)

## bayesglm(formula = ever.sz ~ surgery + worst.gcs + factor(sex) +
##           age, data = imputations, m = 5)
##           coef.est coef.se
## (Intercept)  0.58     1.35
## surgery1    0.99     0.66
## worst.gcs   -0.11    0.10
## factor(sex)Male -0.36   0.77
## age         0.00     0.02
## n = 41, k = 5
## residual deviance = 59.0, null deviance = 62.4 (difference = 3.4)

## Call:
## pool(formula = ever.sz ~ surgery + worst.gcs + factor(sex) +
##       age, data = imputations, m = 5)
## 
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
## -1.6927 -1.1539  0.8245  1.0052  1.4009
## 
```

```

## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)          0.578917  1.348831  0.429   0.668
## surgery1            0.990656  0.662991  1.494   0.135
## worst.gcs          -0.105240  0.095335 -1.104   0.270
## factor(sex)Male   -0.357285  0.772307 -0.463   0.644
## age                 0.000198  0.019702  0.010   0.992
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 62.371 on 45 degrees of freedom
## Residual deviance: 58.995 on 41 degrees of freedom
## AIC: 68.995
##
## Number of Fisher Scoring iterations: 7

coef(summary(model_results))[, 1:2] # get the model coef's and their SE's

##                               Estimate Std. Error
## (Intercept)          0.5789166170 1.34883106
## surgery1            0.9906554934 0.66299111
## worst.gcs          -0.1052399155 0.09533513
## factor(sex)Male   -0.3572845034 0.77230674
## age                 0.0001980042 0.01970153

# Report the summaries of the imputations
data.frames <- complete(imputations, 3)      # extract the first 3 chains
Lapply(data.frames, summary)                  # report summaries

## [[1]]
##      id          age         sex      mechanism
## Min. : 1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4
## 1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4
## Median :23.50 Median :33.00          Fall      :13
## Mean   :23.50 Mean   :36.89          GSW       : 2
## 3rd Qu.:34.75 3rd Qu.:47.25          MCA      : 7
## Max.   :46.00 Max.   :83.00          MVA      :10
##                                         Peds_vs_Auto: 6
...
## missing_max.hr
## Mode :Logical
## FALSE:28
## TRUE :18
## NA's :0

## [[2]]
##      id          age         sex      mechanism
## Min. : 1.00  Min. :16.00  Female: 9  Bike_vs_Auto: 4
## 1st Qu.:12.25 1st Qu.:23.00  Male :37   Blunt      : 4
## Median :23.50 Median :33.00          Fall      :13
## Mean   :23.50 Mean   :36.89          GSW       : 2
## 3rd Qu.:34.75 3rd Qu.:47.25          MCA      : 7
## Max.   :46.00 Max.   :83.00          MVA      :10
##                                         Peds_vs_Auto: 6
...
## missing_max.hr
## Mode :Logical

```

```

## FALSE:28
## TRUE :18
## NA's :0
##
## [[3]]
##      id          age         sex      mechanism
##  Min.   : 1.00   Min.   :16.00   Female: 9   Bike_vs_Auto: 4
##  1st Qu.:12.25  1st Qu.:23.00  Male   :37    Blunt      : 4
##  Median :23.50  Median :33.00          Fall     :13
##  Mean   :23.50  Mean   :36.89          GSW      : 2
##  3rd Qu.:34.75  3rd Qu.:47.25          MCA      : 7
##  Max.   :46.00  Max.   :83.00          MVA      :10
##                                         Peds_vs_Auto: 6
...
##
## missing_max.hr
## Mode :Logical
## FALSE:28
## TRUE :18
## NA's :0

```

3.13.3 *Imputation via Expectation-Maximization*

Below we present the theory and practice of one specific statistical computing strategy for imputing incomplete datasets.

Types of Missing Data

Recall that we have the following three distinct types of incomplete data.

- **MCAR:** Data which is Missing Completely At Random has nothing systematic about which observations are missing. There is no relationship between missingness and either observed or unobserved covariates.
- **MAR:** Missing At Random is weaker than MCAR. The missingness is still random, but solely due to the observed variables. For example, those from a lower socioeconomic status (SES) may be less willing to provide salary information (but we know their SES). The key is that the missingness is not due to the values which are not observed. MCAR implies MAR, but not vice-versa.
- **MNAR:** If the data are Missing Not At Random, then the missingness depends on the values of the missing data. Examples include censored data, self-reported data for individuals who are heavier, who are less likely to report their weight, and response-measuring device that can only measure values above 0.5, anything below that is missing.

General Idea of EM Algorithm

Expectation-Maximization (EM) is an iterative parameter estimation process involving two steps, *expectation* and *maximization*, which are applied in tandem. EM can be employed to find parameter estimates using maximum likelihood and is specifically

useful when the equations determining the relations of the data-parameters cannot be directly solved. For example, a Gaussian mixture modeling assumes that each data point (X) has a corresponding latent (unobserved) variable or a missing value (Y), which may be specified as a mixture of coefficients determining the affinity of the data as a linear combination of Gaussian kernels, determined by a set of parameters (θ), e.g., means and variance-covariances. Thus, EM estimation relies on:

- An observed data set X ,
- A set of missing (or latent) values Y ,
- A parameter θ , which may be a vector of parameters,
- A likelihood function $L(\theta|X, Y) = p(X, Y|\theta)$, and
- The maximum likelihood estimate (MLE) of the unknown parameter(s) θ that is computed using the marginal likelihood of the observed data:

$$L(\theta|X) = p(X|\theta) = \int p(X, Y|\theta) dY.$$

Most of the time, this equation may not be directly solved, e.g., when Y is missing.

- *Expectation step (E step):* computes the expected value of the *log likelihood function*, with respect to the conditional distribution of Z given X using the parameter estimates at the previous iteration (or at the position of initialization, for the first iteration), θ_t :

$$Q(\theta|\theta^{(t)}) = E_{Y|X, \theta^{(t)}} [\log(L(\theta|X, Y))];$$

- *Maximization step (M step):* Determine the parameters, θ , that maximize the expectation above,

$$\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^{(t)}).$$

EM-Based Imputation

The EM algorithm is an alternative to Newton-Raphson, or the method of scoring, for computing MLE in cases where the complications in calculating the MLE are due to incomplete observation and data are MAR, missing at random, with separate parameters for observation and the missing data mechanism, so the missing data mechanism can be ignored.

$$\textbf{Complete Data: } Z = \begin{pmatrix} X \\ Y \end{pmatrix}, ZZ^T = \begin{pmatrix} XX^T & XY^T \\ YX^T & YY^T \end{pmatrix},$$

where X is the observed data and Y is the missing data.

- E-step: (Expectation) Get the expectations of Y and YY^T based on observed data.
- M-step: (Maximization) Maximize the conditional expectation in E-step to estimate the parameters.

Details: If $o = obs$ and $m = mis$ stand for observed and missing, the mean vector, $(\mu_{obs}, \mu_{mis})^T$, and the variance-covariance matrix, $\Sigma^{(t)} = \begin{pmatrix} \Sigma_{oo} & \Sigma_{om} \\ \Sigma_{mo} & \Sigma_{mm} \end{pmatrix}$, are represented by:

$$\mu^{(t)} = \begin{pmatrix} \mu_{obs} \\ \mu_{mis} \end{pmatrix}, \quad \Sigma^{(t)} = \begin{pmatrix} \Sigma_{oo} & \Sigma_{om} \\ \Sigma_{mo} & \Sigma_{mm} \end{pmatrix}$$

E-step:

$$E(Z|X) = \begin{pmatrix} X \\ E(Y|X) \end{pmatrix}, \quad E(ZZ^T|X) = \begin{pmatrix} XX^T & XE(Y|X)^T \\ E(Y|X)X^T & E(YY^T|X) \end{pmatrix}.$$

$$E(Y|X) = \mu_{mis} + \Sigma_{mo}\Sigma_{oo}^{-1}(X - \mu_{obs}).$$

$$E(YY^T|X) = (\Sigma_{mm} - \Sigma_{mo}\Sigma_{oo}^{-1}\Sigma_{om}) + E(Y|X)E(Y|X)^T.$$

M-step:

$$\mu^{(t+1)} = \frac{1}{n} \sum_{i=1}^n E(Z|X) \text{ and } \Sigma^{(t+1)} = \frac{1}{n} \sum_{i=1}^n E(ZZ^T|X) - \mu^{(t+1)}\mu^{(t+1)T}.$$

A Simple Manual Implementation of EM-Based Imputation

To demonstrate the implementation of an EM-based imputation method from first principles, let's simulate 20 (feature) vectors of 200 (cases) Normal Distributed random values.

```
set.seed(202227)
mu <- as.matrix(rep(2, 20))
sig <- diag(c(1:20))
# Add a noise item. The noise is
#   $ \epsilon \sim MVN(as.matrix(rep(0,20)), diag(rep(1,20)))
sim_data <- mvrnorm(n = 200, mu, sig) +
  mvrnorm(n=200, as.matrix(rep(0,20)), diag( rep(1,20) ))

# save these in the "original" object
sim_data.orig <- sim_data

# introduce 500 random missing indices (in the total of 4000=200*20)
# discrete distribution where the probability of the elements of values is proportional to probs, which are normalized to add up to 1.
rand.miss <- e1071::rdiscrete(500, probs = rep(1, length(sim_data)), values =
  seq(1, length(sim_data)))
sim_data[rand.miss] <- NA
sum(is.na(sim_data)) # check now many missing (NA) are there < 500
## [1] 459

# cast the data into a data.frame object and report 15*10 elements
sim_data.df <- data.frame(sim_data)
kable(sim_data.df[1:15, 1:10], caption = "The first 15 rows and first 10 columns of the simulation data")
```

The first 15 rows and first 10 columns of the simulation data are included below, mind the missing values, Table 3.1.

Now, let's define the EM imputation method function:

Table 3.1 Excerpt of 15 rows and 10 columns of the obfuscated simulation data

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
0.1710559	4.8500284	1.7282025	NA	4.5511409	2.1699719	4.7118571	2.6972905	NA	-0.9250101
1.7721898	-1.6955535	0.4234295	6.2619764	-3.1339307	-1.8043889	1.2932913	0.9906410	1.0790366	1.1638161
2.2859237	4.2713494	3.8900670	3.5104578	3.0025607	3.9058619	3.0548309	NA	2.7208548	1.3601231
1.4906345	6.9253703	1.3387561	-2.6177675	-1.4074433	-2.0861790	-0.8847125	5.6485105	1.3348689	0.6512754
-0.0305062	2.3069337	NA	1.2474398	-0.2091862	1.1726298	1.4622491	0.6492390	0.1458781	7.2195026
1.9411674	3.6606019	6.3004943	0.35453817	1.0875460	1.3077307	0.0269935	-0.0112352	2.9724355	2.4616466
0.5209321	4.2225983	-0.6328796	NA	5.7103015	NA	4.5803136	0.2535158	-0.9758147	NA
3.3111727	0.1245427	0.9768425	4.77644851	4.9810860	-0.0438375	-0.3213523	4.3612903	2.7483497	3.7744105
2.1097813	0.2427543	1.1718823	3.6791032	3.7784795	5.5533794	1.8635898	0.1565858	4.6743012	-1.0232277
1.6203382	NA	1.3625543	1.4006816	-1.3467049	6.1636769	3.1703070	-3.4308300	0.2195447	-2.2921107
NA	0.5601779	NA	2.1652769	0.1060700	2.3763668	-3.7629196	-0.9718406	1.2196606	0.7038253
1.8570634	1.6714614	0.2000255	2.3308968	0.4593543	0.1716613	-0.5795159	-1.3327330	NA	1.2607465
2.5416914	1.0388186	4.7421120	2.3110187	NA	-1.5291668	2.4574501	NA	2.5364297	NA
0.9393925	1.8668513	3.4702117	1.6797620	0.7797323	2.5548284	-3.5498813	3.6595296	3.2124694	5.3120133
-0.8334973	NA	1.1949414	-0.6573305	1.7484169	-1.7743495	1.7168340	2.1065878	2.9478528	-0.5562349

```

EM_algorithm <- function(x, tol = 0.001) {
  # identify the missing data entries (Boolean indices)
  missvals <- is.na(x)
  # initialize the EM-iteration
  new.impute <- x
  old.impute <- x
  count.iter <- 1
  reach.tol <- 0

  # compute \Sigma on complete data
  sigma <- as.matrix(var(na.exclude(x)))
  # compute the vector of feature (column) means
  mean.vec <- as.matrix(apply(na.exclude(x), 2, mean))

  while (reach.tol != 1) {
    for (i in 1:nrow(x)) {
      pick.miss <- (c(missvals[i, ]))
      if (sum(pick.miss) != 0) {

        # compute inverse-Sigma_completeData, variance-covariance matrix
        inv.S <- solve(sigma[!pick.miss, !pick.miss], tol = 1e-40)

        # Expectation Step
        #  $E(Y|X) = \mu_{mis} + \Sigma_{mo} \Sigma_{oo}^{-1} (X - \mu_{obs})$ 
        new.impute[i, pick.miss] <- mean.vec[pick.miss] +
          sigma[pick.miss, !pick.miss] %*% inv.S %*%
          (t(new.impute[i, !pick.miss]) - t(t(mean.vec[!pick.miss])))
      }
    }

    # Maximization Step
    # Compute the complete \Sigma complete vector of feature (column) means
    #  $\Sigma_{(t+1)} = \frac{1}{n} \sum_{i=1}^n E(ZZ^T|X) - \mu_{(t+1)} \mu_{(t+1)}^T$ 
    sigma <- var((new.impute))
    #  $\mu_{(t+1)} = \frac{1}{n} \sum_{i=1}^n E(Z|X)$ 
    mean.vec <- as.matrix(apply(new.impute, 2, mean))

    # Inspect for convergence tolerance, start with the 2nd iteration
    if (count.iter > 1) {
      for (l in 1:nrow(new.impute)) {
        for (m in 1:ncol(new.impute)) {
          if (abs((old.impute[l, m] - new.impute[l, m])) > tol) {
            reach.tol <- 0
          } else {
            reach.tol <- 1
          }
        }
      }
      count.iter <- count.iter + 1
      old.impute <- new.impute
    }
  }

  # return the imputation output of the current iteration that passed the
  # tolerance level
  return(new.impute)
}

sim_data.imputed <- EM_algorithm(sim_data.df, tol=0.0001)

```

Plotting Complete and Imputed Data

Smaller black colored points represent observed data, and magenta-color and circle-shapes denote the imputed data (Fig. 3.26).

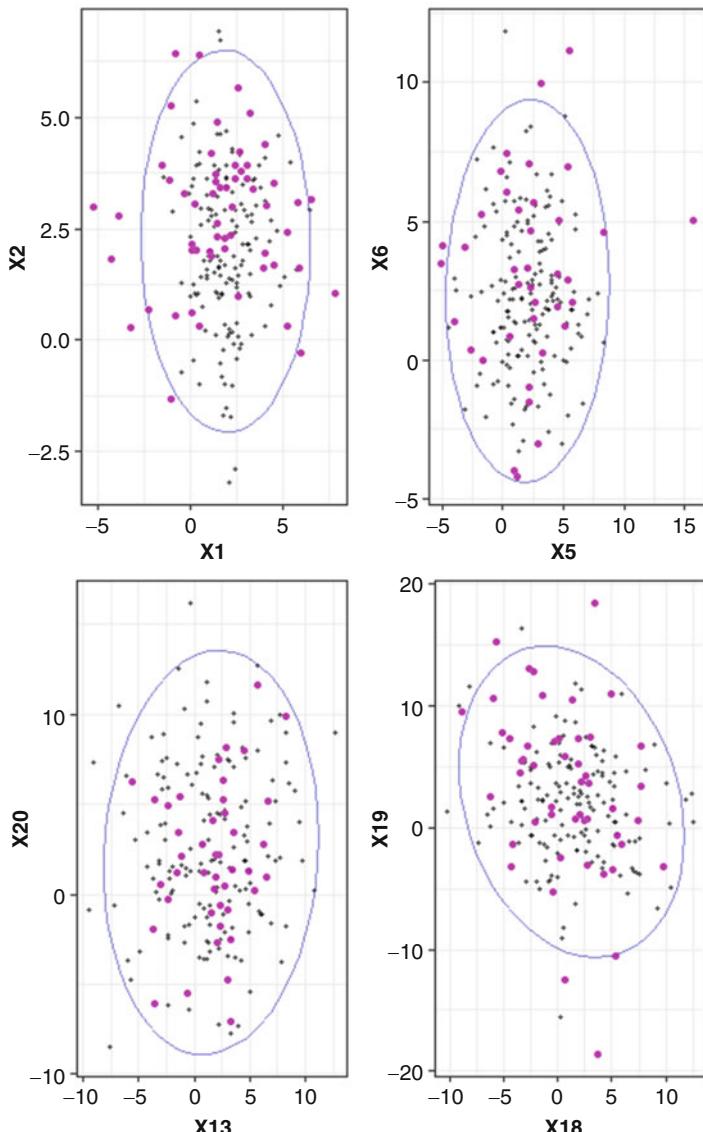


Fig. 3.26 Four scatterplots for pairs of features illustrating the complete data (small-black points), the imputed data points (larger-pink points), and 2D Gaussian kernels

```

plot.me <- function(index1, index2){
  plot.imputed <- sim_data.imputed[,row.names(
    subset(sim_data.df, is.na(sim_data.df[, index1]) |
    is.na(sim_data.df[, index2]))]
  p = ggplot(sim_data.imputed, aes_string( paste0("X",index1) ,
  paste0("X",index2))) +
  geom_point(alpha = 0.5, size = 0.7)+theme_bw() +
  stat_ellipse(type = "norm", color = "#000099", alpha=0.5) +
  geom_point(data = plot.imputed, aes_string( paste0("X",index1) ,
  paste0("X", (index2))),size = 1.5, color = "Magenta", alpha = 0.8)
}

gridExtra::grid.arrange( plot.me(1,2), plot.me(5,6), plot.me(13,20),
plot.me(18,19), nrow = 2)

```

Validation of EM-Imputation Using the **Amelia** R Package

See this Amelia paper (https://gking.harvard.edu/files/gking/files/amelia_jss.pdf) and the corresponding R manual.

Comparison

Let's use the **amelia** function to impute the original data *sim_data_df* and compare the results to the simpler manual **EM_algorithm** imputation defined above.

```

# install.packages("Amelia")
library(Amelia)

dim(sim_data.df)
## [1] 200 20

amelia.out <- amelia(sim_data.df, m = 5)

## -- Imputation 1 --
##  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## -- Imputation 2 --
##  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
## -- Imputation 3 --
##  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## -- Imputation 4 --
##  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 21 22 23 24
## -- Imputation 5 --
##  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

amelia.out

## 
## Amelia output with 5 imputed datasets.
## Return code: 1
## Message:  Normal EM convergence.
##
## Chain Lengths:
## -----
## Imputation 1: 20
## Imputation 2: 15
## Imputation 3: 16
## Imputation 4: 24
## Imputation 5: 17

amelia.imputed.5 <- amelia.out$imputations[[5]]

```

- Magenta-color and circle-shape denote manual imputation via EM_algorithm
- Orange-color and square-shapes denote Amelia imputation (Figs. 3.27 and 3.28).

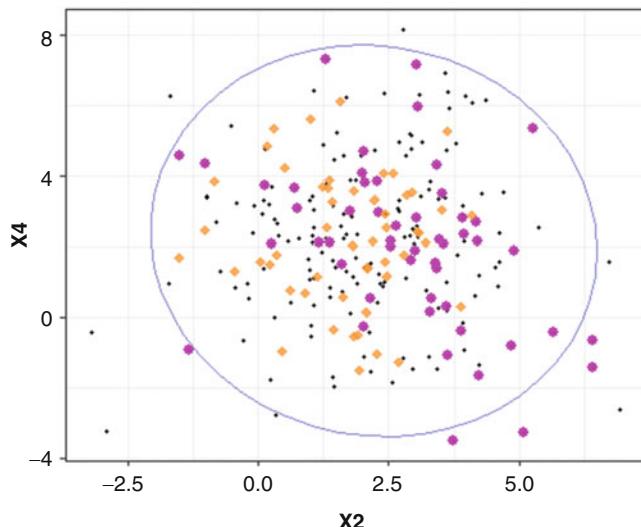


Fig. 3.27 Scatter plot of the second and fourth features. Magenta-circles and Orange-squares represent the manual imputation via EM_algorithm and the automated Amelia-based imputation

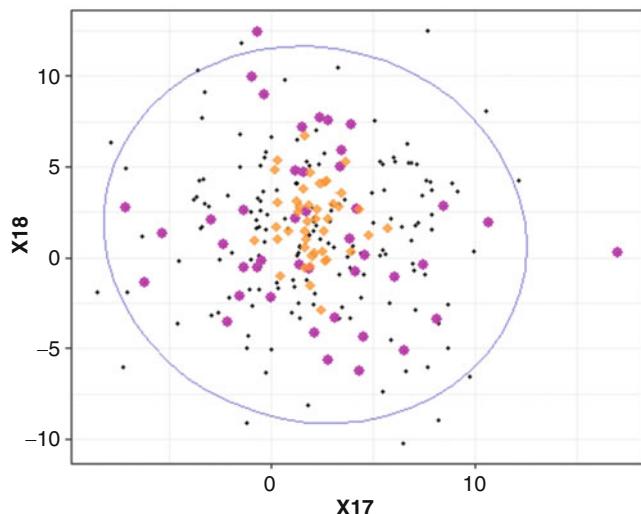


Fig. 3.28 Same as Fig. 3.27, for features 17 and 18

```

plot.me2 <- function(index1, index2){
  plot.imputed <- sim_data.imputed[row.names(  

    subset(sim_data.df, is.na(sim_data.df[, index1]) |  

    is.na(sim_data.df[, index2])), ]  

  plot.imputed2 <- amelia.imputed.5[row.names(  

    subset(sim_data.df, is.na(sim_data.df[, index1]) |  

    is.na(sim_data.df[, index2])), ]  

  p = ggplot(sim_data.imputed, aes_string( paste0("X",index1) ,  

paste0("X",index2 )) +  

  geom_point(alpha = 0.8, size = 0.7)+theme_bw() +  

  stat_ellipse(type = "norm", color = "#000099", alpha=0.5) +  

  geom_point(data = plot.imputed, aes_string( paste0("X",index1) ,  

paste0("X", (index2))),size=2.5, color="Magenta", alpha=0.9, shape=16) +  

  geom_point(data = plot.imputed2, aes( X1 , X2),size = 2.5,  

  color = "#FF9933", alpha = 0.8, shape = 18)
  return(p)
}

plot.me2(2, 4)

```

```
plot.me2(17, 18)
```

Density Plots

Finally, we can compare the densities of the original, manually-imputed and Amelia-imputed datasets. Remember that in this simulation, we had about 500 observations missing out of the 4000 that we synthetically generated (Fig. 3.29).

3.14 Parsing Webpages and Visualizing Tabular HTML Data

In this section, we will utilize the Earthquakes dataset on the SOCR website. It stores information about earthquakes of magnitudes larger than 5 on the Richter scale that were recorded between 1969 and 2007. Here is how we download and parse the data on the source webpage and ingest the information into R:

```

# install.packages("xml2")
library("XML"); library("xml2")
library("rvest")
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR\_Data\_Dinov\_021708\_Earthquakes")
html_nodes(wiki_url, "#content")
## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top" ...
earthquake<- html_table(html_nodes(wiki_url, "table"))[[2]])

```

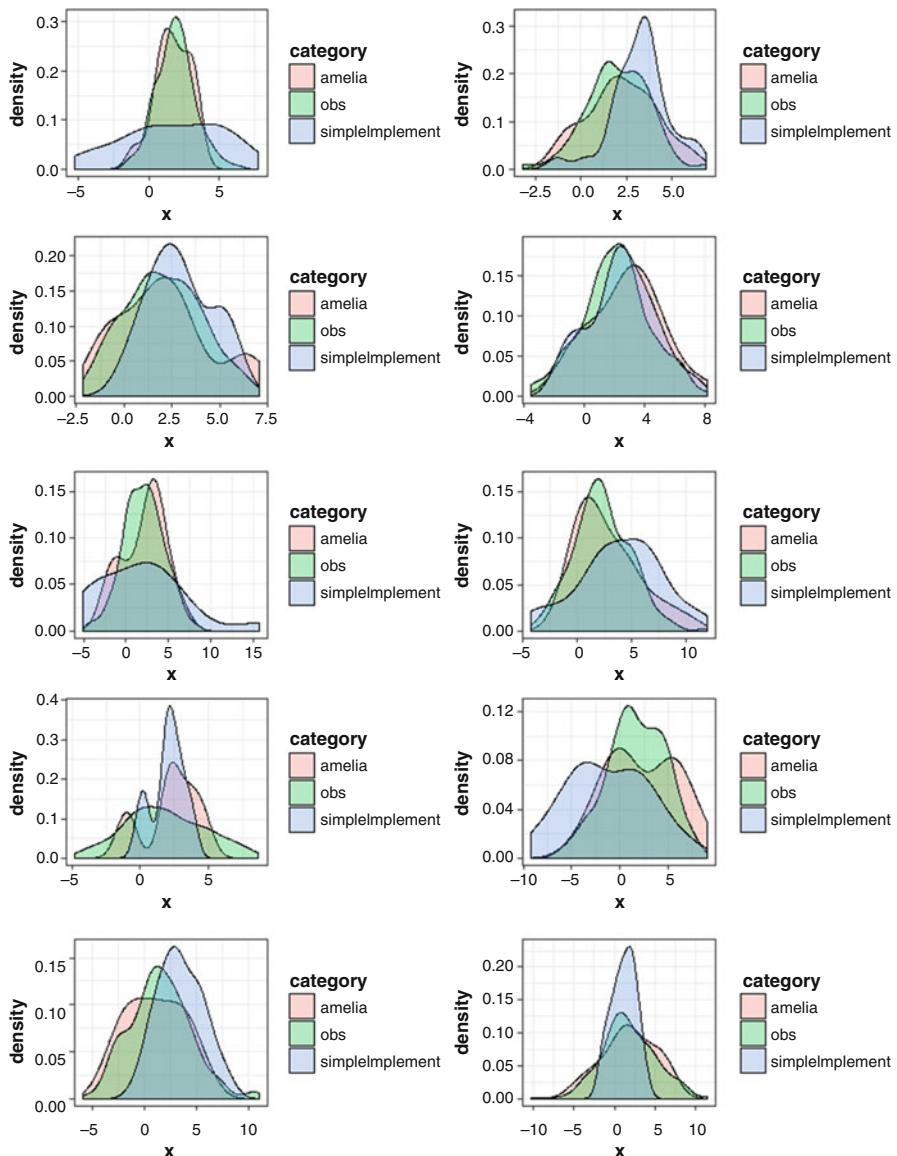


Fig. 3.29 Density plots of the original, manually-imputed and Amelia-imputed datasets, 10 features only

In this dataset, `Magt` (magnitude type) may be used as grouping variable. We will draw a “Longitude vs. Latitude” line plot from this dataset. The function we are using is called `ggplot()`, available from R package `ggplot2`. The input type for this function is a data frame, and `aes()` specifies the axes (Fig. 3.30).

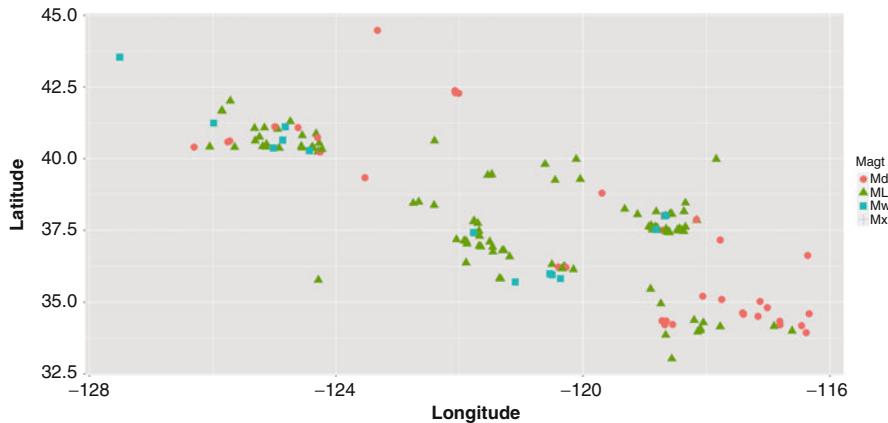


Fig. 3.30 Earthquake data plot of magnitude type (color/shape) against longitude (x) and latitude (y)

```
library(ggplot2)
plot4<-ggplot(earthquake, aes(Longitude, Latitude, group=Magt, color=Magt))+ 
  geom_point(data=earthquake, size=4, mapping=aes(x=Longitude, y=Latitude,
shape=Magt))
plot4 # or print(plot4)
```

We can see the plotting script consists of two parts. The first part `ggplot(earthquake, aes(Longitude, Latitude, group = Magt, color=Magt))` specifies the setting of the plot: dataset, group and color. The second part specifies that we are going to draw points for all data points. In later Chapters, we will frequently use `ggplot2`; which always takes multiple function calls, e.g., `function1 + function2`.

We can visualize the distribution for different variables using density plots. The following chunk of codes plots the distribution for Latitude among different Magnitude types. Also, it uses the `ggplot()` function combined with `geom_density()` (Fig. 3.31).

```
plot5<-ggplot(earthquake, aes(Latitude, size=1))+ 
  geom_density(aes(color=Magt))
plot5
```

We can also compute and display 2D Kernel Density and 3D Surface Plots. Plotting 2D Kernel Density and 3D Surface plots is very important and useful in multivariate exploratory data analytic.

We will use the `plot_ly()` function under `plotly` package, which takes data frame inputs.

To create a surface plot, we use two vectors: x and y , with length m and n , respectively. We also need a matrix: z of size $m \times n$. This z matrix is created from matrix multiplication between x and y .

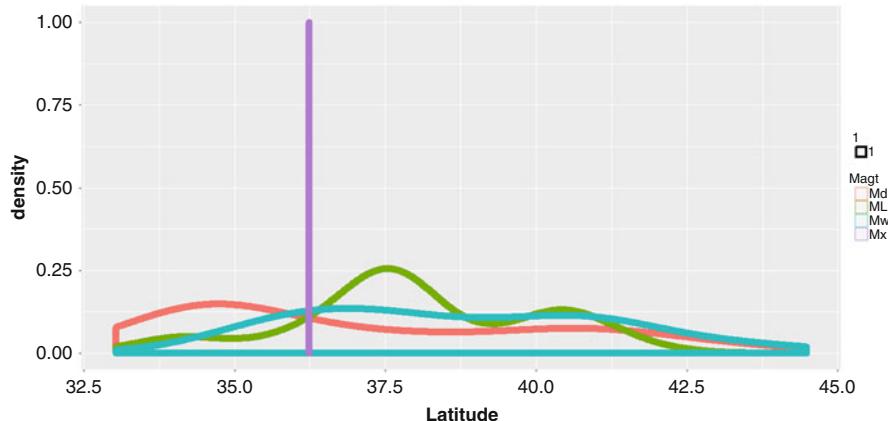


Fig. 3.31 Modified Earthquake density plot (y) of magnitude type against latitude coordinates (x)

The `kde2d()` function is needed for 2D kernel density estimation.

```
kernel_density <- with(earthquake, MASS::kde2d(Longitude, Latitude, n = 50))
```

Here, `z` is an estimate of the kernel density function. Then, we apply `plot_ly` to the list `kernel_density` via the `with()` function.

```
library(plotly)
with(kernel_density, plot_ly(x=x, y=y, z=z, type="surface"))
```

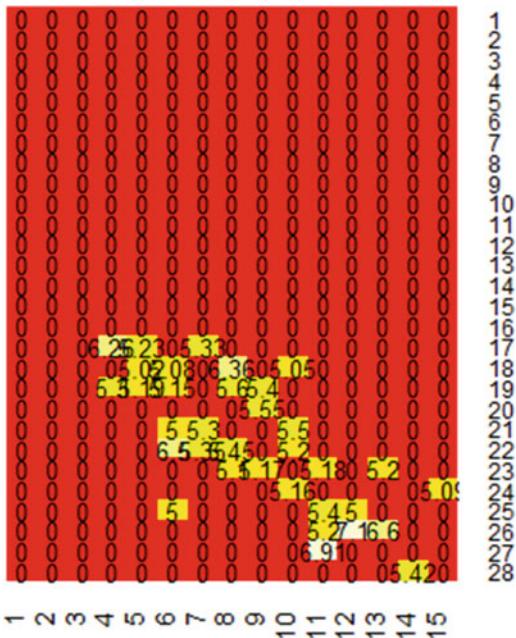
Note that we used the option “`surface`”, however you can experiment with the `type` option.

Alternatively, one can plot 1D, 2D, or 3D plots (Fig. 3.32):

```
plot_ly(x = ~earthquake$Longitude)
## No trace type specified:
## Based on info supplied, a 'histogram' trace seems appropriate.
## Read more about this trace type->https://plot.ly/r/reference/#histogram
plot_ly(x = ~earthquake$Longitude, y = ~earthquake$Latitude)
plot_ly(x=~earthquake$Longitude, y=~earthquake$Latitude, z=~earthquake$Mag)
df3D <- data.frame(x=earthquake$Longitude, y=earthquake$Latitude,
z=earthquake$Mag)

# Convert the Long (X, Y, Z) Earthquake format data into a Matrix Format
# install.packages("Matrix")
library("Matrix")
matrix_EarthQuakes <- with(df3D, sparseMatrix(i = as.numeric(180-x),
```

Fig. 3.32 Image representation of the kernel-density estimation of the Earthquake magnitude rendered as a heatmap



```
j=as.numeric(y), x=z, use.Last.ij=T, dimnames=list(Levels(x), Levels(y)))
dim(matrix_EarthQuakes)

## [1] 307 44

View(as.matrix(matrix_EarthQuakes))

# view matrix is 2D heatmap :
library("ggplot2"); library("gplots")

heatmap.2( as.matrix(matrix_EarthQuakes[280:307, 30:44]), Rowv=FALSE,
Colv=FALSE, dendrogram='none', cellnote=as.matrix(matrix_EarthQuakes[280:307, 30:44]),
notecol="black", trace='none', key=FALSE, lwid = c(.01, .99), lhei = c(.01, .99), margins = c(5, 15))

# Long -180<x<-170, Lat: 30<y<45, Z: 5<Mag<8
matrix_EarthQuakes <- with(df3D, sparseMatrix(i = as.numeric(180+x),
j=as.numeric(y), x=z,use.last.ij=TRUE, dimnames=list(Levels(x), Levels(y))))
mat1 <- as.matrix(matrix_EarthQuakes)
plot_ly(z = ~mat1, type = "surface")

# To plot the Aggregate (Summed) Magnitudes at all Long/Lat:
matrix_EarthQuakes <- with(df3D, sparseMatrix(i = as.numeric(180+x),
j=as.numeric(y), x=z, dimnames=list(Levels(x), Levels(y))))
mat1 <- as.matrix(matrix_EarthQuakes)
plot_ly(z = ~mat1, type = "surface")
# plot_ly(z = ~mat1[30:60, 20:40], type = "surface")
```

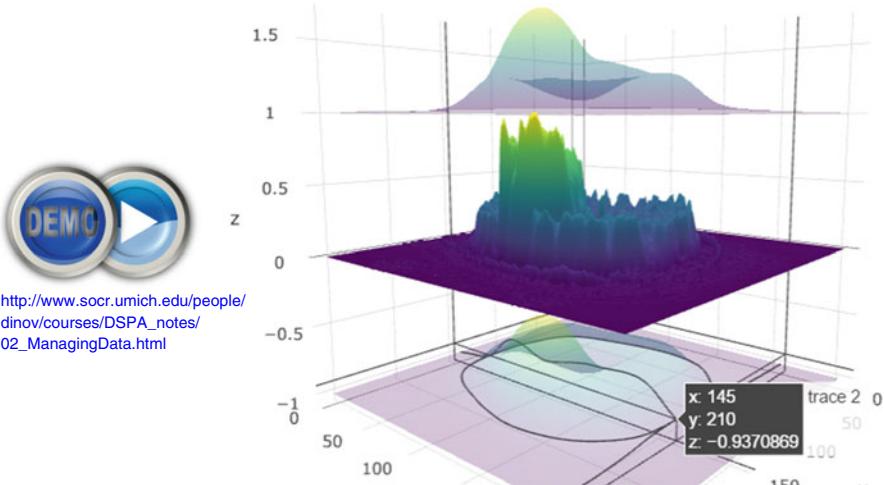


Fig. 3.33 Live demo of 3D kernel density surface plots using the Earthquake and 2D brain imaging data (http://www.socr.umich.edu/people/dinov/courses/DSPA_notes/02_ManagingData.html)

You can see interactive surface plot generated by plotly in the live demo listed on Fig. 3.33.

3.15 Cohort-Rebalancing (for Imbalanced Groups)

Comparing cohorts with imbalanced sample sizes (unbalanced designs) may present hidden biases in the results. Frequently, a cohort-rebalancing protocol is necessary to avoid such unexpected effects. Extremely unequal sample sizes can invalidate various parametric assumptions (e.g., homogeneity of variances). Also, there may be insufficient data representing the patterns belonging to the minority class(es) leading to inadequate capturing of the feature distributions. Although, the groups do not have to have equal sizes, a general rule of thumb is $0.5 < \frac{\text{size1}}{\text{size2}} < 2$. Tat is group sizes where one group is more than an order of magnitude larger than the size of another group has the potential for bias.

Example 1 Parkinson's Diseases Study involving neuroimaging, genetics, clinical, and phenotypic data for over 600 volunteers produced multivariate data for three cohorts: *HC=Healthy Controls(166)*, *PD=Parkinson's (434)*, *SWEDD = subjects without evidence for dopaminergic deficit (61)* (Figs. 3.34 and 3.35).

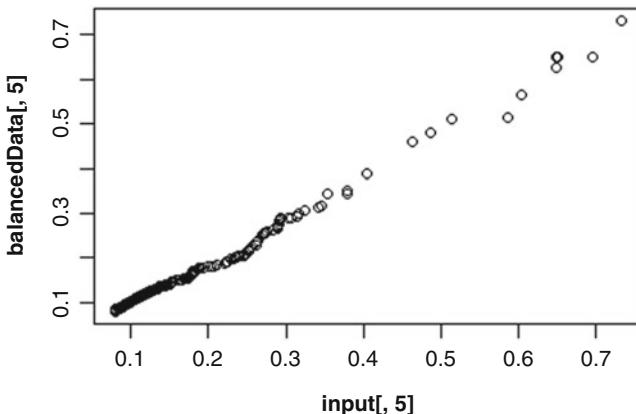


Fig. 3.34 Validation that cohort rebalancing does not substantially alter the distributions of features. This QQ plot of one variable shows the linearity of the quantiles of the initial (x) and rebalanced (y) data

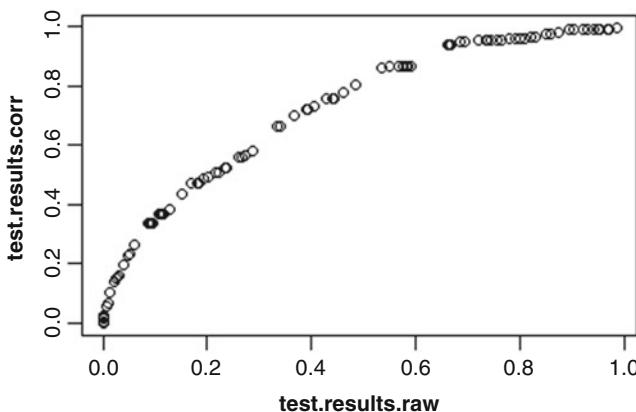


Fig. 3.35 Scatter plot of the raw (x) and corrected/adjusted (y) p-values corresponding to the paired two-sample Wilcoxon non-parametric test comparing the raw and rebalanced features

```

# update.packages()
# load the data: 06_PPMI_ClassificationValidationData_Short.csv
ppmi_data <- read.csv("https://umich.instructure.com/files/330400/downLoad?do
wnLoad_frd=1", header=TRUE)
table(ppmi_data$ResearchGroup)

# binarize the Dx classes
ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control",
"Control", "Patient")
attach(ppmi_data)

head(ppmi_data)

```

```

# Model-free analysis, classification
# install.packages("crossval")
# install.packages("ada")
# library("crossval")
require(crossval)
require(ada)
#set up adaboosting prediction function

# Define a new classification result-reporting function
my.ada <- function (train.x, train.y, test.x, test.y, negative, formula)
{
  ada.fit <- ada(train.x, train.y)
  predict.y <- predict(ada.fit, test.x)
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
  # negative is the label of a negative "null" sample (default: "control").
  return (out)
}

# balance cases
# SMOTE: Synthetic Minority Oversampling Technique to handle class
misbalance in binary classification.
set.seed(1000)
# install.packages("unbalanced") to deal with unbalanced group data
require(unbalanced)
ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
uniqueID <- unique(ppmi_data$FID_IID)
ppmi_data <- ppmi_data[ppmi_data$VisitID==1, ]
ppmi_data$PD <- factor(ppmi_data$PD)

colnames(ppmi_data)
# ppmi_data[1<-ppmi_data[, c(3:281, 284, 287, 336:340, 341)]]
n <- ncol(ppmi_data)
output.1 <- ppmi_data$PD

# remove Default Real Clinical subject classifications!
ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup",
"PD", "X", "FID_IID"))]
# output <- as.matrix(ppmi_data[, which(names(ppmi_data) %in% {"PD"})])
output <- as.factor(ppmi_data$PD)
c(dim(input), dim(output))

#balance the dataset
data.1<-ubBalance(X= input, Y=output, type="ubSMOTE", percOver=300,
percUnder=150, verbose=TRUE)

# percOver = A number that drives the decision of how many extra cases from
the minority class are generated (known as over-sampling).
# k = A number indicating the number of nearest neighbors that are used to g
enerate the new examples of the minority class.
# percUnder = A number that drives the decision of how many extra cases from
the majority classes are selected for each case generated from the minority
class (known as under-sampling)

```

```

balancedData<-cbind(data.1$X, data.1$Y)
table(data.1$Y)

nrow(data.1$X); ncol(data.1$X)
nrow(balancedData); ncol(balancedData)
nrow(input); ncol(input)

colnames(balancedData) <- c(colnames(input), "PD")

# check visually for differences between the distributions of the raw
# (input) and rebalanced data (for only one variable, in this case)
qqplot(input[, 5], balancedData [, 5])

####Check balance
## Wilcoxon test
alpha.0.05 <- 0.05
test.results.bin <- NULL          # binarized/dichotomized p-values
test.results.raw <- NULL          # raw p-values

for (i in 1:(ncol(balancedData)-1))
{
  test.results.raw[i]<-wilcox.test(input[, i], balancedData [, i])$p.value
  test.results.bin [i] <- ifelse(test.results.raw [i] > alpha.0.05, 1, 0)
  print(c("i=", i, "Wilcoxon-test=", test.results.raw [i]))
}
print(c("Wilcoxon test results: ", test.results.bin))

test.results.corr <- stats:::p.adjust(test.results.raw, method = "fdr", n = length(test.results.raw))
# where methods are "holm", "hochberg", "hommel", "bonferroni", "BH", "BY",
#"fdr", "none")
plot(test.results.raw, test.results.corr)

# zeros (0) are significant independent between-group T-test differences,
# ones (1) are insignificant

# Check the Differences between the rate of significance between the raw and
# FDR-corrected p-values
test.results.bin <- ifelse(test.results.raw > alpha.0.05, 1, 0)
table(test.results.bin)
test.results.corr.bin <- ifelse(test.results.corr > alpha.0.05, 1, 0)
table(test.results.corr.bin)

```

3.16 Appendix

3.16.1 Importing Data from SQL Databases

We can also import SQL databases into R. First, we need to install and load the RODBC(R Open Database Connectivity) package.

```

install.packages("RODBC", repos = "http://cran.us.r-project.org")
library(RODBC)

```

Then, we could open a connection to the SQL server database with Data Source Name (DSN), via Microsoft Access. More details are provided online.

3.16.2 R Code Fragments

Below are some code snippets used to generate some of the graphs shown in this Chapter.

```
#Right Skewed
N <- 10000
x <- rnbinom(N, 10, .5)
hist(x,
  xlim=c(min(x), max(x)), probability=T, nclass=max(x)-min(x)+1,
  col='lightblue', xlab=' ', ylab=' ', axes=F,
  main='Right Skewed')
Lines(density(x, bw=1), col='red', lwd=3)

#No Skew
N <- 10000
x <- rnorm(N, 0, 1)
hist(x, probability=T,
  col='lightblue', xlab=' ', ylab=' ', axes=F,
  main='No Skew')
Lines(density(x, bw=0.4), col='red', lwd=3)

#Uniform density
x<-runif(1000, 1, 50)
hist(x, col='lightblue', main="Uniform Distribution", probability = T, xlab=
  "", ylab="Density", axes=F)
abline(h=0.02, col='red', lwd=3)

#68-95-99.7 rule
x <- rnorm(N, 0, 1)
hist(x, probability=T,
  col='lightblue', xlab=' ', ylab=' ', axes = F,
  main='68-95-99.7 Rule')
Lines(density(x, bw=0.4), col='red', lwd=3)
axis(1, at=c(-3, -2, -1, 0, 1, 2, 3), labels = expression(mu-3*sigma,
  mu-2*sigma, mu-sigma, mu, mu+sigma, mu+2*sigma, mu+3*sigma))
abline(v=-1, lwd=3, lty=2)
abline(v=1, lwd=3, lty=2)
abline(v=-2, lwd=3, lty=2)
abline(v=2, lwd=3, lty=2)
abline(v=-3, lwd=3, lty=2)
abline(v=3, lwd=3, lty=2)
text(0, 0.2, "68%")
segments(-1, 0.2, -0.3, 0.2, col = 'red', lwd=2)
segments(1, 0.2, 0.3, 0.2, col = 'red', lwd=2)
text(0, 0.15, "95%")
segments(-2, 0.15, -0.3, 0.15, col = 'red', lwd=2)
segments(2, 0.15, 0.3, 0.15, col = 'red', lwd=2)
text(0, 0.1, "99.7%")
segments(-3, 0.1, -0.3, 0.1, col = 'red', lwd=2)
segments(3, 0.1, 0.3, 0.1, col = 'red', lwd=2)
```

3.17 Assignments: 3. Managing Data in R

3.17.1 Import, Plot, Summarize and Save Data

Load the following two datasets, generate summary statistics for all variables, plot some of the features (e.g., histograms, box plots, density plots, etc.), and save the data locally as CSV files:

- ALS case-study data, https://umich.instructure.com/courses/38100/files/folder/Case_Studies/15_ALS_CaseStudy.
- SOCR Knee Pain Data, http://wiki.socr.umich.edu/index.php/SOCR_Data_KneePainData_041409.

3.17.2 Explore some Bivariate Relations in the Data

Use ALS case-study data or SOCR Knee Pain Data to explore some bivariate relations (e.g. bivariate plot, correlation, table, crosstable, etc.)

Use 07_UMich_AnnArbor_MI_TempPrecipitation_HistData_1900_2015 data to show the relations between temperature and time. [Hint: use `geom_line` or `geom_bar`].

Some sample code for dealing with the table of temperatures data is included below.

```

<code>
Temp_Data <- as.data.frame(read.csv("https://umich.instructure.com/files/706163/download?download_frd=1", header=T, na.strings=c("", ".", "NA", "NR")))
summary(Temp_Data)
# View(Temp_Data); colnames(Temp_Data)

# Wide-to-Long transformation: reshape arguments include
# (1) list of variable names that define the different times or metrics (varying),
# (2) the name we wish to give the variable containing these values in our long dataset (v.names),
# (3) the name we wish to give the variable describing the different times or metrics (timevar),
# (4) the values this variable will have (times), and
# (5) the end format for the data (direction)
# Before reshaping make sure all data types are the same as putting them in 1 column will
# otherwise generate inconsistencies/errors
colN <- colnames(Temp_Data)[-1]
LongTempData <- reshape(Temp_Data, varying = colN, v.names = "Temps", timevar="Months", times = colN, direction = "Long")

# View(LongTempData)
bar2 <- ggplot(LongTempData, aes(x = Months, y = Temps, fill = Months)) +
  geom_bar(stat = "identity")
print(bar2)
bar3 <- ggplot(LongTempData, aes(x = Year, y = Temps, fill = Months)) +
  geom_bar(stat = "identity")
print(bar3)

p <- ggplot(LongTempData, aes(x=Year, y=as.integer(Temps), colour=Months)) +
  geom_line()
p
</code>

```

3.17.3 Missing Data

Introduce (artificially) some missing data, impute the missing values and examine the differences between the original, incomplete, and imputed data.

3.17.4 Surface Plots

Generate a surface plot for the (RF) Knee Pain data illustrating the 2D distribution of locations of the patient reported knee pain (use *plot_ly* and kernel density estimation).

3.17.5 Unbalanced Designs

Rebalance the groups of ALS (training data) patients according to $Age > 50$ and $Age \leq 50$ using synthetic minority oversampling (SMOTE) to ensure approximately equal cohort sizes. (Hint: you may need to set 1 as the minority class.)

3.17.6 Aggregate Analysis

Use the California Ozone Data to generate a summary report. Make sure to include: summary for every variable, the structure of the data, convert to appropriate data type, discuss the tendency of the ozone average concentration, explore the differences of the ozone concentration a specific area (you may select year 2006), explore the seasonal change of ozone concentration.

References

<https://plot.ly/r/>
<http://www.statmethods.net/management>

Chapter 4

Data Visualization



In this chapter, we use a broad range of simulations and hands-on activities to highlight some of the basic data visualization techniques using R. A brief discussion of alternative visualization methods is followed by demonstrations of histograms, density, pie, jitter, bar, line and scatter plots, as well as strategies for displaying trees, more general graphs, and 3D surface plots. Many of these are also used throughout the textbook in the context of addressing the graphical needs of specific case-studies.

It is practically impossible to cover all options of every different visualization routine. Readers are encouraged to experiment with each visualization type, change input data and parameters, explore the function documentation using R-help (e.g., `?plot`), and search online for new R visualization packages and new functionality, which are continuously being developed.

We will cover (1) one specific classification of visualization methods, (2) composition (e.g., density, histogram), comparison (e.g., jitter, bar, correlation) and relationship (e.g., line) plots, (3) 2D kernel density and 3D surface plots, and (4) 3D and 4D visualization of solids, (hyper)volumes.

4.1 Common Questions

- What exploratory visualization techniques are available to graphically interrogate my specific data?
- How do we examine paired associations and correlations in a multivariate dataset?

4.2 Classification of Visualization Methods

Scientific data-driven or simulation-driven visualization methods are hard to classify. The following list of criteria can be used to characterize alternative data visualization strategies:

- **Data Type:** structured/unstructured, small/large, complete/incomplete, time/space, ASCII/binary, Euclidean/non-Euclidean, etc.
- **Task type:** Task type is one of the aspects considered in classification of visualization techniques, which provides a means of interaction between the researcher, the data, and the display software/platform
- **Scalability:** Visualization techniques are subject to some limitations, such as the amount of data that a particular technique can exhibit
- **Dimensionality:** Visualization techniques can also be classified according to the number of attributes
- **Positioning and Attributes:** the distribution of attributes on the chart may affect the interpretation of the display representation, e.g., correlation analysis, where the relative distance among the plotted attributes is relevant for observation
- **Investigative Need:** the specific scientific question or exploratory interest may also determine the type of visualization:
 - Examining the composition of the data
 - Exploring the distribution of the data
 - Contrasting or comparing several data elements, relations, association
 - Unsupervised exploratory data mining.

Also, we have the following table for common data visualization methods according to task types (Fig. 4.1):

We introduce common data visualization methods according to this classification criterion, albeit this is not a unique or even broadly agreed upon ontological characterization of exploratory data visualization.

4.3 Composition

In this section, we will see composition plots for different types of variables and data structures.

4.3.1 *Histograms and Density Plots*

One of the first few graphs we learn is a histogram plot. In R, the command `hist()` is applied to a vector of values and used for plotting histograms. The famous nineteenth century statistician Karl Pearson introduced histograms as graphical

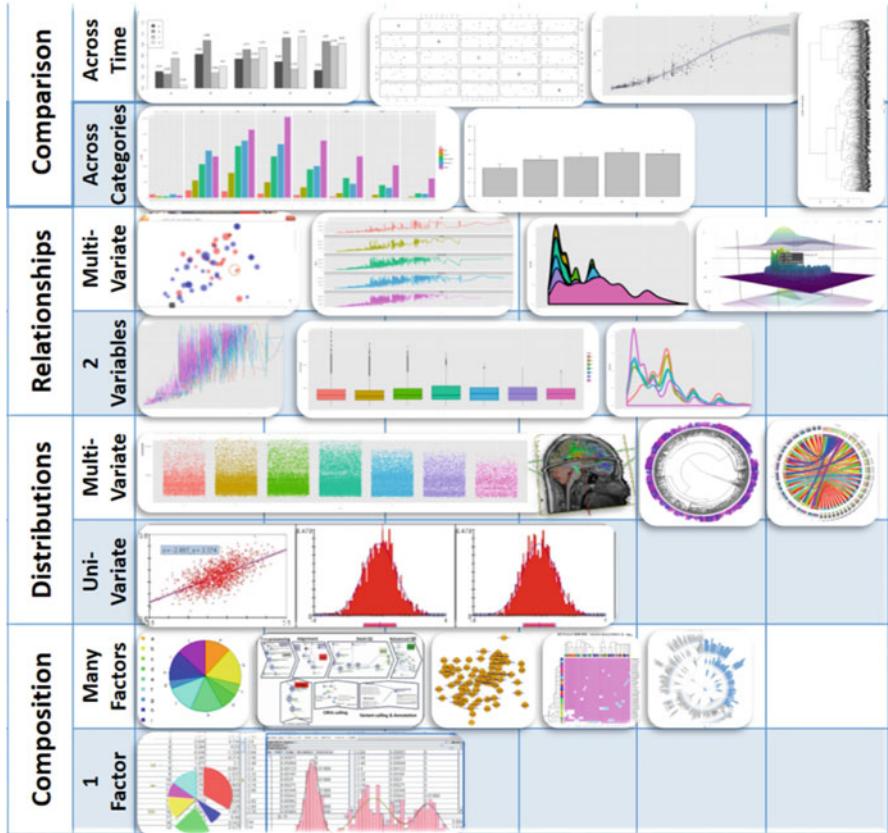


Fig. 4.1 Schematic depiction of a hierarchical classification of different visualization methods

representations of the distribution of a sample of numeric data. The histogram plot uses the data to infer and display the probability distribution of the underlying population that the data is sampled from. Histograms are constructed by selecting a certain number of bins covering the range of values of the observed process. Typically, the number of bins for a data array of size N should be equal to \sqrt{N} . These bins form a partition (disjoint and covering sets) of the range. Finally, we compute the relative frequency representing the number of observations that fall within each bin interval. The histogram just plots a piece-wise step-function defined over the union of the bin interfaces whose height equals the observed relative frequencies (Fig. 4.2).

Fig. 4.2 Overlay of Normal distribution histogram and density curve plot

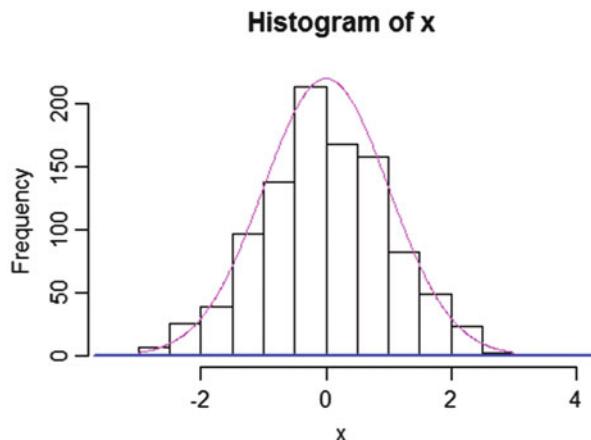
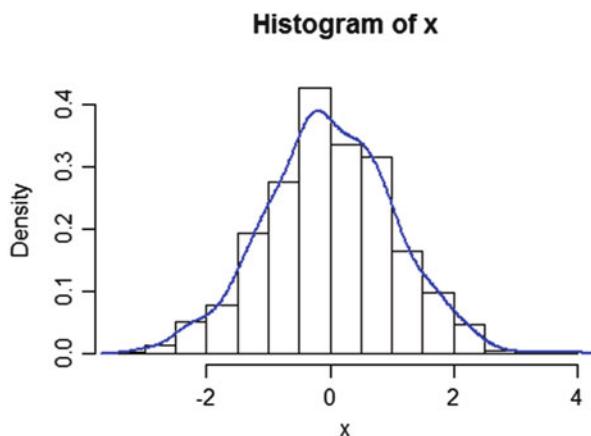


Fig. 4.3 Overlay of histogram plot and a density curve estimate



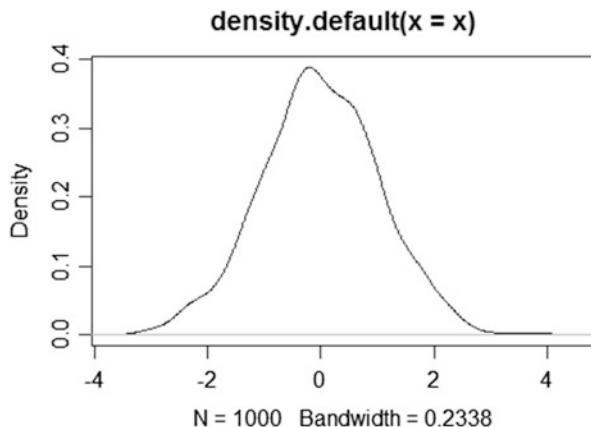
```
set.seed(1)
x<-rnorm(1000)
hist(x, freq=T, breaks = 10)
Lines(density(x), lwd=2, col="blue")
t <- seq(-3, 3, by=0.01)
Lines(t, 550*dnorm(t,0,1), col="magenta") # add the theoretical density line
```

Here, `freq=T` shows the frequency for each x value and `breaks` controls for number of bars in our histogram.

The shape of the last histogram we drew is very close to a Normal distribution (because we sampled from this distribution by `rnorm`). We can add a density line to the histogram (Fig. 4.3).

```
hist(x, freq=F, breaks = 10)
Lines(density(x), lwd=2, col="blue")
```

Fig. 4.4 Direct plot of the estimated Normal distribution density curve



We used the option `freq=F` to make the y axis represent the “relative frequency”, or “density”. We can also use `plot(density(x))` to draw the density plot by itself (Fig. 4.4).

```
plot(density(x))
```

4.3.2 Pie Chart

We are all very familiar with pie charts that show us the components of a big “cake”. Although pie charts provide effective simple visualization in certain situations, it may also be difficult to compare segments within a pie chart or across different pie charts. Other plots like bar chart, box or dot plots may be attractive alternatives.

We will use the Letter Frequency Data on SOCR website to illustrate the use of pie charts.

```
library(rvest)
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_LetterFrequencyData")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top" ...
...
letter <- html_table(html_nodes(wiki_url, "table"))[[1]]
summary(letter)
```

```

##      Letter      English      French      German
## Length:27      Min. :0.00000  Min. :0.00000  Min. :0.00000
## Class :character 1st Qu.:0.01000  1st Qu.:0.01000  1st Qu.:0.01000
## Mode :character Median :0.02000  Median :0.03000  Median :0.03000
##                               Mean :0.03667  Mean :0.03704  Mean :0.03741
##                               3rd Qu.:0.06000  3rd Qu.:0.06500  3rd Qu.:0.05500
##                               Max. :0.13000  Max. :0.15000  Max. :0.17000
##      Spanish      Portuguese      Esperanto      Italian
## Min. :0.00000  Min. :0.00000  Min. :0.00000  Min. :0.00000
## 1st Qu.:0.01000 1st Qu.:0.00500  1st Qu.:0.01000  1st Qu.:0.00500
## Median :0.03000 Median :0.03000  Median :0.03000  Median :0.03000
## Mean :0.03815  Mean :0.03778  Mean :0.03704  Mean :0.03815
## 3rd Qu.:0.06000 3rd Qu.:0.05000  3rd Qu.:0.06000  3rd Qu.:0.06000
## Max. :0.14000  Max. :0.15000  Max. :0.12000  Max. :0.12000
##      Turkish      Swedish      Polish      Toki_Pona
## Min. :0.00000  Min. :0.00000  Min. :0.00000  Min. :0.00000
## 1st Qu.:0.01000 1st Qu.:0.01000  1st Qu.:0.01500  1st Qu.:0.00000
## Median :0.03000 Median :0.03000  Median :0.03000  Median :0.03000
## Mean :0.03667  Mean :0.03704  Mean :0.03704  Mean :0.03704
## 3rd Qu.:0.05500 3rd Qu.:0.05500  3rd Qu.:0.04500  3rd Qu.:0.05000
## Max. :0.12000  Max. :0.10000  Max. :0.20000  Max. :0.17000
##      Dutch      Avgverage
## Min. :0.00000  Min. :0.00000
## 1st Qu.:0.01000 1st Qu.:0.01000
## Median :0.02000 Median :0.03000
## Mean :0.03704  Mean :0.03741
## 3rd Qu.:0.06000 3rd Qu.:0.06000
## Max. :0.19000  Max. :0.12000

```

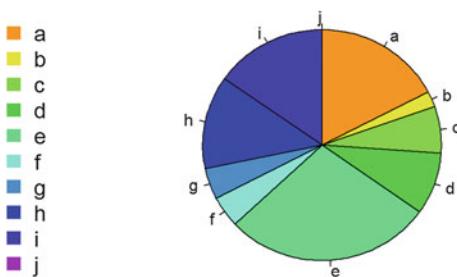
We can try to plot the frequency for the first ten English letters. The left hand side shows a table made by the function `legend` (Fig. 4.5).

```

par(mfrow=c(1, 2))
pie(letter$English[1:10], labels=Letter$Letter[1:10], col=rainbow(10, start=0.1, end=0.8), clockwise=TRUE, main="First 10 Letters Pie Chart")
pie(letter$English[1:10], labels=Letter$Letter[1:10], col=rainbow(10, start=0.1, end=0.8), clockwise=TRUE, main="First 10 Letters Pie Chart")
legend("topleft", legend=Letter$Letter[1:10], cex=1.3, bty="n", pch=15, pt.cex=1.8, col=rainbow(10, start=0.1, end=0.8), ncol=1)

```

Fig. 4.5 Pie chart showing the frequency of English use of the first ten Latin letters (a–j)



The input type for `pie()` is a vector of non-negative numerical quantities. In the `pie` function, we list the data that we are going to use (positive and numeric), the labels for each of them, and the colors we want to use for each sector. In the `legend` function, we put the location in the first slot and use `legend` as the labels for the colors. Also `cex`, `bty`, `pch`, and `pt.cex` are all graphic parameters that we have talked about in Chap. 2.

More elaborate pie charts, using the Latin letter data, will be demonstrated using `ggplot` below (in the Appendix of this chapter).

4.3.3 Heat Map

Another common data visualization method is the `heat map`. Heat maps can help us visualize intuitively the individual values in a matrix. It is widely used in genetics research and financial applications.

We will illustrate the use of heat maps, based on a neuroimaging genetics case-study data about the association (p-values) of different brain regions of interest (ROIs) and genetic traits (SNPs) for Alzheimer's disease (AD) patients, subjects with mild cognitive impairment (MCI), and normal controls (NC). First, let's import the data into R. The data are 2D arrays where the rows represent different genetic SNPs, columns represent brain ROIs, and the cell values represent the strength of the SNP-ROI association, a probability value (smaller p-values indicate stronger neuroimaging-genetic associations).

```
AD_Data <- read.table("https://umich.instructure.com/files/330387/download?d
ownload_frd=1", header=TRUE, row.names=1, sep=",", dec=".")  
MCI_Data <- read.table("https://umich.instructure.com/files/330390/download?d
ownload_frd=1", header=TRUE, row.names=1, sep=",", dec=".")  
NC_Data <- read.table("https://umich.instructure.com/files/330391/download?d
ownload_frd=1", header=TRUE, row.names=1, sep=",", dec=".")
```

Then, we load the R packages we need for heat maps (use `install.packages("package name")` first if you have not previously install them).

```
require(graphics)
require(grDevices)
library(gplots)
```

We can convert the datasets into matrices.

```
AD_mat <- as.matrix(AD_Data); class(AD_mat) <- "numeric"  
MCI_mat <- as.matrix(MCI_Data); class(MCI_mat) <- "numeric"  
NC_mat <- as.matrix(NC_Data); class(NC_mat) <- "numeric"
```

We may also want to set up the row (rc) and column (cc) colors for each cohort.

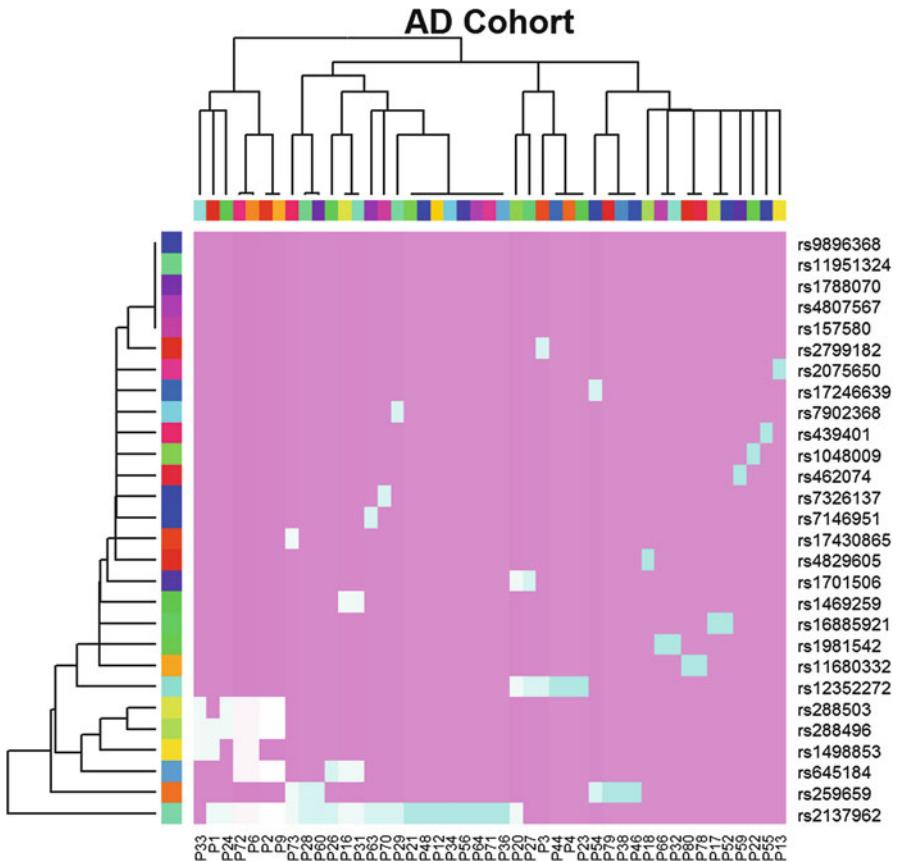


Fig. 4.6 Hierarchically clustered heatmap for the Alzheimer's disease (AD) cohort of the dementia study. The rows indicate the unique SNP reference sequence (rs) IDs and the columns index specific brain regions of interest (ROIs) that are associated with the genomic biomarkers (rows)

```
rcAD <- rainbow(nrow(AD_mat), start = 0, end = 1.0); ccAD<-rainbow(ncol(AD_mat), start = 0, end = 1.0)
rcMCI <- rainbow(nrow(MCI_mat), start = 0, end=1.0); ccMCI<-rainbow(ncol(MCI_mat), start=0, end=1.0)
rcNC <- rainbow(nrow(NC_mat), start = 0, end = 1.0); ccNC<-rainbow(ncol(NC_mat), start = 0, end = 1.0)
```

Finally, we can plot the heat maps by specifying the input type of `heatmap()` to be a numeric matrix (Figs. 4.6, 4.7, and 4.8).

```
hvAD <- heatmap(AD_mat, col=cm.colors(256), scale="column", RowSideColors = rcAD, ColSideColors = ccAD, margins = c(2, 2), main="AD Cohort")
```

```
hvMCI <- heatmap(MCI_mat, col = cm.colors(256), scale = "column", RowSideColors = rcMCI, ColSideColors = ccMCI, margins = c(2, 2), main="MCI Cohort")
```

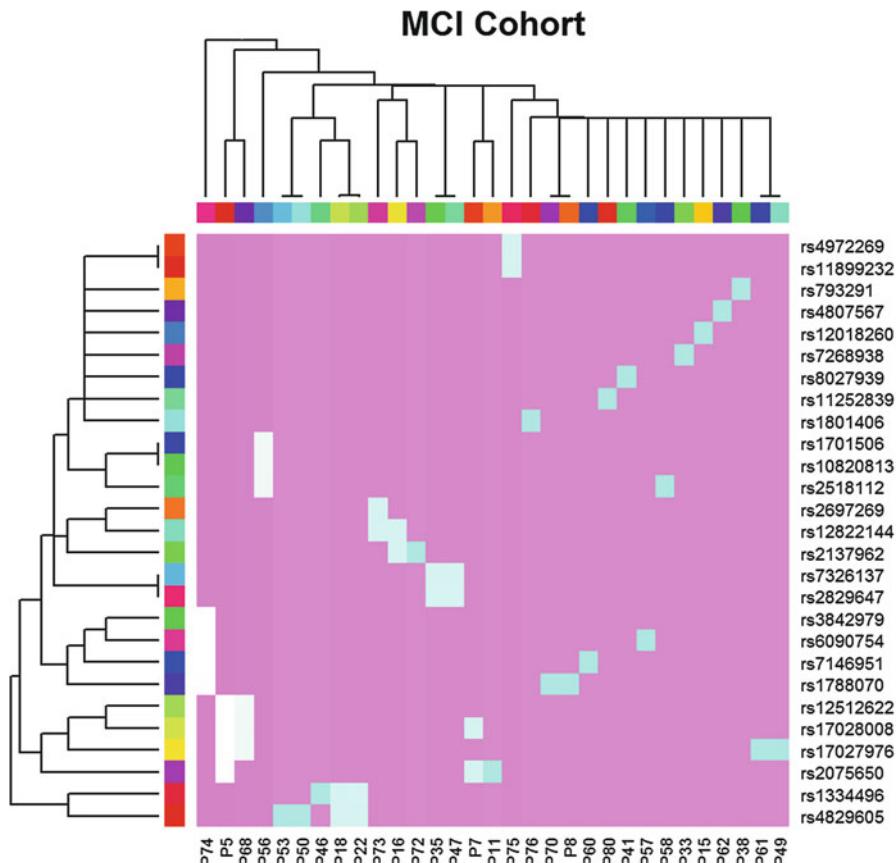


Fig. 4.7 Hierarchically clustered heatmap for the Mild Cognitive Impairment (MCI) cohort

```
hvNC <- heatmap(NC_mat, col=cm.colors(256), scale="column", RowSideColors = rcNC, ColSideColors = ccNC, margins = c(2, 2), main="NC Cohort")
```

In the `heatmap()` function, the first argument provides the input matrix we want to use. `col` is the color scheme; `scale` is a character indicating if the values should be centered and scaled in either the row direction or the column direction, or `none` ("row", "column", and "none"); `RowSideColors` and `ColSideColors` creates the color names for horizontal side bars.

The differences between the AD, MCI, and NC heat maps are suggestive of variations of genetic traits or alternative brain regions that may be affected in the three clinically different cohorts.

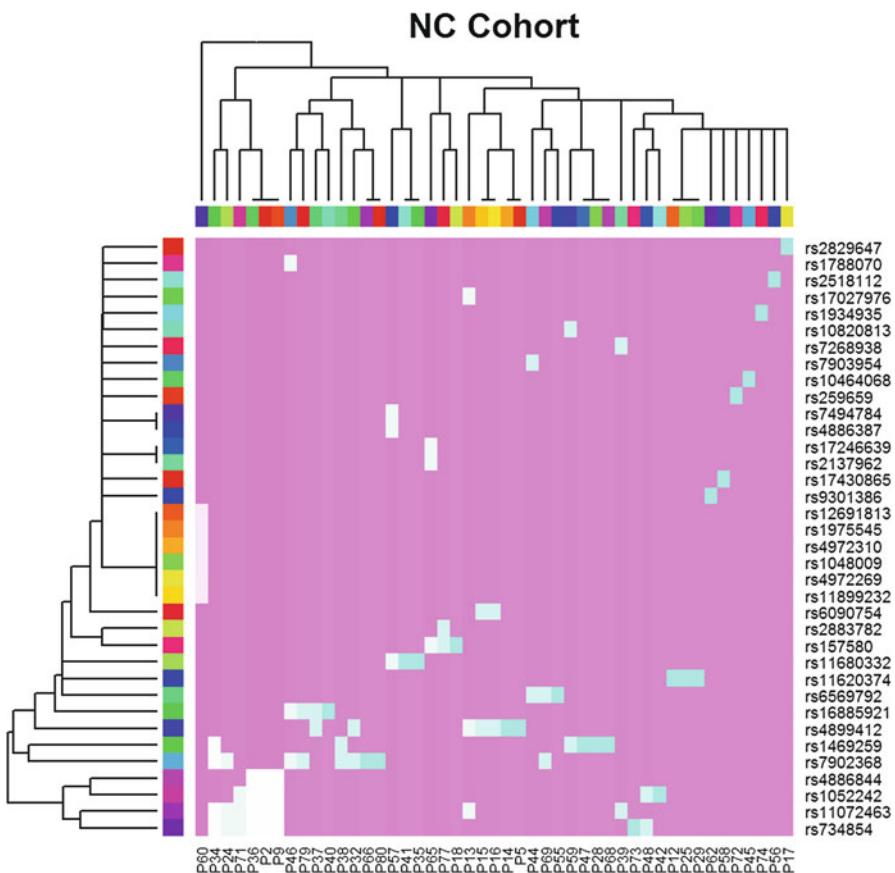


Fig. 4.8 Hierarchically clustered heatmap for the healthy normal controls (NC) cohort

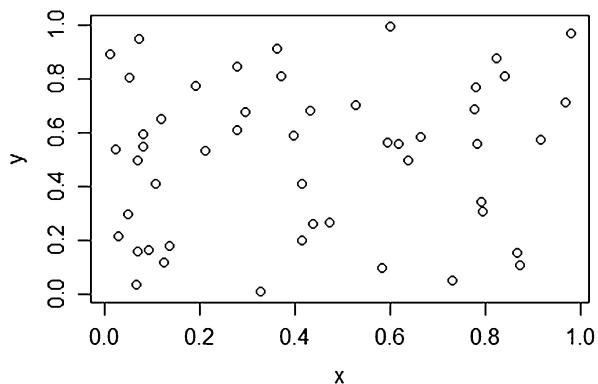
4.4 Comparison

Plots used for comparing different individuals, groups of subjects, or multiple units represent another set of popular exploratory visualization tools.

4.4.1 Paired Scatter Plots

Scatter plots use the 2D Cartesian plane to display a pair of variables. 2D points represent the values of the two variables corresponding to the two coordinate axes. The position of each 2D point on is determined by the values of the first and second variables, which represent the horizontal and vertical axes. If no clear dependent

Fig. 4.9 Scatter plot of bivariate uniform process



variable exists, either variable can be plotted on the X axis and the corresponding scatter plot will illustrate the degree of correlation (not necessarily causation) between the two variables.

Basic scatter plots can be plotted by function `plot(x, y)` (Fig. 4.9).

```
x<-runif(50)
y<-runif(50)
plot(x, y, main="Scatter Plot")
```

`qplot()` is another way to display elaborate scatter plots. We can manage the colors and sizes of dots. The input type for `qplot()` is a data frame. In the following example, larger x will have larger dot sizes. We also grouped the data as ten points per group (Fig. 4.10).

```
library(ggplot2)
cat <- rep(c("A", "B", "C", "D", "E"), 10)
plot.1 <- qplot(x, y, geom="point", size=5*x, color=cat, main="GGplot with R
relative Dot Size and Color")
print(plot.1)
```

Now, let's draw a paired scatter plot with three variables. The input type for `pairs()` function is a matrix or data frame (Fig. 4.11).

```
z<-runif(50)
pairs(data.frame(x, y, z))
```

We can see that variable names are on the diagonal of this scatter plot matrix. Each plot uses the column variable as its X-axis and row variable as its Y-axis.

Let's see a real word data example. First, we can import the Mental Health Services Survey Data into R, which is on the case-studies website.

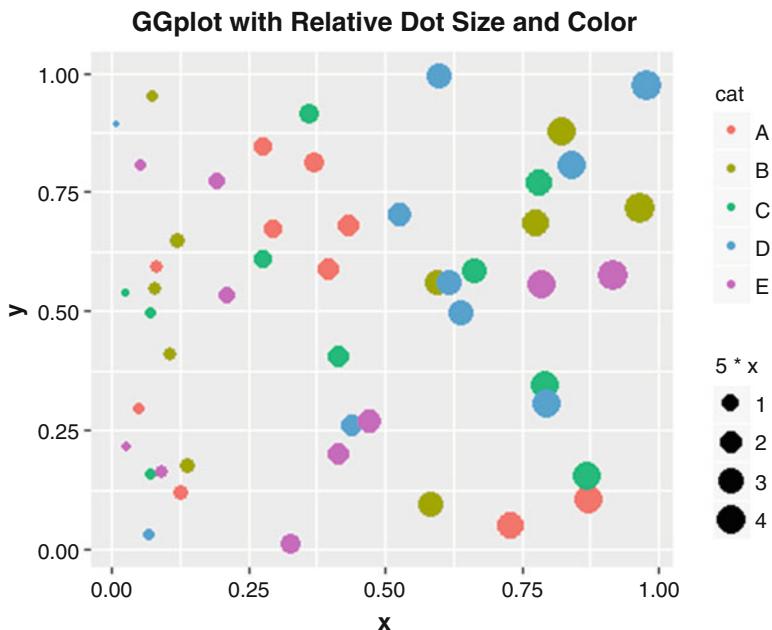


Fig. 4.10 Simulated bubble plot depicting four variable features represented as x and y axes, size and color

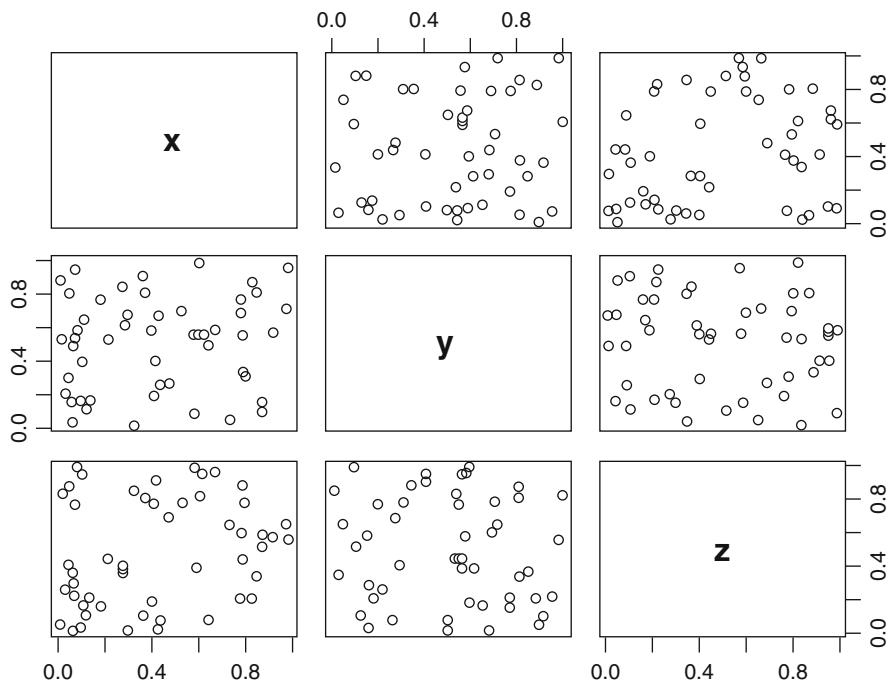


Fig. 4.11 A pairs plot depicts the bivariate relations in multivariate datasets

```

data1 <- read.table('https://umich.instructure.com/files/399128/download?down
nload_frd=1', header=T)
head(data1)

##      STFIPS majorfundtype FacilityType Ownership Focus PostTraum GLBT
## 1  southeast           1           5        2     1       0      0
## 2  southeast           3           5        3     1       0      0
## 3  southeast           1           6        2     1       1      1
## 4 greatlakes          NA          NA        2     2       0      0
## 5 rockymountain        1           5        5     2       0      0
## 6 mideast              NA          NA        2     2       1      0
##   num qual supp
## 1  5   NA  NA
## 2  4   15  4
## 3  9   15  NA
## 4  7   14  6
## 5  9   18  NA
## 6  8   14  NA

attach(data1)

```

From the `head()` output, we observe that there are a lot of `NA`'s in the dataset. `pairs` automatically deals with this problem (Figs. 4.12 and 4.13).

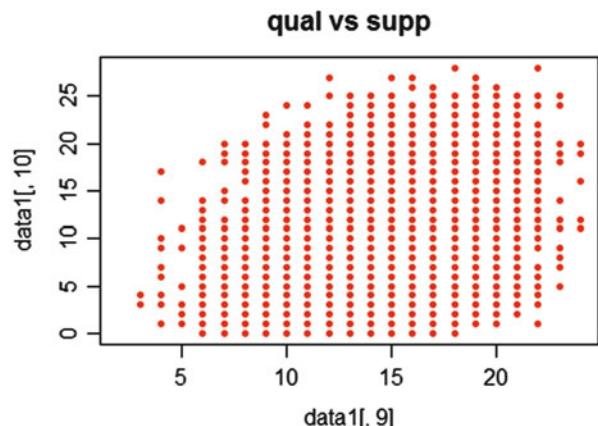
```
plot(data1[, 9], data1[, 10], pch=20, col="red", main="qual vs supp")
```

```
pairs(data1[, 5:10])
```

Figure 4.12 represents just one of the plots shown in the collage on Fig. 4.13. We can see that `Focus` and `PostTraum` have no relationship - `Focus` can equal to 3 or 1 in either of the `PostTraum` values (0 or 1). On the other hand, larger `supp` tends to correspond to larger `qual` values.

To see this trend we can also make a plot using `qplot` function. This allow us to add a smooth model curve forecasting a possible trend (Fig. 4.14).

Fig. 4.12 Each of the bivariate plots in a pairs plot collage may be zoomed up and explored further



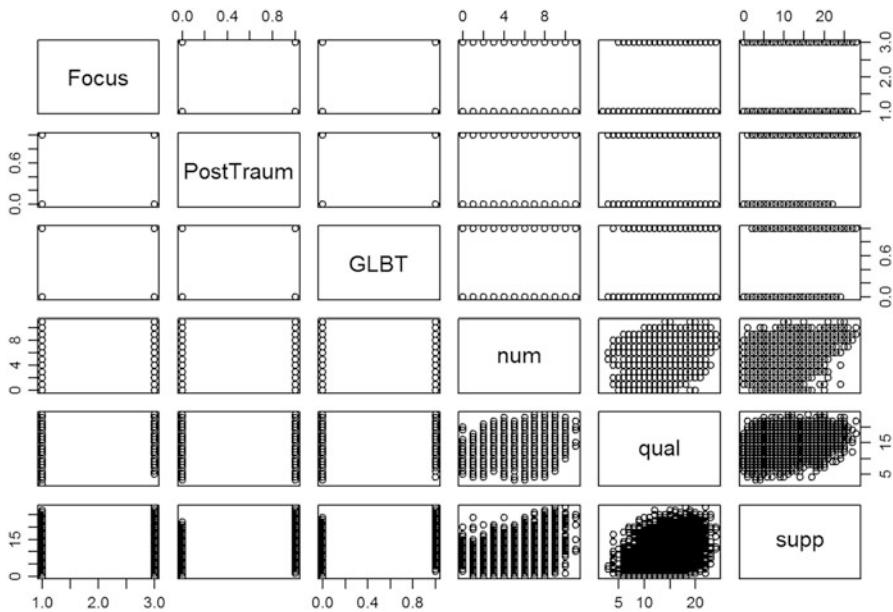


Fig. 4.13 A more elaborate 6D pairs plot showing the type and scale of each variable and their bivariate relations

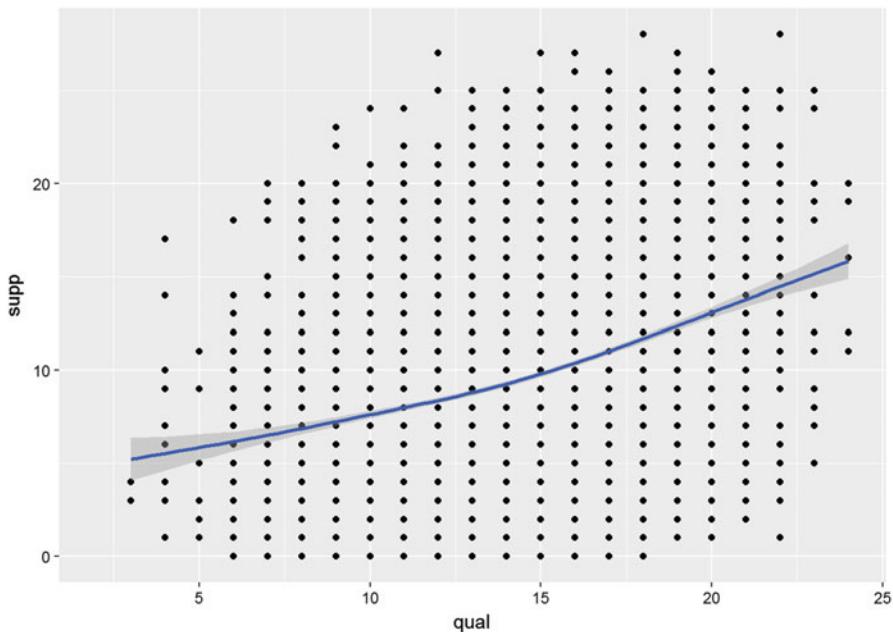


Fig. 4.14 Plotting the bivariate trend along with its confidence limits

```
plot.2 <- qplot(qual, supp, data = data1, geom = c("point", "smooth"))
print(plot.2)
```

You can also use the human height and weight dataset or the knee pain dataset to illustrate some interesting scatter plots.

4.4.2 Jitter Plot

Jitter plots can help us deal with the complexity issues when we have many points in the data. The function we will be using is in package `ggplot2` is called `position_jitter()`.

Let's use the earthquake data for this example. We will compare the differences with and without the `position_jitter()` function (Figs. 4.15 and 4.16).

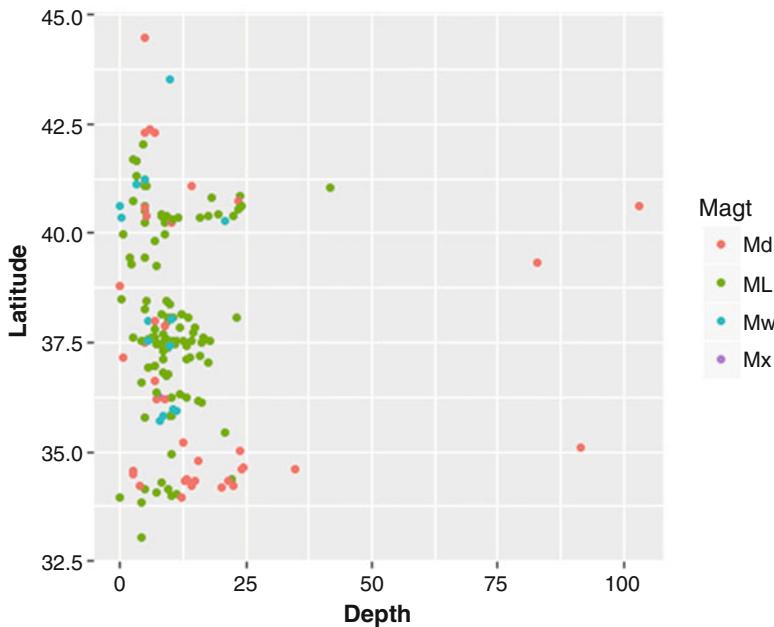


Fig. 4.15 Jitter plot of magnitude type against depth and latitude (Earthquake dataset)

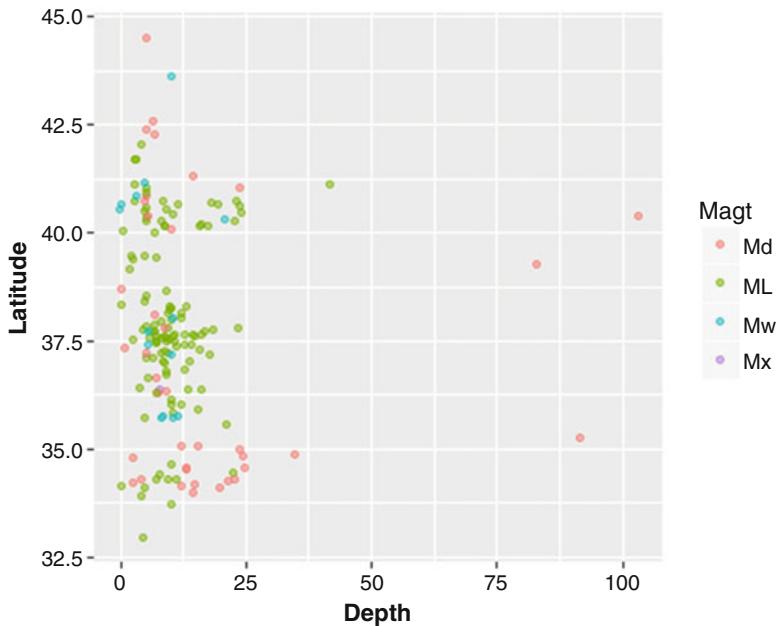


Fig. 4.16 A lower opacity jitter plot of magnitude type against depth and latitude

```

# library("xml2"); library("rvest")
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_021708_Earthquakes")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...
earthquake <- html_table(html_nodes(wiki_url, "table"))[[2]])

plot6.1<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+geom_point()
plot6.2<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+geom_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)
print(plot6.1)

print(plot6.2)

```

Note that with option `alpha=0.5` the “crowded” places are darker than the places with only one data point. Sometimes, we need to add text to these points, i.e., add label in `aes` or add `geom_text`. The result may look messy (Fig. 4.17).

```

ggplot(earthquake, aes(Depth, Latitude, group=Magt,
color=Magt, label=rownames(earthquake)))+
geom_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)+geom_text()

```

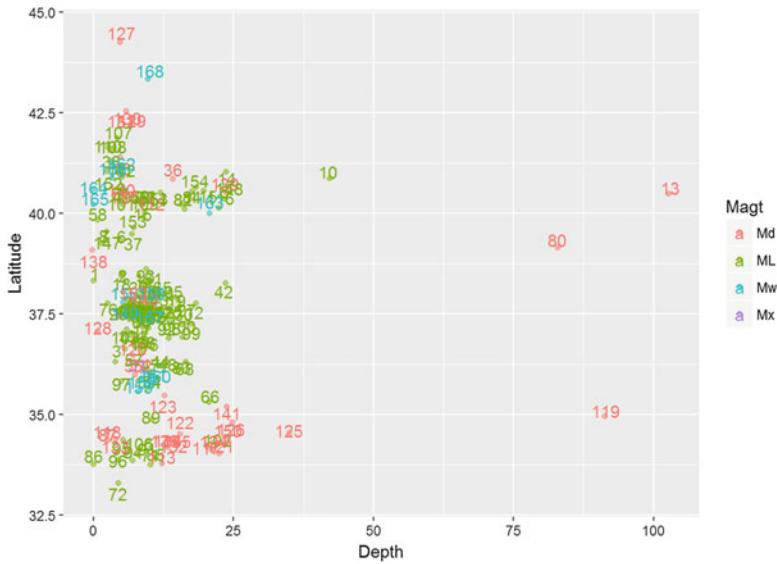


Fig. 4.17 Another version of the jitter plot of magnitude type explicitly listing the Earthquake ID label

Let's try to fix the overlap of points and labels. We need to add `check_overlap` in `geom_text` and adjust the positions of the text labels with respect to the points (Figs. 4.18 and 4.19).

```
ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt,
  label=rownames(earthquake))+
  geom_point(position = position_jitter(w = 0.3, h = 0.3), alpha=0.5)+
```

```
# Or you can simply use the text to denote the positions of points.
ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt,
  label=rownames(earthquake)))+
  geom_text(check_overlap = T, vjust = 0, nudge_y = 0.5, size = 2, angle = 45)
```

4.4.3 Bar Plots

Bar plots, or bar charts, represent group data with rectangular bars. There are many variants of bar charts for comparison among categories. Typically, either horizontal or vertical bars are used where one of the axes shows the compared categories and the other axis represents a discrete value. It's possible, and sometimes desirable, to plot bar graphs including bars clustered by groups.

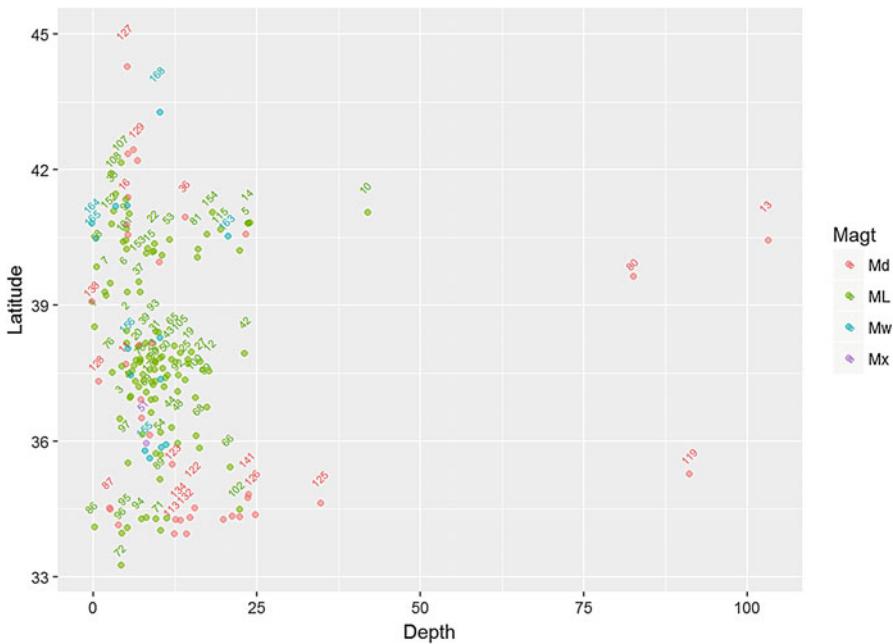


Fig. 4.18 Yet another version of the previous jitter plot illustrating label specifications

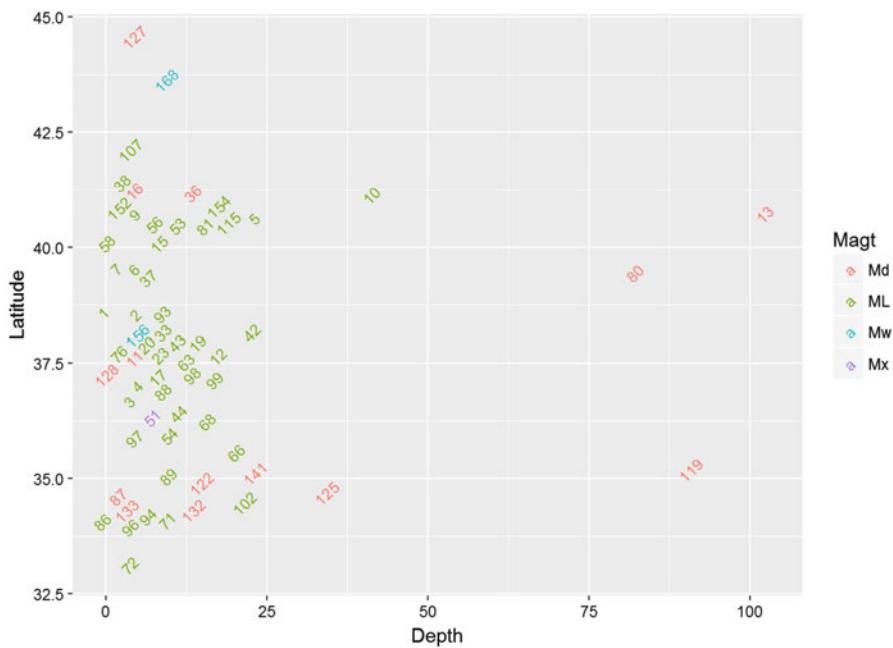


Fig. 4.19 This jitter plot suppresses the scatter point bubbles in favor of ID labels

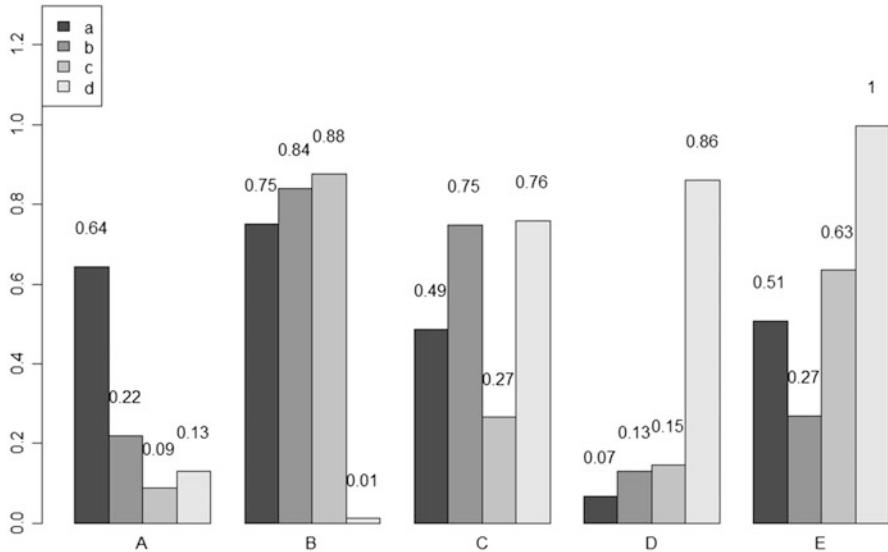


Fig. 4.20 Example of a labeled boxplot using simulated data with grouping categorical labels

In R, we have the `barplot()` function to generate bar plots. The input for `barplot()` is either a vector or a matrix (Fig. 4.20).

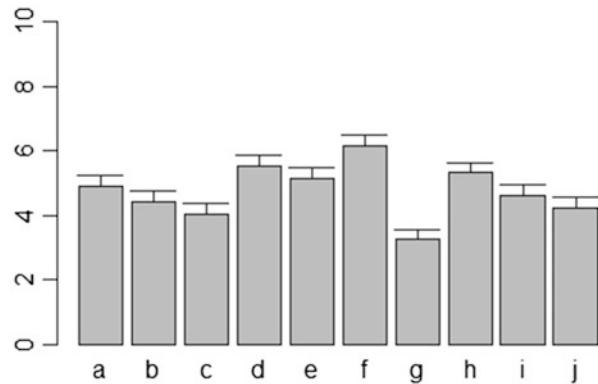
```
x <- matrix(runif(50), ncol=5, dimnames=list(letters[1:10], LETTERS[1:5]))
x

##          A         B         C         D         E
## a 0.64397479 0.75069788 0.4859278 0.068299279 0.5069665
## b 0.21981304 0.84028392 0.7489431 0.130542241 0.2694441
## c 0.08903728 0.87540556 0.2656034 0.146773063 0.6346498
## d 0.13075121 0.01106876 0.7586781 0.860316695 0.9976566
## e 0.87938851 0.04156918 0.1960069 0.949276015 0.5050743
## f 0.65204025 0.21135891 0.3774320 0.896443296 0.9332330
## g 0.02814806 0.72618285 0.5603189 0.113651731 0.1912089
## h 0.13106307 0.79411904 0.4526415 0.793385952 0.4847625
## i 0.15759514 0.63369297 0.8861631 0.004317772 0.6341256
## j 0.47347613 0.14976052 0.5887866 0.698139910 0.2023031

barplot(x[1:4, ], ylim=c(0, max(x[1:4, ])+0.3), beside=TRUE, Legend.text = letters[1:4],
        args.Legend = list(x = "topleft"))
text(labels=round(as.vector(as.matrix(x[1:4, ])), 2), x=seq(1.5, 21, by=1) +
rep(c(0, 1, 2, 3, 4), each=4), y=as.vector(as.matrix(x[1:4, ]))+0.1)
```

It may require some creativity to add value labels on each bar. First, let's specify the location on the x-axis `x=seq(1.5, 21, by=1) + rep(c(0, 1, 2, 3, 4), each=4)`. In this example there are 20 bars. The x location for middle of the first bar is 1.5 (there is one empty space before the first bar). The middle of the last bar is

Fig. 4.21 Statistical barplot showing point-estimates and their error limits (simulated data)



24.5. `seq(1.5, 21, by=1)` starts at 1.5 and creates 20 bars that end with `x=21`. Then, we use `rep(c(0, 1, 2, 3, 4), each=4)` to add 0 to the first group, 1 to the second group, and so forth. Thus, we have the desired positions on the x-axis. The y-axis positions are obtained just by adding 0.1 to each bar height.

We can also add standard deviations to the means on the bars. To do this, we need to use the `arrows()` function and the option `angle = 90`, the result is shown on Fig. 4.21.

```
bar <- barplot(m <- rowMeans(x) * 10, ylim=c(0, 10))
stdev <- sd(t(x[1:4, ]))
arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```

Let's look at a more complex example. We will utilize the dataset `Case_04_ChildTrauma` for illustration. This case study examines associations between post-traumatic psychopathology and service utilization by trauma-exposed children.

```
data2 <- read.table('https://umich.instructure.com/files/399129/download?down
nLoad_frd=1', header=T)
attach(data2)
head(data2)

##   id sex age ses race traumatype ptsd dissoc service
## 1  1   1   6   0 black sexabuse   1     1    17
## 2  2   1  14   0 black sexabuse   0     0    12
## 3  3   0   6   0 black sexabuse   0     1     9
## 4  4   0   11   0 black sexabuse   0     1    11
## 5  5   1   7   0 black sexabuse   1     1    15
## 6  6   0   9   0 black sexabuse   1     0     6
```

We have two character variables. Our goal is to draw a bar plot comparing the means of `age` and `service` among different races in this study, and we want to add standard deviation to each bar. The first thing to do is to delete the two character columns. Remember, the input for `barplot()` is a numerical vector or a matrix.

However, we will need race information for the categorical classification. Thus, we will store race in a different variable.

```
data2.sub <- data2[, c(-5, -6)]
data2<-data2[, -6]
```

We are now ready to separate the groups and compute the group means.

```
data2.matrix <- as.data.frame(data2)
Blacks <- data2[which(data2$race=="black"), ]
Other <- data2[which(data2$race=="other"), ]
Hispanic <- data2[which(data2$race=="hispanic"), ]
White <- data2[which(data2$race=="white"), ]

B <- c(mean(Blacks$age), mean(Blacks$service))
O <- c(mean(Other$age), mean(Other$service))
H <- c(mean(Hispanic$age), mean(Hispanic$service))
W <- c(mean(White$age), mean(White$service))

x <- cbind(B, O, H, W)
x

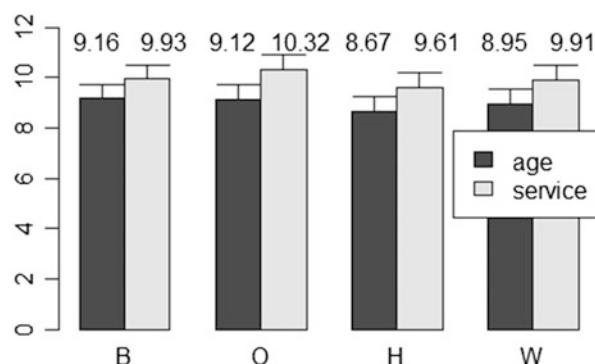
##           B         O         H         W
## [1,] 9.165  9.12  8.67 8.950000
## [2,] 9.930 10.32  9.61 9.911667
```

Now that we have a numerical matrix for the means, we can compute a second order statistics, standard deviation, and plot it along with the means, to illustrate the amount of dispersion for each variable (Fig. 4.22).

```
bar <- barplot(x, ylim=c(0, max(x)+2.0), beside=TRUE,
Legend.text = c("age", "service") , args.legend = list(x = "right"))
text(labels=round(as.vector(as.matrix(x)), 2),
      x=seq(1.4, 21, by=1.5), #y=as.vector(as.matrix(x[1:2, ]))+0.3)
      y=11.5)

m <- x; stdev <- sd(t(x))
arrows(bar, m, bar, m + stdev, length=0.15, angle = 90)
```

Fig. 4.22 Barplot showing point-estimates and their error limits (Child Trauma dataset)



Here, we want the y margin to be a little higher than the greatest value (`ylim=c(0, max(x)+2.0)`) because we need to leave space for value labels. The plot shows that Hispanic trauma-exposed children may be younger, in terms of average age, and less likely to utilize services like primary care, emergency room, outpatient therapy, outpatient psychiatrist, etc.

Another way to plot bar plots is to use `ggplot()` in the `ggplot` package. This kind of bar plot is quite different from the one we introduced previously. It displays the counts of character variables rather than the means of numerical variables. It takes the values from a `data.frame`. Unlike `barplot()`, drawing bar plots using `ggplot2` requires that the character variables remain in the original data frame (Fig. 4.23).

```
library(ggplot2)
data2 <- read.table('https://umich.instructure.com/files/399129/download?down
nLoad_frd=1', header=T)
bar1 <- ggplot(data2, aes(race, fill=race)) + geom_bar() +
  facet_grid(. ~ traumatotype)
print(bar1)
```

This plot helps us compare the occurrence of different types of child-trauma among different races.

4.4.4 Trees and Graphs

In general, a graph is an ordered pair $G = (V, E)$ of vertices (V), i.e., nodes or points, and edges (E), arcs or lines connecting pairs of nodes in V . A tree is a special type of acyclic graph that does not include looping paths. Visualization of graphs is critical in many biosocial and health studies, and we will see menu such examples throughout this textbook.

In Chaps. 10 and 13, we will learn more about how to build tree models and other clustering methods, and in Chap. 23, we will discuss deep learning and neural networks, which have a direct graphical representation.

This section will be focused on displaying tree graphs. We will use a self-efficacy study, `02_Nof1_Data.csv`, for this demonstration.

```
data3<- read.table("https://umich.instructure.com/files/330385/download?down
nLoad_frd=1", sep=",", header = TRUE)
head(data3)

##   ID Day Tx SelfEff SelfEff25  WPSS SocSuppt PMss PMss3 PhyAct
## 1  1   1  1      33      8  0.97  5.00 4.03  1.03    53
## 2  1   2  1      33      8 -0.17  3.87 4.03  1.03    73
## 3  1   3  0      33      8  0.81  4.84 4.03  1.03    23
## 4  1   4  0      33      8 -0.41  3.62 4.03  1.03    36
## 5  1   5  1      33      8  0.59  4.62 4.03  1.03    21
## 6  1   6  1      33      8 -1.16  2.87 4.03  1.03     0
```

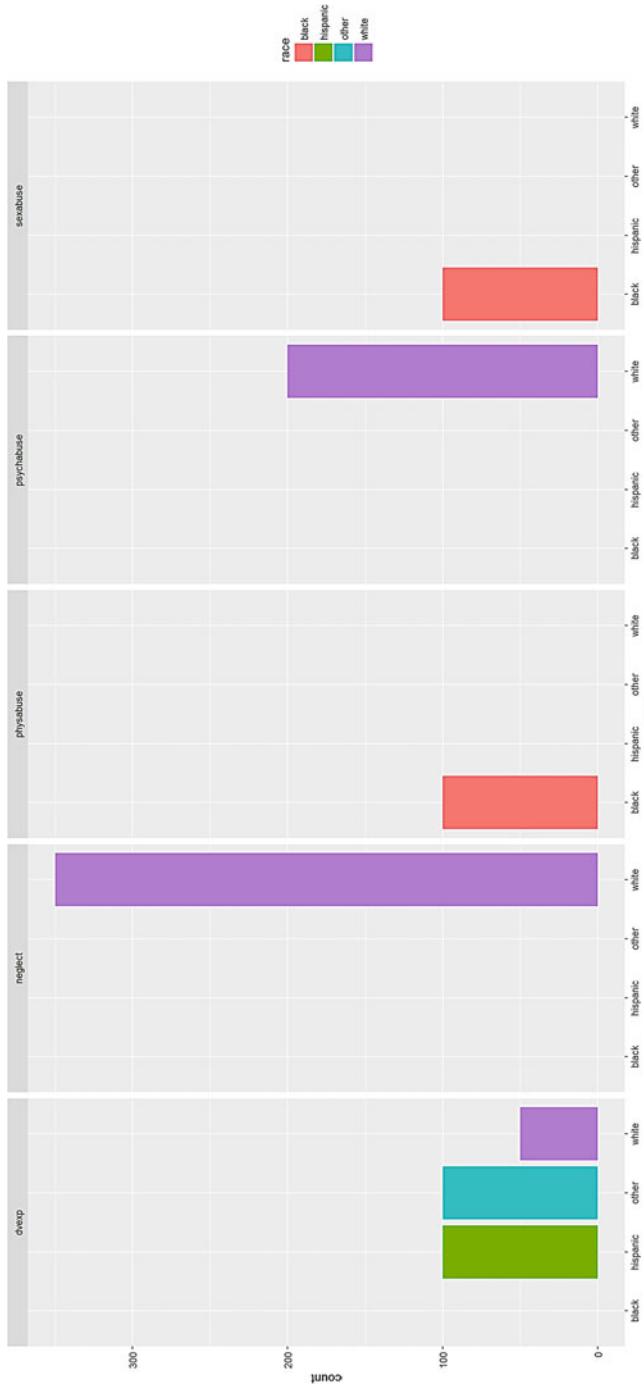
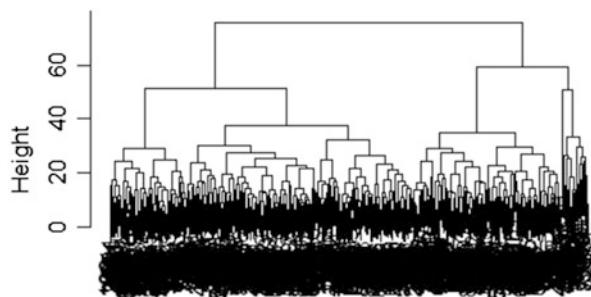


Fig. 4.23 Barplot of counts for different types of child trauma by race (color label)

Fig. 4.24 Hierarchical clustering dendrogram of the 900 self-efficacy records of 30 participants including the nine features tracked over a month



We will use `hclust` to build the hierarchical cluster model. `hclust` takes only inputs that have dissimilarity structure as produced by `dist()`. Also, we use the `ave` method for agglomeration, see the tree graph on Fig. 4.24.

```
hc<-hclust(dist(data3), method='ave')
par (mfrow=c(1, 1))
plot(hc)
```

When we specify no limit for the maximum cluster groups, we will get the graph, on Fig. 4.24, which is not easy to interpret. Luckily, `cutree` will help us limit the cluster numbers. `cutree()` takes a `hclust` object and returns a vector of group indicators for all observations.

```
require(graphics)
mem <- cutree(hc, k = 10)

# mem; # to print the hierarchical tree labels for each case
# which(mem==5) # to identify which cases belong to class/cluster 5
#To see the number of Subjects in which cluster:
# table(cutree(hc, k=5))
```

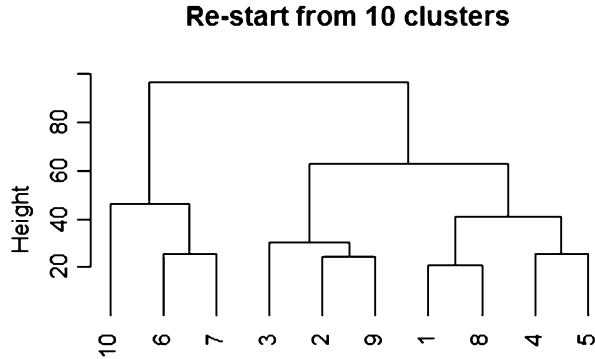
Using a for loop, we can get the mean of each variable within groups.

```
cent <- NULL
for(k in 1:10){
  cent <- rbind(cent, colMeans(data3[mem == k, , drop = FALSE]))
}
```

Now, we can plot a new tree graph with ten groups. Using the `members=table(mem)` option, the matrix is taken to be a dissimilarity matrix between clusters, instead of dissimilarities between singletons, and `members` represents the number of observations per cluster (Fig. 4.25).

```
hc1 <- hclust(dist(cent), method = "ave", members = table(mem))
plot(hc1, hang = -1, main = "Re-start from 10 clusters")
```

Fig. 4.25 A ten-cluster hierarchical dendrogram of the same dataset as before



4.4.5 Correlation Plots

The `corrplot` package enables the graphical display of correlation matrices, confidence intervals, and other plots showing matrix reordering. There are seven visualization methods (parameter methods) in `corrplot` package, named "circle", "square", "ellipse", "number", "shade", "color", and "pie".

Let's use `03_NC_SNP_ROI_Assoc_P_values.csv` again to investigate the associations among SNPs using correlation plots.

The `corrplot()` function we will be using only accepts correlation matrices. So, we need to first obtain the correlation matrix of our data first using the `cor()` function.

```
# install.packages("corrplot")
library(corrplot)
NC_Associations_Data <- read.table("https://umich.instructure.com/files/33091/download?download_frd=1", header=TRUE, row.names=1, sep=",", dec=".")
M <- cor(NC_Associations_Data)
M[1:10, 1:10]

##          P2          P5          P9          P12          P13
## P2  1.0000000 -0.05976123  0.99999944 -0.05976123  0.21245299
## P5 -0.05976123  1.00000000 -0.05976131 -0.02857143  0.56024640
## P9  0.99999944 -0.05976131  1.00000000 -0.05976131  0.21248635
## P12 -0.05976123 -0.02857143 -0.05976131  1.00000000 -0.05096471
## P13  0.21245299  0.56024640  0.21248635 -0.05096471  1.00000000
## P14 -0.05976123  1.00000000 -0.05976131 -0.02857143  0.56024640
## P15 -0.08574886  0.69821536 -0.08574898 -0.04099594  0.36613665
## P16 -0.08574886  0.69821536 -0.08574898 -0.04099594  0.36613665
## P17 -0.05976123 -0.02857143 -0.05976131 -0.02857143 -0.05096471
## P18 -0.05976123 -0.02857143 -0.05976131 -0.02857143 -0.05096471
##          P14          P15          P16          P17          P18
## P2 -0.05976123 -0.08574886 -0.08574886 -0.05976123 -0.05976123
## P5  1.00000000  0.69821536  0.69821536 -0.02857143 -0.02857143
## P9 -0.05976131 -0.08574898 -0.08574898 -0.05976131 -0.05976131
## P12 -0.02857143 -0.04099594 -0.04099594 -0.02857143 -0.02857143
## P13  0.56024640  0.36613665  0.36613665 -0.05096471 -0.05096471
## P14  1.00000000  0.69821536  0.69821536 -0.02857143 -0.02857143
## P15  0.69821536  1.00000000  1.00000000 -0.04099594 -0.04099594
## P16  0.69821536  1.00000000  1.00000000 -0.04099594 -0.04099594
## P17 -0.02857143 -0.04099594 -0.04099594  1.00000000 -0.02857143
## P18 -0.02857143 -0.04099594 -0.04099594 -0.02857143  1.00000000
```

We will illustrate alternative correlation plots using the `corrplot` function in Figs. 4.26, 4.27, 4.28, 4.29, 4.30, and 4.31.

```
corrplot(M, method = "circle", title = "circle", tl.cex = 0.5, tl.col = 'black',
  mar=c(1, 1, 1, 1))
# par specs c(bottom, left, top, right) which gives the margin size
# specified in inches
corrplot(M, method = "square", title = "square", tl.cex = 0.5, tl.col =
  'black', mar=c(1, 1, 1, 1))
corrplot(M, method = "ellipse", title = "ellipse", tl.cex = 0.5, tl.col =
  'black', mar=c(1, 1, 1, 1))
corrplot(M, method = "pie", title = "pie", tl.cex = 0.5, tl.col = 'black',
  mar=c(1, 1, 1, 1))
corrplot(M, type = "upper", tl.pos = "td",
  method = "circle", tl.cex = 0.5, tl.col = 'black',
  order = "hclust", diaa = FALSE, mar=c(1, 1, 0, 1))
corrplot.mixed(M, number.cex = 0.6, tl.cex = 0.6)
```

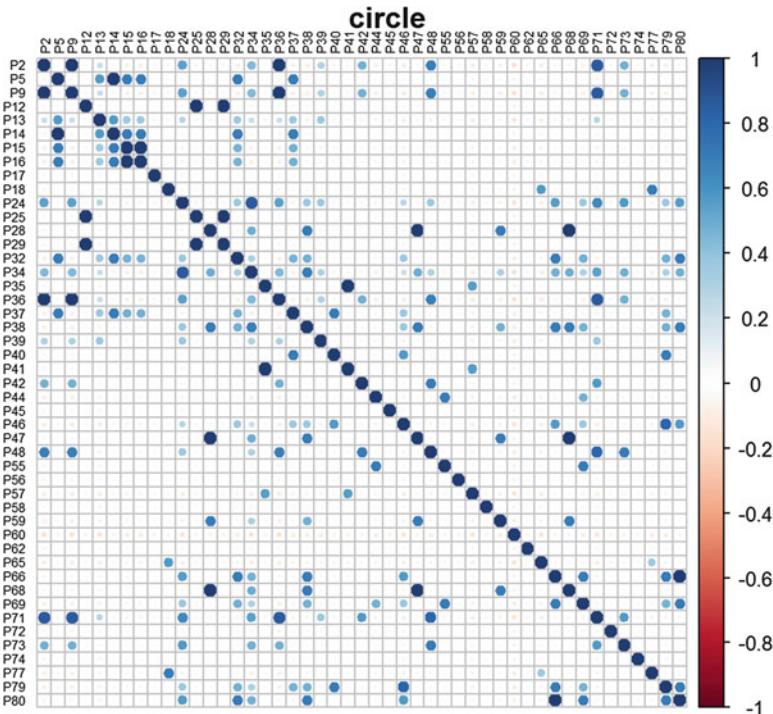


Fig. 4.26 Correlation plot of regional brain volumes of the healthy normal controls using circles

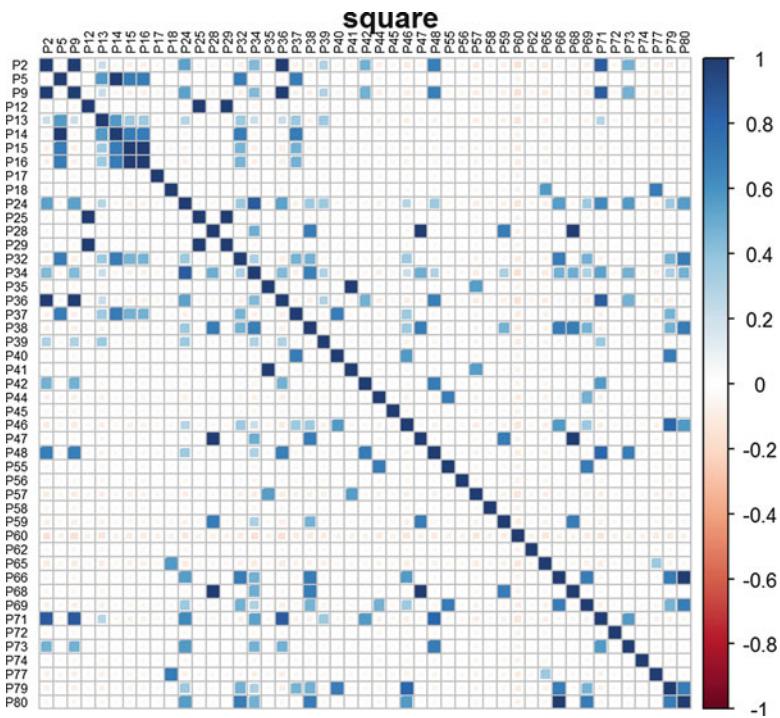


Fig. 4.27 The same correlation plot of regional NC brain volumes using squares

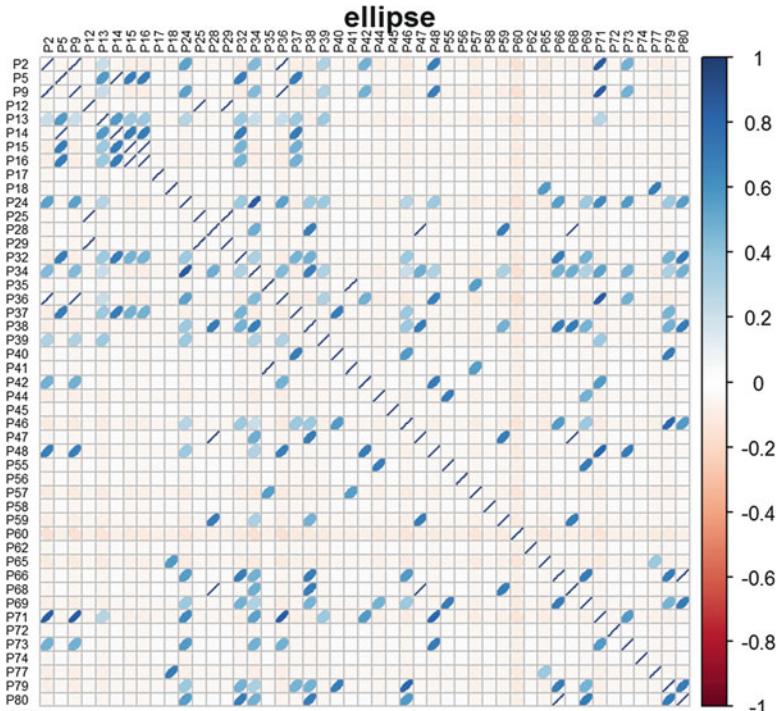


Fig. 4.28 The same correlation plot of regional NC brain volumes using ellipses

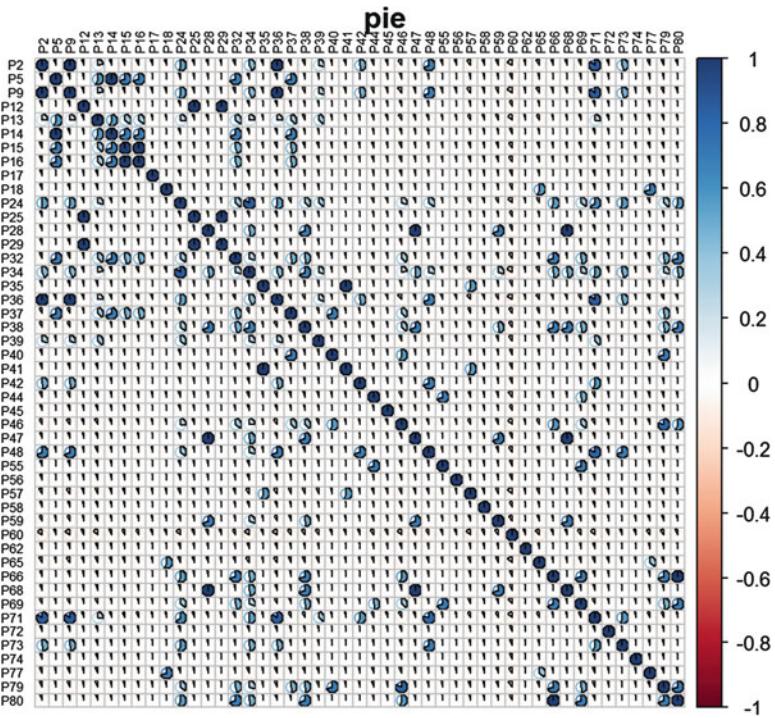


Fig. 4.29 The same correlation plot of regional NC brain volumes using pie segments

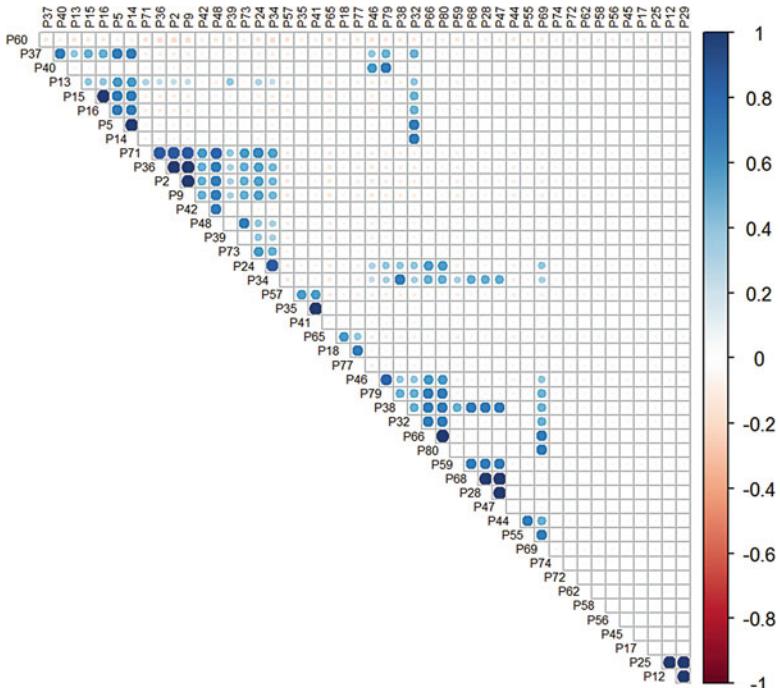


Fig. 4.30 Upper diagonal correlation plot of regional NC brain volumes using circles

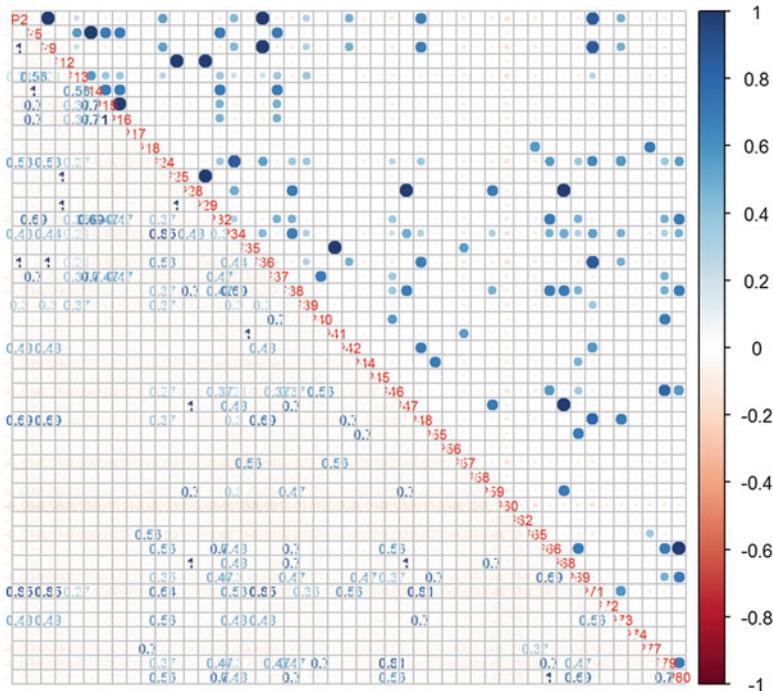


Fig. 4.31 Mixed correlation plot of regional NC brain volumes using circles and numbers

In the figures above, different shades of colors represent low-and-high correlations of the two variables corresponding to the x and y indices.

4.5 Relationships

4.5.1 Line Plots Using *ggplot*

Line charts display a series of data points (e.g., observed intensities (Y) over time (X)) by connecting them with straight-line segments. These can be used to either track temporal changes of a process or compare the trajectories of multiple cases, time series, or subjects over time, space, or state.

In this section, we will utilize the Earthquakes dataset on SOCR website. It records information about earthquakes that occurred between 1969 and 2007 with magnitudes larger than 5 on the Richter scale.

```
# library("xml2"); library("rvest")
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_021708_Earthquakes")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top"
...
earthquake<- html_table(html_nodes(wiki_url, "table"))[[2]]
```

In this dataset, we group the data by Magt (magnitude type). We will draw a “Depth vs. Latitude” line plot from this dataset. The function we are using is called `ggplot()` under `ggplot2`. The input type for this function is a data frame and `aes()` specifies aesthetic mappings of how variables in the data are mapped to visual properties (aesthetics) of the `geom` objects, e.g., lines (Fig. 4.32).

```
library(ggplot2)
plot4<-ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))+
  geom_line()
print(plot4)
```

There are two important components in the script. The first part, `ggplot(earthquake, aes(Depth, Latitude, group=Magt, color=Magt))`, specifies the setting of the plot: dataset, group, and color. The second part specifies that we are going to draw lines between data points. In later chapters, we will frequently use package `ggplot2` whose generic structure always involves concatenating function calls like `function1+function2+....`

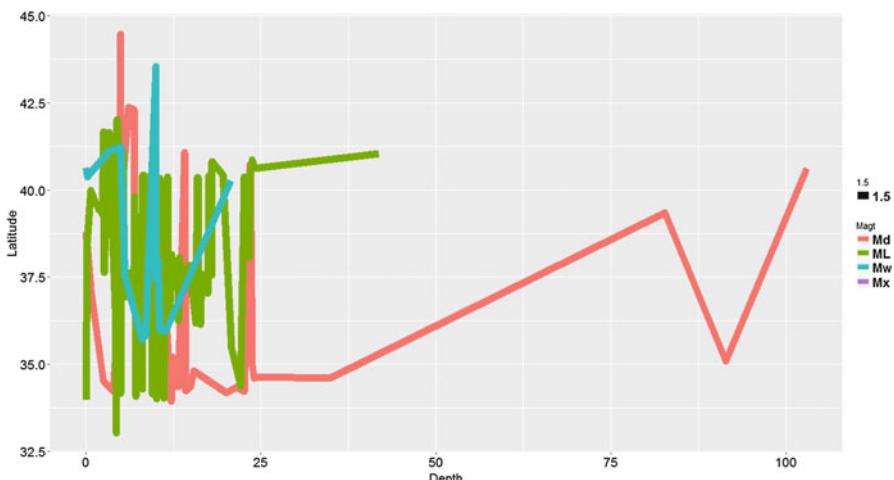


Fig. 4.32 Line plot of Earthquake magnitude type by its ground depth and latitude

4.5.2 Density Plots

We can visualize the distribution of different variables using density plots.

The following segment of R code plots the distribution for latitude among different earthquake magnitude types. Also, it uses the `ggplot()` function combined with `geom_density()` (Fig. 4.33).

```
# library("ggplot2")
plot5<-ggplot(earthquake, aes(Latitude, group=Magt, newsize=2))+  
  geom_density(aes(color=Magt), size = 2) +  
  theme(legend.position = 'right',  
        Legend.text = element_text(color= 'black', size = 12, face = 'bold'),  
        Legend.key = element_rect(size = 0.5, linetype='solid'),  
        Legend.key.size = unit(1.5, 'lines'))  
print(plot5)  
# table(earthquake$Magt) # to see the distribution of magnitude types
```

Note the green magt type (Local (ML) earthquakes) peaks at latitude 37.5, which represents 37–38° North, near San Francisco, California.

4.5.3 Distributions

Probability distribution plots depict the characteristics of the underlying process that can be used to contrast and compare the shapes of distributions as proxy of the corresponding natural phenomena. For univariate, bivariate, and multivariate processes, the distribution plots are drawn as curves, surfaces, or manifolds, respectively. These plots may be used to inspect areas under the distribution plot that correspond to either probabilities or data values. The Distributome Cauchy distribution calculator and the SOCR 2D bivariate Normal Distribution plot provide simple examples of distribution plots in 1D and 2D, respectively (Fig. 4.34).

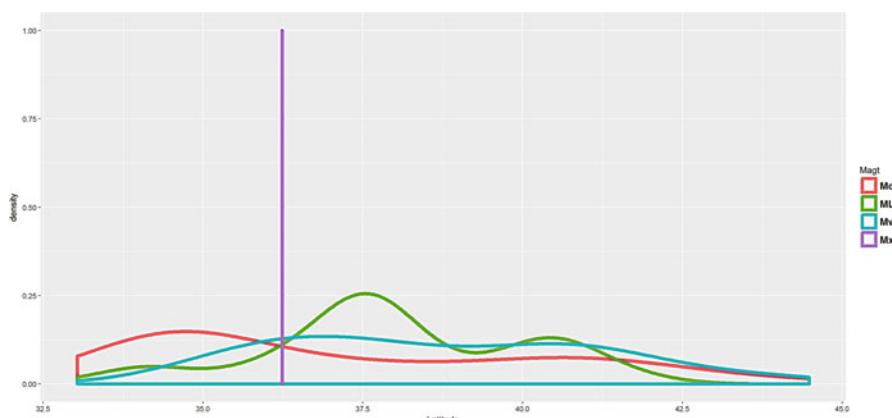
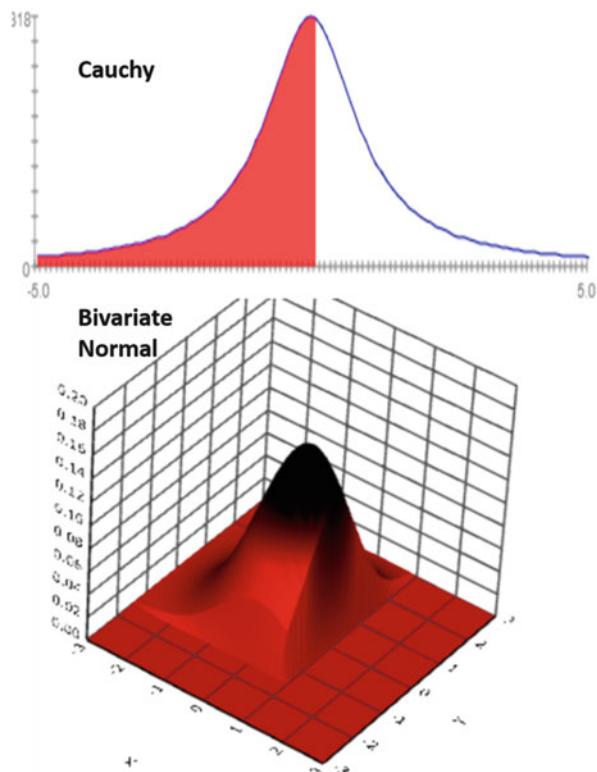


Fig. 4.33 Density plot of Earthquakes according to their magnitude types and latitude location

Fig. 4.34 Univariate and bivariate probability distribution calculators (Distributome Project)



4.5.4 2D Kernel Density and 3D Surface Plots

Density estimation is the process of using observed data to compute an estimate of the underlying process' probability density function. There are several approaches to obtain density estimation, but the most basic technique is to use a rescaled histogram.

Plotting 2D Kernel Density and 3D Surface plots is very important and useful in multivariate exploratory data analytics.

We will use `plot_ly()` function under `plotly` package, which requires a data frame input.

To create a surface plot, we use two vectors: x and y with length m and n respectively. We also need a matrix: z of size $m \times n$. This z matrix is created from matrix multiplication between x and y .

To plot the 2D Kernel Density estimation plot we will use the eruptions data from the “Old Faithful” geyser in Yellowstone National Park, Wyoming, stored in R as `geyser`. Also, the `kde2d()` function is needed for 2D kernel density estimation.

```

kd <- with(MASS::geyser, MASS::kde2d(duration, waiting, n = 50))
kd$x[1:5]
## [1] 0.8333333 0.9275510 1.0217687 1.1159864 1.2102041
kd$y[1:5]
## [1] 43.00000 44.32653 45.65306 46.97959 48.30612
kd$z[1:5, 1:5]
## [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 9.068691e-13 4.238943e-12 1.839285e-11 7.415672e-11 2.781459e-10
## [2,] 1.814923e-12 8.473636e-12 3.671290e-11 1.477410e-10 5.528260e-10
## [3,] 3.428664e-12 1.599235e-11 6.920273e-11 2.780463e-10 1.038314e-09
## [4,] 6.114498e-12 2.849475e-11 1.231748e-10 4.942437e-10 1.842547e-09
## [5,] 1.029643e-11 4.793481e-11 2.070127e-10 8.297218e-10 3.088676e-09

```

Here $z=t(x) \otimes y$ and we apply `plot_ly` to the list `kd` via the `with()` function (Fig. 4.35).

```

library(plotly)
with(kd, plot_ly(x=x, y=y, z=z, type="surface"))

```

Note that we used the option "surface".

For 3D surfaces, we have a built-in dataset in R called `volcano`. It records the volcano height at location x , y (longitude, latitude). Because z is always made from x and y , we can simply specify z to get the complete surface plot (Fig. 4.36).

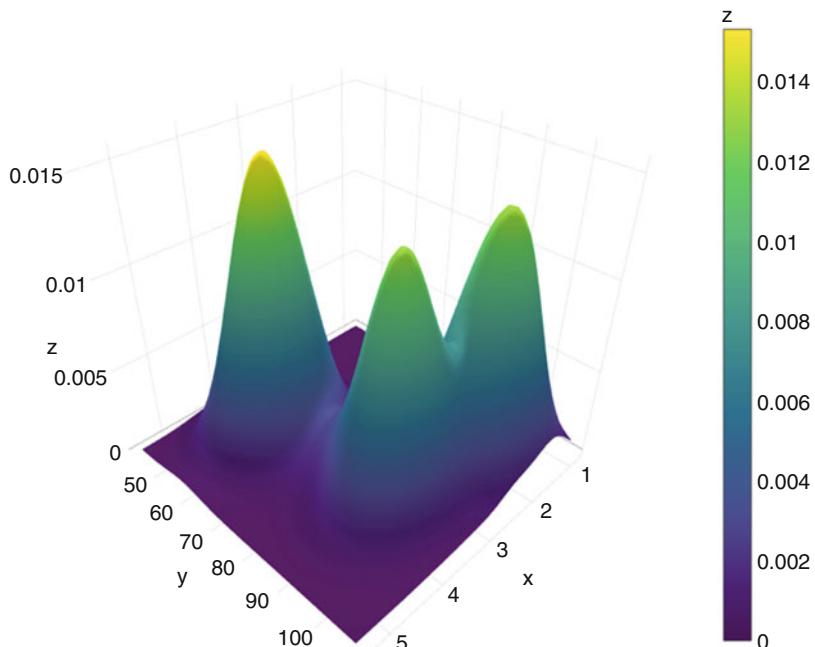


Fig. 4.35 Interactive surface plot of kernel density for the Old Faithful geyser eruptions

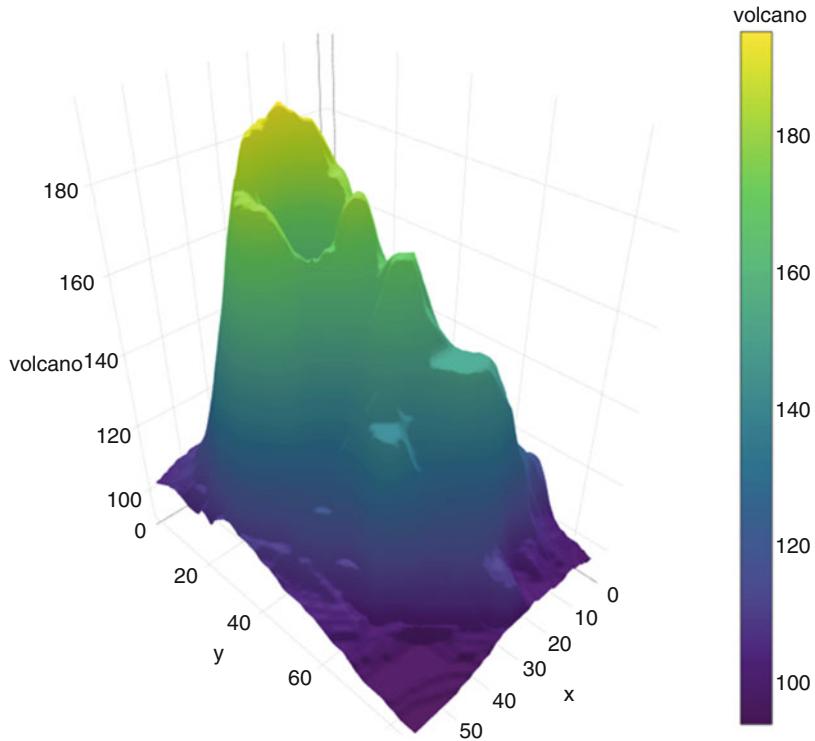


Fig. 4.36 Interactive surface plot of kernel density for the R volcano dataset

```
volcano[1:10, 1:10]
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,] 100 100 101 101 101 101 101 100 100 100
## [2,] 101 101 102 102 102 102 102 101 101 101
## [3,] 102 102 103 103 103 103 103 102 102 102
## [4,] 103 103 104 104 104 104 104 103 103 103
## [5,] 104 104 105 105 105 105 105 104 104 103
## [6,] 105 105 105 106 106 106 106 105 105 104
## [7,] 105 106 106 107 107 107 107 106 106 105
## [8,] 106 107 107 108 108 108 108 107 107 106
## [9,] 107 108 108 109 109 109 109 108 108 107
## [10,] 108 109 109 110 110 110 110 109 109 108
```

```
plot_ly(z=volcano, type="surface")
```

4.5.5 Multiple 2D Image Surface Plots

Let's look at another example using a 2D brain image (Fig. 4.37).

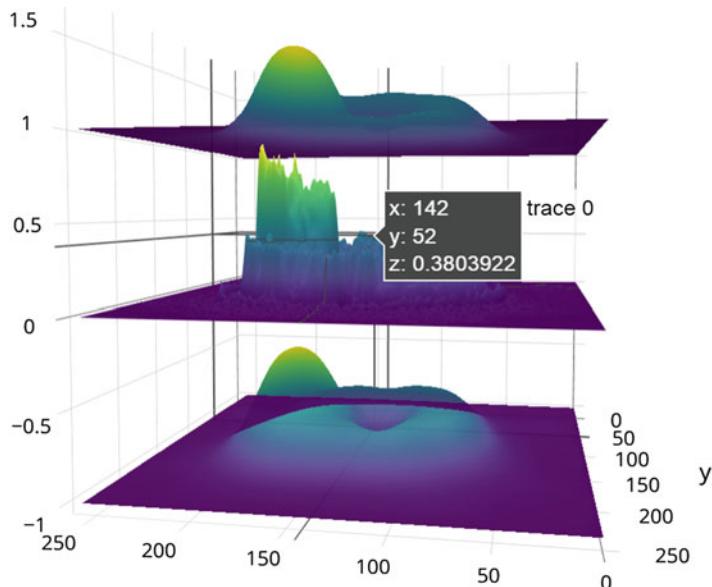


Fig. 4.37 Interactive surface plot of kernel density for the 2D brain imaging data

```
#install.packages("jpeg") ## if necessary
library(jpeg)

# Get an image file downloaded (default: MRI_ImageHematoma.jpg)
img_url <- "https://umich.instructure.com/files/1627149/download?download_file=1"
img_file <- tempfile(); download.file(img_url, img_file, mode="wb")
img <- readJPEG(img_file)
file.info(img_file)

file.remove(img_file) # cleanup
## [1] TRUE

img <- img[, , 1] # extract the first channel (from RGB intensity spectrum)
# as a univariate 2D array

# install.packages("spatstat")
# package spatstat has a function blur() that applies a Gaussian blur
library(spatstat)

img_s <- as.matrix(blur(as.im(img), sigma=10)) # the smoothed version of the
# image

z2 <- img_s + 1 # abs(rnorm(1, 1, 1)) # Upper confidence surface
z3 <- img_s - 1 # abs(rnorm(1, 1, 1)) # Lower confidence limit

# Plot the image surfaces
p <- plot_ly(z=img, type="surface", showscale=FALSE) %>%
  add_trace(z=z2, type="surface", showscale=FALSE, opacity=0.98) %>%
  add_trace(z=z3, type="surface", showscale=FALSE, opacity=0.98)
p # Plot the mean-surface along with lower and upper confidence services.
```

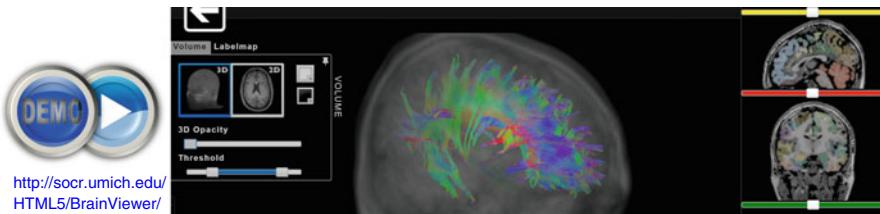


Fig. 4.38 Live demo: interactive brain viewer

The DSPA Online appendix provides additional details on shape representation, modeling, and computing on surfaces and manifolds.

4.5.6 3D and 4D Visualizations

Many datasets have intrinsic multi-dimensional characteristics. For instance, the human body is a 3D solid of matter (three spatial dimensions can be used to describe the position of every component, e.g., sMRI volume) that changes over time (the fourth dimension, e.g., fMRI hypervolumes).

The SOCR BrainViewer shows how to use a web-browser to visualize 2D cross-sections of 3D volumes, display volume-rendering, and show 1D (e.g., 1-manifold curves embedded in 3D) and 2D (e.g., surfaces, shapes) models jointly into the same 3D scene (Fig. 4.38).

We will now illustrate an example of 3D/4D visualization in R using the packages `brainR` and `rgl`.

```
# install.packages("brainR") ## if necessary
require(brainR)

# Test data: http://socr.umich.edu/HTML5/BrainViewer/data/TestBrain.nii.gz
brainURL <- "http://socr.umich.edu/HTML5/BrainViewer/data/TestBrain.nii.gz"
brainFile <- file.path(tempdir(), "TestBrain.nii.gz")
download.file(brainURL, dest=brainFile, quiet=TRUE)
brainVolume <- readNIfTI(brainFile, reorient=FALSE)

brainVolDims <- dim(brainVolume); brainVolDims
## [1] 181 217 181

# try different levels at which to construct contour surfaces (10 fast)
# lower values yield smoother surfaces # see ?contour3d
contour3d(brainVolume, level = 20, alpha = 0.1, draw = TRUE)

# multiple levels may be used to show multiple shells
# "activations" or surfaces like hyper-intense white matter
# This will take 1-2 minutes to rend!
contour3d(brainVolume, level = c(10, 120), alpha = c(0.3, 0.5),
          add = TRUE, color=c("yellow", "red"))
```

```
# create text for orientation of right/left
text3d(x=brainVolDims[1]/2, y=brainVolDims[2]/2, z = brainVolDims[3]*0.98,
text="Top")
text3d(x=brainVolDims[1]*0.98, y=brainVolDims[2]/2, z = brainVolDims[3]/2,
text="Right")

### render this on a webpage and view it!
#browseURL(paste("file://",
#                  writeWebGL_split(dir= file.path(tempdir(),"webGL"),
#                  template = system.file("my_template.html", package="brainR"),
#                  width=500), sep=""))
```

For 4D fMRI time-series, we can load the hypervolumes similarly and then display them (Figs. 4.39, 4.40, 4.41, 4.42, 4.43, and 4.44):

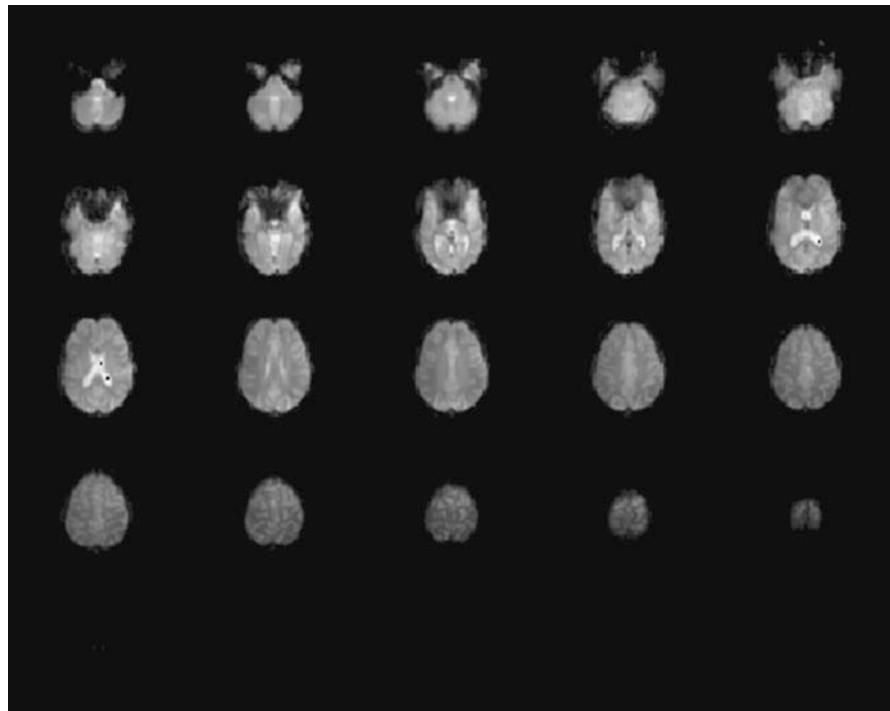


Fig. 4.39 2D cross-sectional (axial) views of the 4D fMRI data

Fig. 4.40 Histogram plot of the fMRI intensities

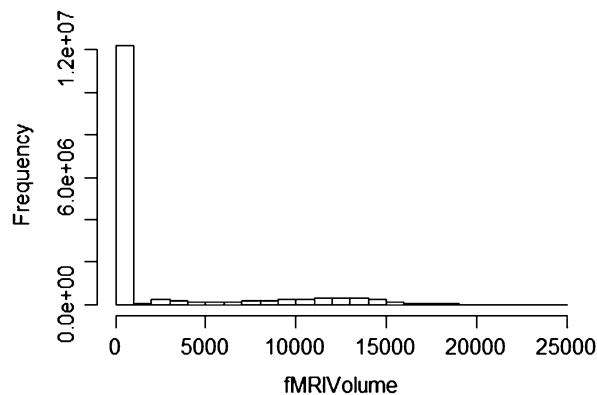


Fig. 4.41 Coronal (top-left), sagittal (top-right), and axial (bottom-left) views of the 4D fMRI data

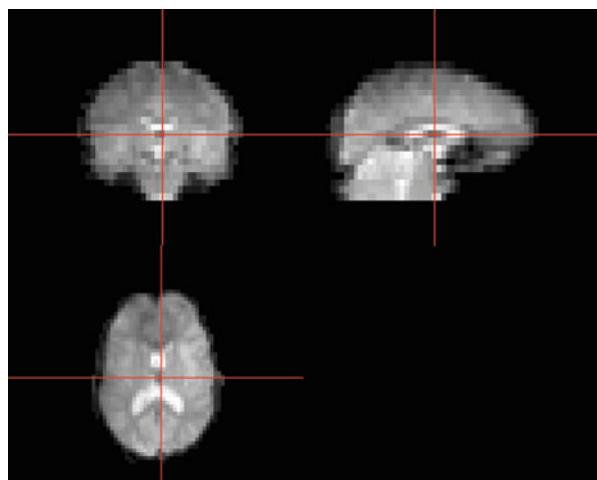
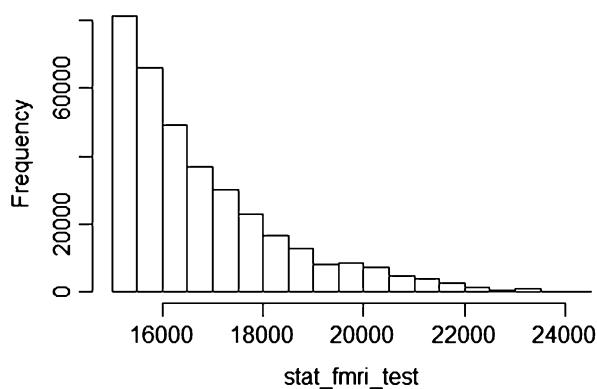


Fig. 4.42 Truncated histogram of the fMRI hyper-volume intensities



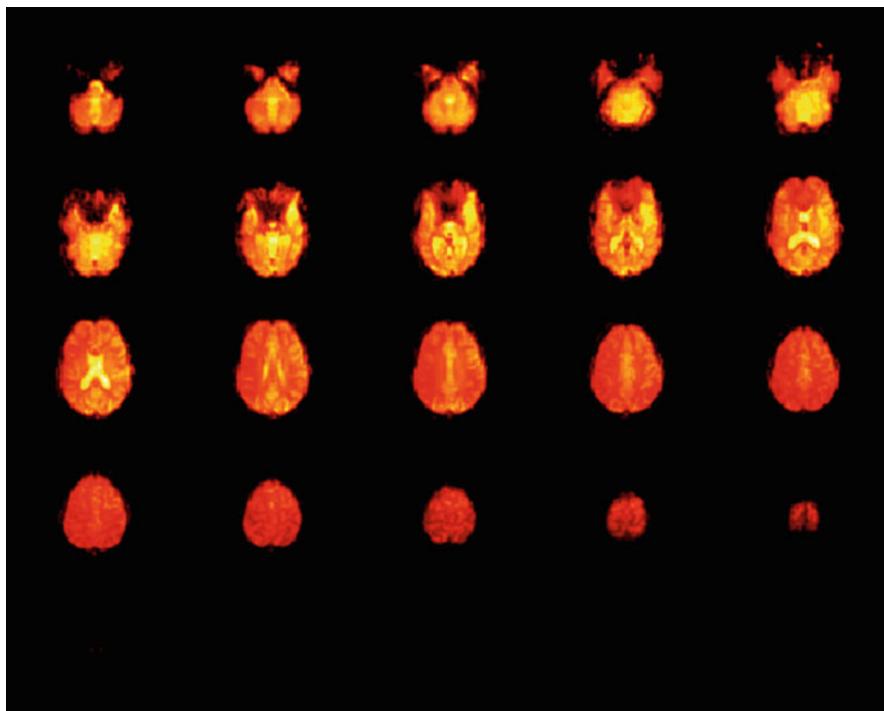


Fig. 4.43 Intensities of the fifth timepoint epoch of the 4D fMRI time series

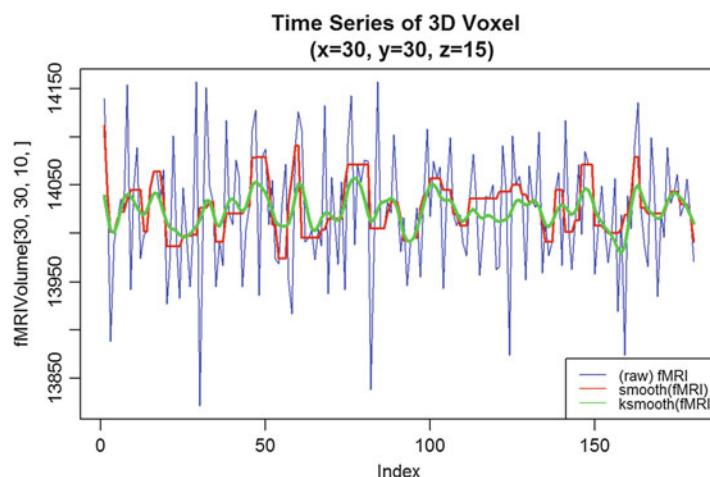


Fig. 4.44 The complete time course of the raw (blue) and two smoothed versions of the fMRI timeseries at one specific voxel location (30, 30, 15)

```

# See examples here: https://cran.r-project.org/web/packages/oro.nifti/vignettes/nifti.pdf
# and here: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0089470

fMRIURL <- "http://socr.umich.edu/HTML5/BrainViewer/data/fMRI_FilteredData_4D.nii.gz"
fMRIFile <- file.path(tempdir(), "fMRI_FilteredData_4D.nii.gz")
download.file(fMRIURL, dest=fMRIFile, quiet=TRUE)
(fMRIVolume <- readNIfTI(fMRIFile, reorient=FALSE))

## NIfTI-1 format
## Type : nifti
## Data Type : 4 (INT16)
## Bits per Pixel : 16
## Slice Code : 0 (Unknown)
## Intent Code : 0 (None)
## Oform Code : 1 (Scanner_Anat)
## Sform Code : 0 (Unknown)
## Dimension : 64 x 64 x 21 x 180
## Pixel Dimension : 4 x 4 x 6 x 3
## Voxel Units : mm
## Time Units : sec

# dimensions: 64 x 64 x 21 x 180 ; 4mm x 4mm x 6mm x 3 sec

fMRIVolDims <- dim(fMRIVolume); fMRIVolDims
## [1] 64 64 21 180
time_dim <- fMRIVolDims[4]; time_dim
## [1] 180

# Plot the 4D array of imaging data in a 5x5 grid of images
# The first three dimensions are spatial locations of the voxel (volume element) and the fourth dimension is time for this functional MRI (fMRI) acquisition.
image(fMRIVolume, zlim=range(fMRIVolume)*0.95)

hist(fMRIVolume)

# Plot an orthographic display of the fMRI data using the axial plane
# containing the left-and-right thalamus to approximately center
# the crosshair vertically

orthographic(fMRIVolume, xyz=c(34,29,10), zlim=range(fMRIVolume)*0.9)

stat_fmri_test <- ifelse(fMRIVolume > 15000, fMRIVolume, NA)
hist(stat_fmri_test)

dim(stat_fmri_test)
## [1] 64 64 21 180
overlay(fMRIVolume, fMRIVolume[,,,5], zLim.x=range(fMRIVolume)*0.95)

```

```
# overlay(fMRIVolume, stat_fMRI_test[,,,5], zlim.x=range(fMRIVolume)*0.95)

# To examine the time course of a specific 3D voxel (say the one at x=30, y=30, z=15):
plot(fMRIVolume[30, 30, 10,], type='l', main="Time Series of 3D Voxel \n (x=30, y=30, z=15)", col="blue")

x1 <- c(1:180)
y1 <- loess(fMRIVolume[30, 30, 10,]~x1, family = "gaussian")
lines(x1, smooth(fMRIVolume[30, 30, 10,]), col = "red", lwd = 2)
lines(ksmooth(x1, fMRIVolume[30, 30, 10,], kernel = "normal", bandwidth = 5),
, col = "green", lwd = 3)
```

Chapter 19 provides more details about longitudinal and time-series data analysis.

4.6 Appendix

4.6.1 Hands-on Activity (Health Behavior Risks)

```
# load data CaseStudy09_HealthBehaviorRisks_Data
data_2 <- read.csv("https://umich.instructure.com/files/602090/download?down
load_frd=1", sep=",", header = TRUE)
```

Classify the cases using these variables: "AGE_G" "SEX" "RACEGR3" "IMPEDUC" "IMPMRTL" "EMPLOY1" "INCOMG" "CVDINFR4" "CVDCRHD4" "CVDSTRK3" "DIABETE3" "RFSMOK3" "FRTLTI1" "VEGLT1"

```
data.raw <- data_2[, -c(1, 14, 17)]

# Does the classification match either of these:
# TOTINDA (Leisure time physical activities per month, 1=Yes, 2=No,
9=Don't know/Refused/Missing)
# RFDRHV4 (Heavy alcohol consumption, 1=No, 2=Yes,
9=Don't know/Refused/Missing)

hc = hclust(dist(data.raw), 'ave')
# the agglomeration method can be specified "ward.D", "ward.D2", "single",
"complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC)
or "centroid" (= UPGMC)
```

Plot a clustering diagram (Fig. 4.45)

```
par (mfrow=c(1, 1))
# very simple dendrogram
plot(hc)
```

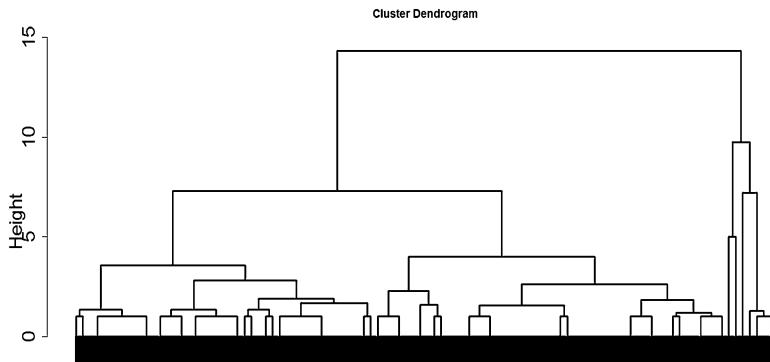


Fig. 4.45 Clustering dendrogram using the Health Behavior Risks case-study

Let's try to identify the number of cases for varying number of clusters.

```
# To identify the number of cases for varying number of clusters we
# can combine calls to cutree and table in a call to sapply -
# to see the sizes of the clusters for $2\leq k \leq 10$ cluster-solutions:
# numbClusters=4;
myClusters = sapply(2:5, function(numbClusters)table(cutree(hc,
numbClusters)))
names(myClusters) <- paste("Number of Clusters=", 2:5, sep = "")
myClusters
```

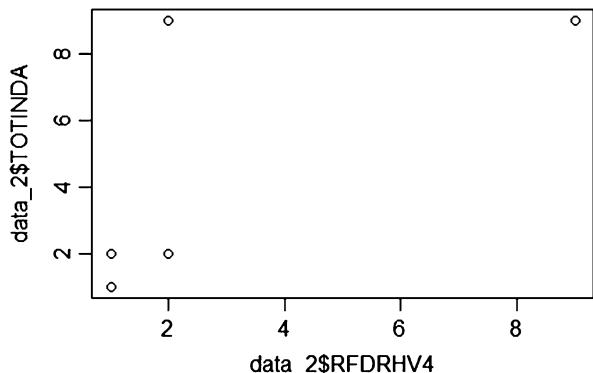
```
## $`Number of Clusters=2`  
##  
## 1 2  
## 930 70  
##  
## $`Number of Clusters=3`  
##  
## 1 2 3  
## 930 50 20  
##  
## $`Number of Clusters=4`  
##  
## 1 2 3 4  
## 500 430 50 20  
##  
## $`Number of Clusters=5`  
##  
## 1 2 3 4 5  
## 500 430 10 40 20
```

Next, inspect the cluster labels for all **SubjectIDs**:

```
#To see which SubjectIDs are in which clusters:  
table(cutree(hc, k=2))  
  
##  
## 1 2  
## 930 70  
  
groups.k.2 <- cutree(hc, k = 2)  
sapply(unique(groups.k.2), function(g) data_2$ID[groups.k.2 == g])
```

We can examine which TOTINDA (Leisure time physical activities per month, 1 = Yes, 2 = No, 9 = Don't know/Refused/Missing) and which RFDRHV4 are in which clusters (Fig. 4.46):

Fig. 4.46 Scatterplot between two variables in the Health Behavior Risks case-study




```
# drill down deeper
table(groups.k.3, data_2$RFDRHV4)

## 
## groups.k.3   1    2    9
##           1 910  20    0
##           2   0  40  10
##           3   0    0  20
```

To characterize the clusters, we can look at cluster summary statistics, like the median, of the variables that were used to perform the cluster analysis. These can be broken down by the groups identified by the cluster analysis. The aggregate function will compute statistics (e.g., median) on many variables simultaneously. Let's examine the median values for each variable we used in the cluster analysis, broken up by cluster groups:

```

aggregate(data_2, list(groups.k.3), median)
##   Group.1     ID AGE_G SEX RACEGR3 IMPEDUC IMPMRTL EMPLOY1 INCOMG CVDINFR4
## 1      1 465.5    5   2      1      5      1      2      4      2
## 2      2 955.5    6   2      4      6      5      8      6      2
## 3      3 990.5    6   2      9      6      6      8      6      2
##   CVDCRHD4 CVDSTRK3 DIABETE3 RFMSMOK3 RFDRHV4 FRTLT1 VEGLT1 TOTINDA
## 1      2.0      2      3      1      1      1      1      1
## 2      2.0      2      3      2      2      9      9      2
## 3      4.5      2      4      9      9      9      9      9

```

4.6.2 Additional *ggplot* Examples

Below, we will show additional visualization examples.

Housing Price Data

This example uses the SOCR Home Price Index data of 19 major US cities from 1991 to 2009 (Fig. 4.47).

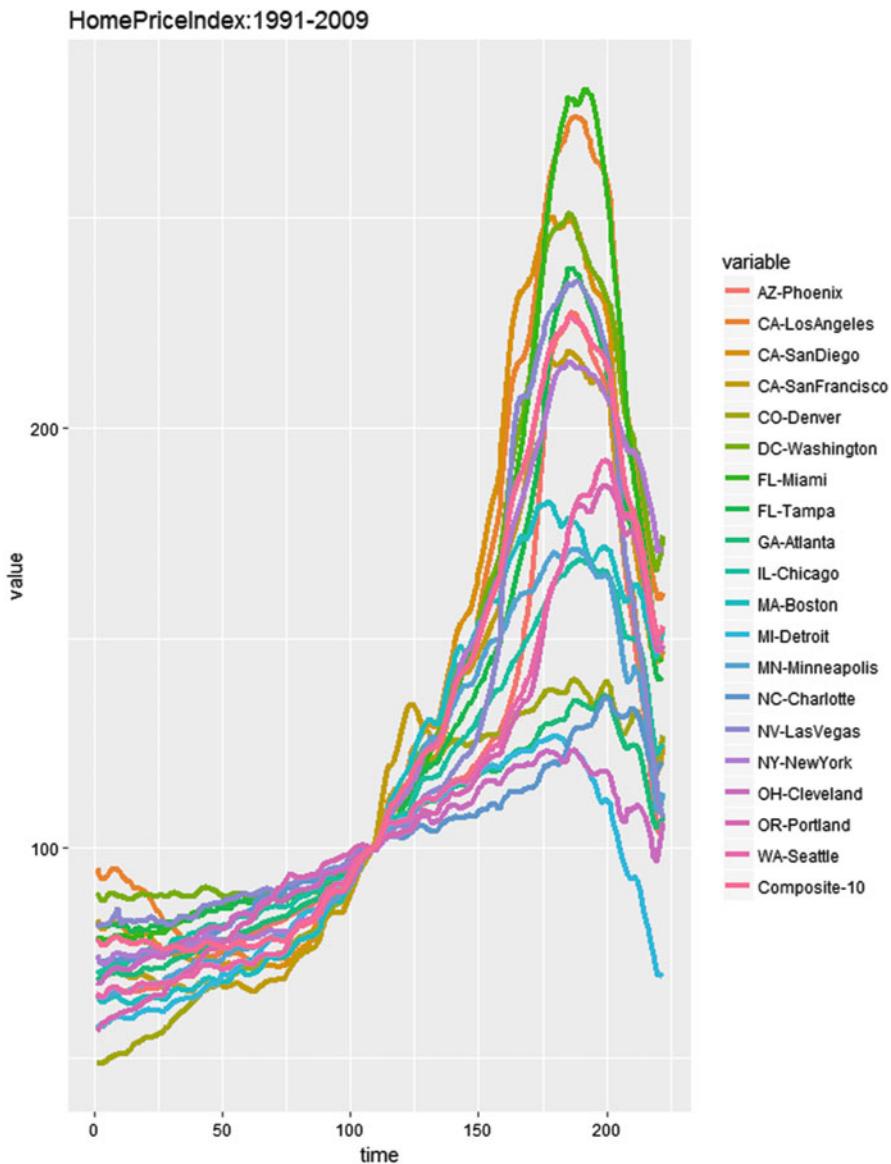


Fig. 4.47 Home price index plot over time

```

library(rvest)
# draw data
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_Dinov_091609_SnP_HomePriceIndex")
hm_price_index<- html_table(html_nodes(wiki_url, "table"))[[1]]
head(hm_price_index)

##   Index Year Month AZ-Phoenix CA-LosAngeles CA-SanDiego CA-SanFrancisco
## 1 1 1991 January 65.26 95.28 83.13 71.17
## 2 2 1991 February 65.29 94.12 81.87 70.27
## 3 3 1991 March 64.60 92.83 80.89 69.56
## 4 4 1991 April 64.35 92.83 80.73 69.46
## 5 5 1991 May 64.37 93.37 81.41 70.13
## 6 6 1991 June 64.88 94.25 82.20 70.83
##   CO-Denver DC-Washington FL-Miami FL-Tampa GA-Atlanta IL-Chicago
## 1 48.67 89.38 79.08 81.75 69.61 70.04
## 2 48.68 88.80 78.55 81.76 69.17 70.50
## 3 48.85 87.59 78.44 81.43 69.05 70.63
## 4 49.20 87.56 78.55 81.46 69.40 71.09
## 5 49.51 88.61 77.95 81.33 69.69 71.36
## 6 50.09 89.28 78.49 81.77 70.14 71.66
##   MA-Boston MI-Detroit MN-Minneapolis NC-Charlotte NV-LasVegas NY-NewYork
## 1 64.97 58.24 64.21 73.32 80.96 74.59
## 2 64.17 57.76 64.20 73.26 81.58 73.69
## 3 63.57 57.63 64.19 72.75 81.65 72.87
## 4 63.35 57.85 64.30 72.88 81.67 72.29
## 5 63.84 58.36 64.75 73.26 82.02 72.63
## 6 64.25 58.90 64.95 73.49 81.91 73.50
##   OH-Cleveland OR-Portland WA-Seattle Composite-10
## 1 68.24 56.53 65.53 78.53
## 2 67.96 56.94 64.60 77.77
## 3 68.18 58.03 64.47 77.00
## 4 69.10 58.39 65.09 76.86
## 5 69.92 58.90 66.03 77.31
## 6 70.55 59.54 66.68 78.02
hm_price_index <- hm_price_index[, c(-2, -3)]
colnames(hm_price_index)[1] <- c('time')
require(reshape)

hm_index_melted = melt(hm_price_index, id.vars='time') #a common trick for plot, wide -> long format
ggplot(data=hm_index_melted, aes(x=time, y=value, color=variable)) +
  geom_line(size=1.5) + ggtitle("HomePriceIndex:1991-2009")

```

Modeling the Home Price Index Data (Fig. 4.48)

```

# Linear regression and predict
hm_price_index$pred = predict(lm(`CA-SanFrancisco` ~ `CA-LosAngeles`,
data=hm_price_index))
ggplot(data=hm_price_index, aes(x = `CA-LosAngeles`)) +
  geom_point(aes(y = `CA-SanFrancisco`)) +
  geom_line(aes(y = pred), color='Magenta', size=2) +
  ggtitle("PredictHomeIndex SF - LA")

```

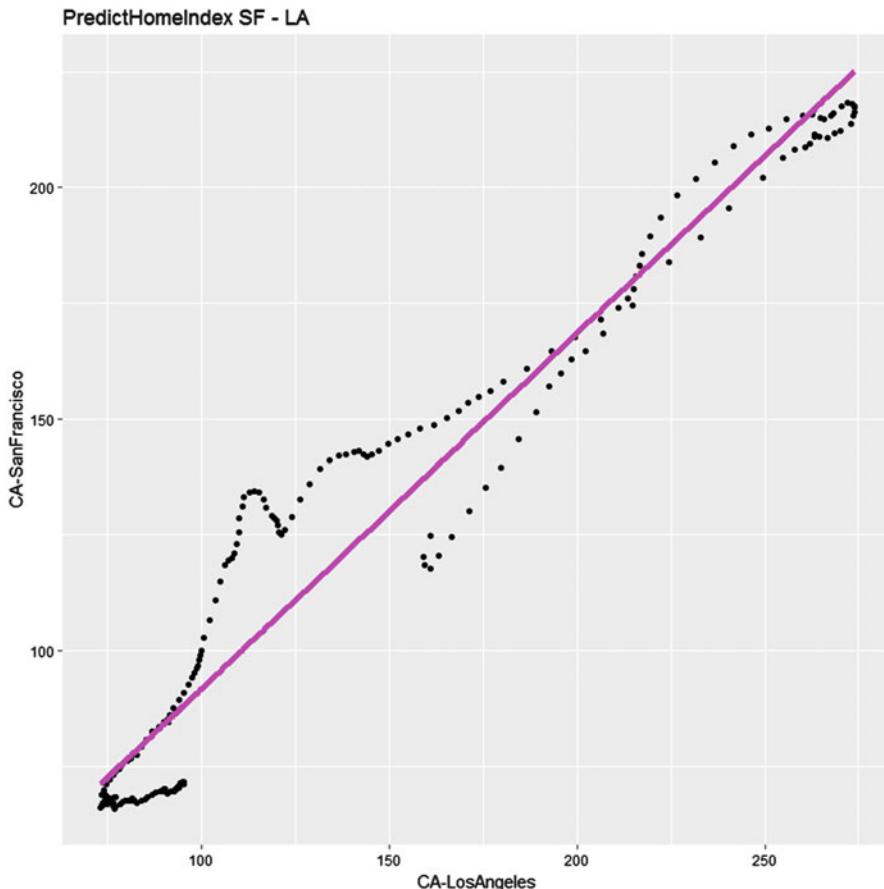


Fig. 4.48 Predicting the San Francisco home process using data from the Los Angeles home sales

We can also use `ggplot` to draw pairs plots (Fig. 4.49).

```
# install.packages("GGally")
require(GGally)

pairs <- hm_price_index[, 10:15]
head(pairs)

##   GA-Atlanta IL-Chicago MA-Boston MI-Detroit MN-Minneapolis NC-Charlotte
## 1   69.61     70.04    64.97     58.24      64.21     73.32
## 2   69.17     70.50    64.17     57.76      64.20     73.26
## 3   69.05     70.63    63.57     57.63      64.19     72.75
## 4   69.40     71.09    63.35     57.85      64.30     72.88
## 5   69.69     71.36    63.84     58.36      64.75     73.26
## 6   70.14     71.66    64.25     58.90      64.95     73.49

colnames(pairs) <- c("Atlanta", "Chicago", "Boston", "Detroit",
"Minneapolis", "Charlotte")
ggpairs(pairs) # you can define the plot design by claim "upper", "lower",
"diag" etc.
```

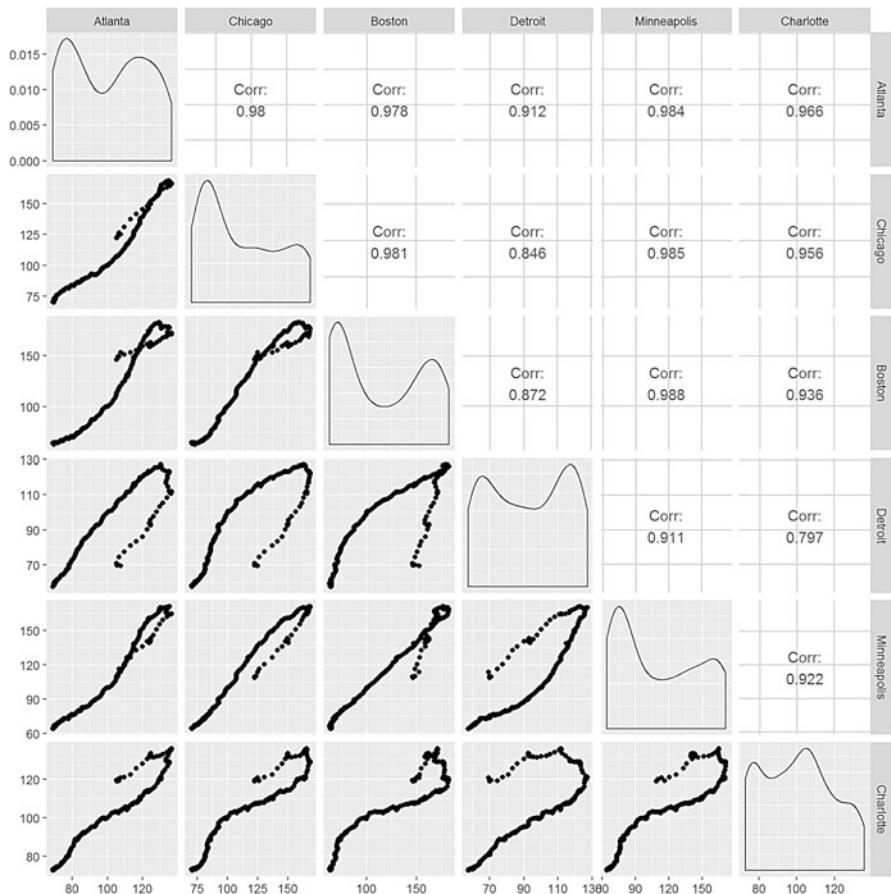


Fig. 4.49 A more elaborate pairs plot of the home price index dataset illustrating the distributions of home prices within a metropolitan area, as well as the paired relations between regions

Map of the Neighborhoods of Los Angeles (LA)

This example interrogates data of 110 LA neighborhoods, which includes measures of education, income, and population demographics.

Here, we select the **Longitude** and **Latitude** as the axes, mark these 110 Neighborhoods according to their population, fill out those points according to the income of each area, and label each neighborhood (Fig. 4.50).

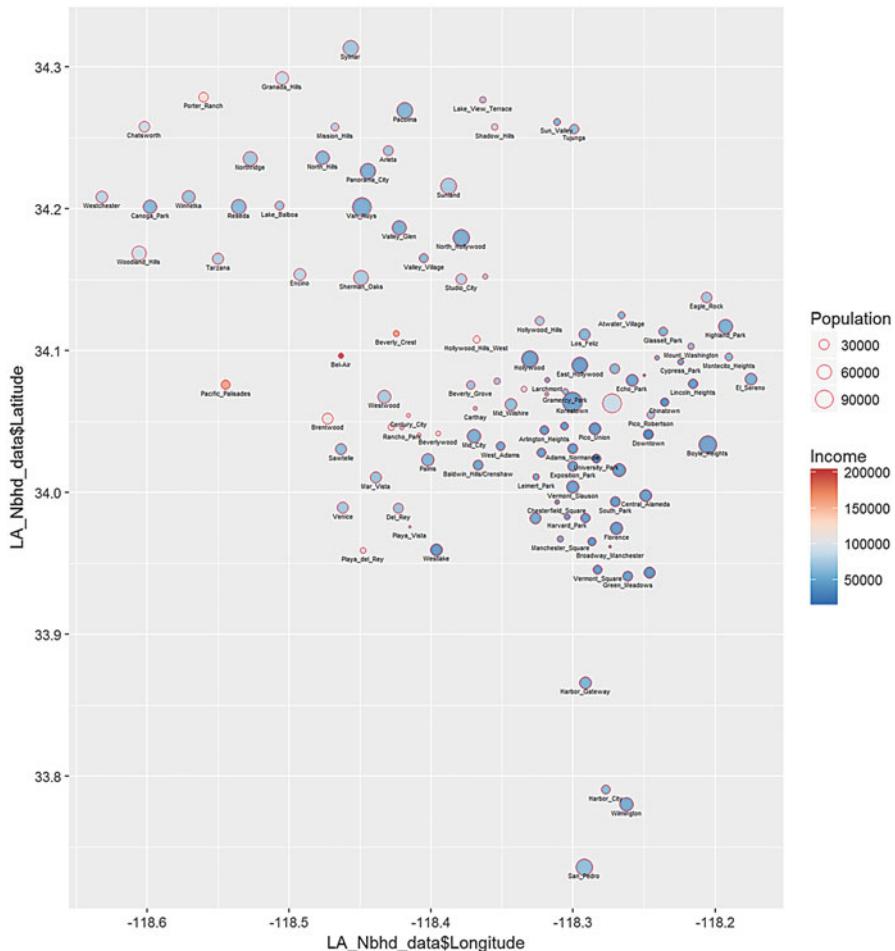


Fig. 4.50 Bubble plot of Los Angeles neighborhood location (longitude vs latitude), population size, and income

```
library(rvest)
require(ggplot2)
#draw data
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_LA_Neighborhoods_Data")
html_nodes(wiki_url, "#content")
## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...</div>
```

```

LA_Nbhd_data <- html_table(html_nodes(wiki_url, "table")[[2]])
#display several lines of data
head(LA_Nbhd_data);

##           LA_Nbhd Income Schools Diversity Age Homes Vets Asian
## 1      Adams_Normandie 29606     691      0.6  26  0.26 0.05  0.05
## 2             Arleta 65649     719      0.4  29  0.29 0.07  0.11
## 3      Arlington_Heights 31423     687      0.8  31  0.31 0.05  0.13
## 4      Atwater_Village 53872     762      0.9  34  0.34 0.06  0.20
## 5 Baldwin_Hills/Crenshaw 37948     656      0.4  36  0.36 0.10  0.05
## 6            Bel-Air 208861     924      0.2  46  0.46 0.13  0.08
##   Black Latino White Population Area Longitude Latitude
## 1  0.25  0.62  0.06    31068  0.8 -118.3003 34.03097
## 2  0.02  0.72  0.13    31068  3.1 -118.4300 34.24060
## 3  0.25  0.57  0.05   22106  1.0 -118.3201 34.04361
## 4  0.01  0.51  0.22   14888  1.8 -118.2658 34.12491
## 5  0.71  0.17  0.03   30123  3.0 -118.3667 34.01909
## 6  0.01  0.05  0.83    7928  6.6 -118.4636 34.09615

theme_set(theme_grey())
#treat ggplot as a variable
##When claim "data", we can access its column directly eg "x = Longitude"
plot1 = ggplot(data=LA_Nbhd_data, aes(x=LA_Nbhd_data$Longitude,
y=LA_Nbhd_data$Latitude))
#you can easily add attribute, points, label(eg:text)
plot1 + geom_point(aes(size=Population, fill=LA_Nbhd_data$Income), pch=21,
stroke=0.2, alpha=0.7, color=2) +
  geom_text(aes(Label=LA_Nbhd_data$LA_Nbhd), size=1.5, hjust=0.5, vjust=2,
check_overlap = T) +
  scale_size_area() + scale_fill_distiller(limits=c(range(LA_Nbhd_data$Income)),
palette='RdBu', na.value='white', name='Income') +
  scale_y_continuous(limits=c(min(LA_Nbhd_data$Latitude), max(LA_Nbhd_data$Latitude))) +
  coord_fixed(ratio=1) + ggtitle('LA Neighborhoods Scatter Plot (Location,
Population, Income)')

```

Observe that some areas (e.g., Beverly Hills) have disproportionately higher incomes. In addition, it is worth pointing out that the resulting plot resembles this plot of LA County (Fig. 4.51).

Latin Letter Frequency in Different Languages

This example uses ggplot to interrogate the SOCR Latin letter frequency data, which includes the frequencies of the 26 common Latin characters in several derivative languages. There is quite a variation between the frequencies of Latin letters in different languages (Figs. 4.52 and 4.53).

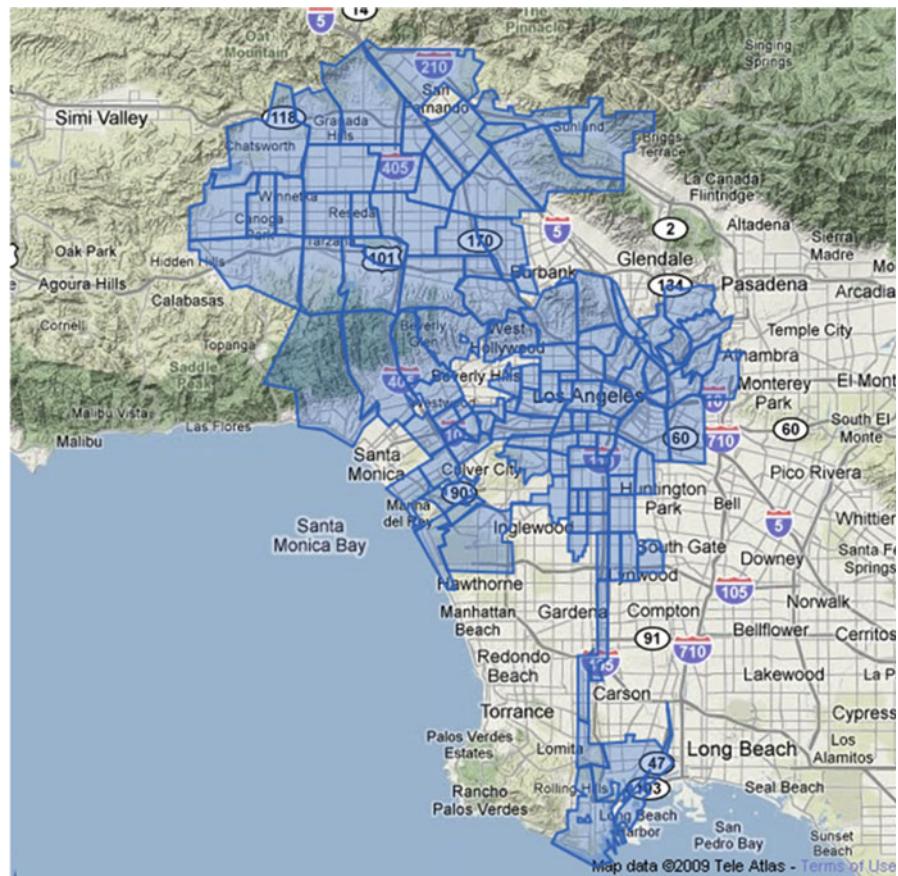


Fig. 4.51 The Los Angeles county map resembles the plot on Fig. 4.50

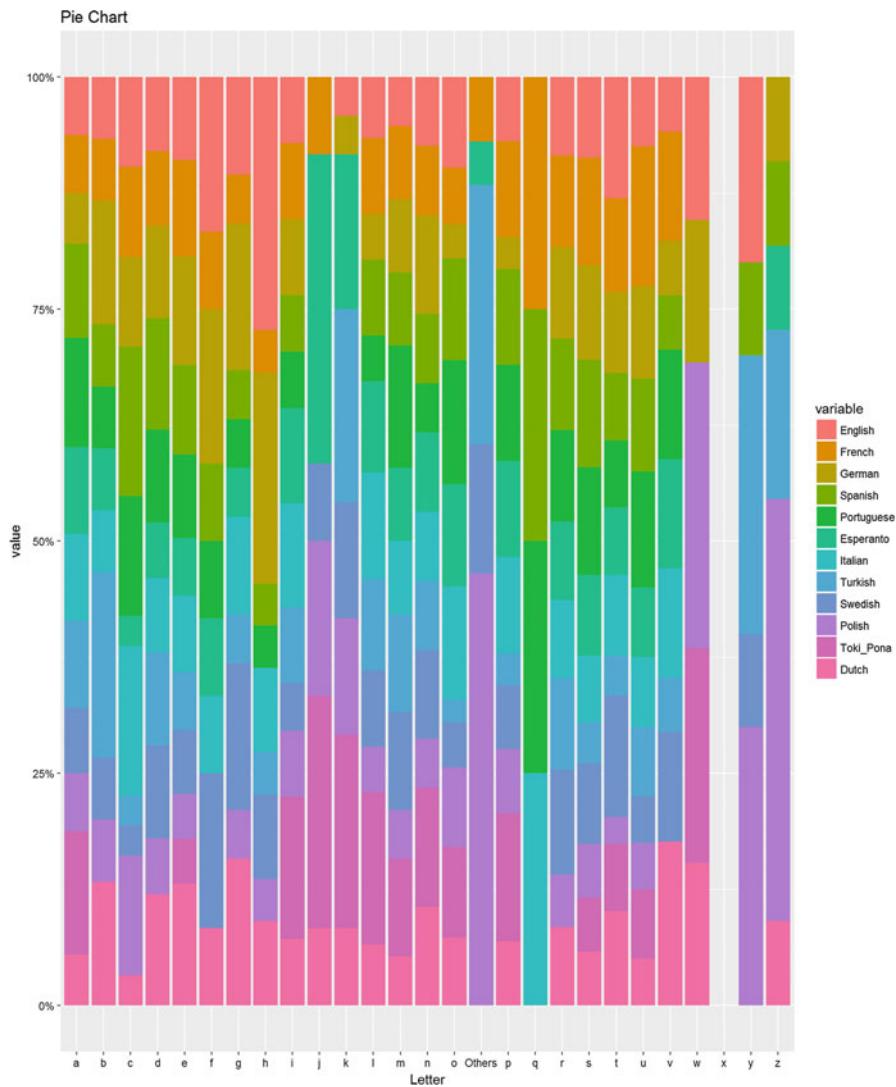


Fig. 4.52 Frequency distributions of Latin letters in several languages

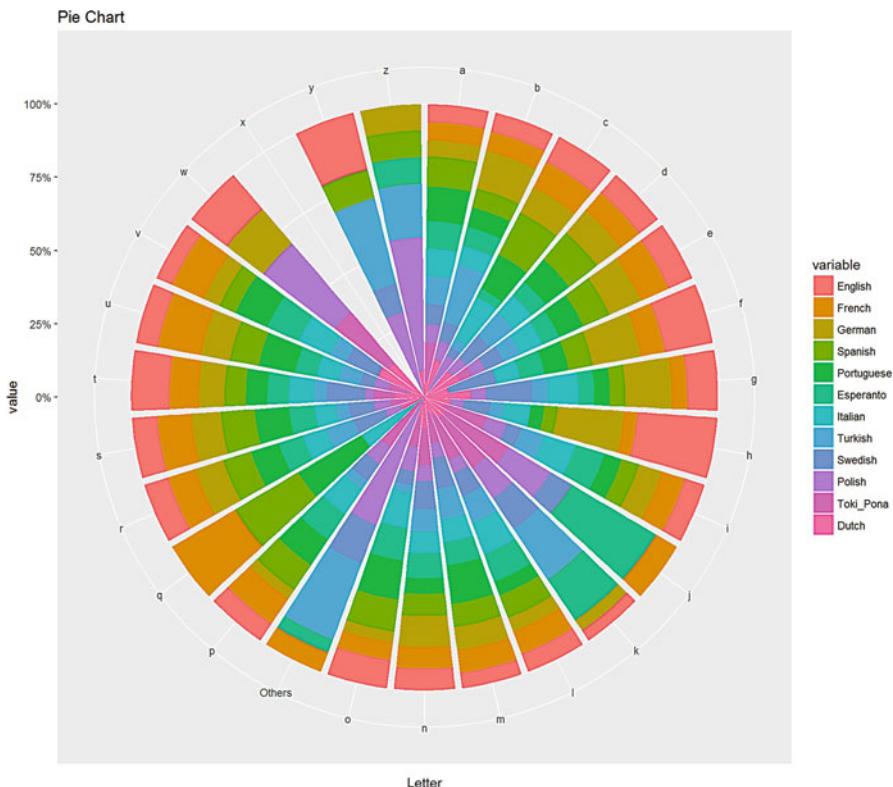


Fig. 4.53 Pie chart similar to the stacked bar chart, Fig. 4.52

```

library(rvest)
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_LetterFrequencyData")
letter<- html_table(html_nodes(wiki_url, "table"))[[1]]
summary(letter)

##      Letter      English      French      German
## Length:27  Min.   :0.00000  Min.   :0.00000  Min.   :0.00000
## Class :character 1st Qu.:0.01000  1st Qu.:0.01000  1st Qu.:0.01000
## Mode  :character Median :0.02000  Median :0.03000  Median :0.03000
##               Mean  :0.03667  Mean  :0.03704  Mean  :0.03741
##               3rd Qu.:0.06000  3rd Qu.:0.06500  3rd Qu.:0.05500
##               Max.  :0.13000  Max.  :0.15000  Max.  :0.17000
##      Spanish      Portuguese      Esperanto      Italian
##  Min.   :0.00000  Min.   :0.00000  Min.   :0.00000  Min.   :0.00000
##  1st Qu.:0.01000  1st Qu.:0.00500  1st Qu.:0.01000  1st Qu.:0.00500
##  Median :0.03000  Median :0.03000  Median :0.03000  Median :0.03000
##  Mean   :0.03815  Mean   :0.03778  Mean   :0.03704  Mean   :0.03815
##  3rd Qu.:0.06000  3rd Qu.:0.05000  3rd Qu.:0.06000  3rd Qu.:0.06000
##  Max.   :0.14000  Max.   :0.15000  Max.   :0.12000  Max.   :0.12000

```

```

##      Turkish      Swedish      Polish      Toki_Pona
## Min. :0.00000  Min. :0.00000  Min. :0.00000  Min. :0.00000
## 1st Qu.:0.01000 1st Qu.:0.01000 1st Qu.:0.01500 1st Qu.:0.00000
## Median :0.03000 Median :0.03000 Median :0.03000 Median :0.03000
## Mean   :0.03667 Mean   :0.03704 Mean   :0.03704 Mean   :0.03704
## 3rd Qu.:0.05500 3rd Qu.:0.05500 3rd Qu.:0.04500 3rd Qu.:0.05000
## Max.   :0.12000 Max.   :0.10000 Max.   :0.20000 Max.   :0.17000
##      Dutch      Avgerage
## Min. :0.00000  Min. :0.00000
## 1st Qu.:0.01000 1st Qu.:0.01000
## Median :0.02000 Median :0.03000
## Mean   :0.03704 Mean   :0.03741
## 3rd Qu.:0.06000 3rd Qu.:0.06000
## Max.   :0.19000 Max.   :0.12000

head(Letter)

##   Letter English French German Spanish Portuguese Esperanto Italian
## 1      a    0.08   0.08   0.07   0.13    0.15    0.12    0.12
## 2      b    0.01   0.01   0.02   0.01    0.01    0.01    0.01
## 3      c    0.03   0.03   0.03   0.05    0.04    0.01    0.05
## 4      d    0.04   0.04   0.05   0.06    0.05    0.03    0.04
## 5      e    0.13   0.15   0.17   0.14    0.13    0.09    0.12
## 6      f    0.02   0.01   0.02   0.01    0.01    0.01    0.01

##      Turkish Swedish Polish Toki_Pona Dutch Avgerage
## 1      0.12    0.09   0.08   0.17   0.07    0.11
## 2      0.03    0.01   0.01   0.00   0.02    0.01
## 3      0.01    0.01   0.04   0.00   0.01    0.03
## 4      0.05    0.05   0.03   0.00   0.06    0.04
## 5      0.09    0.10   0.07   0.07   0.19    0.12
## 6      0.00    0.02   0.00   0.00   0.01    0.01

sum(Letter[, -1]) #reasonable
## [1] 13.08

require(reshape)
library(scales)

dtm = melt(Letter[, -14], id.vars = c('Letter'))
p = ggplot(dtm, aes(x = Letter, y = value, fill = variable)) +
  geom_bar(position = "fill", stat = "identity") +
  scale_y_continuous(labels = percent_format())+ggtitle('Pie Chart')
#or exchange
#p = ggplot(dtm, aes(x = variable, y = value, fill = Letter)) +
#  geom_bar(position = "fill", stat = "identity") +
#  scale_y_continuous(labels = percent_format())
p

#gg pie plot actually is stack plot + polar coordinate
p + coord_polar()

```

You can experiment with the SOCR interactive motion chart, see Fig. 4.54.

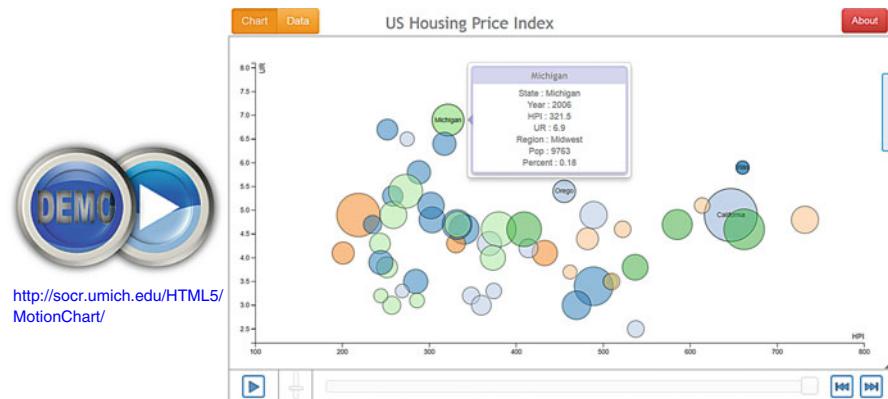


Fig. 4.54 Live demo: 6D SOCR MotionChart

4.7 Assignments 4: Data Visualization

4.7.1 Common Plots

Use the Divorce data (Case Study 01) or the TBI dataset (CaseStudy11_TBI) to generate appropriate visualization of histograms, density plots, pie charts, heatmaps, barplots, and paired correlation plots.

4.7.2 Trees and Graphs

Use the SOCR Resource Hierarchical data (JSON) or the DSPA Dynamic Certificate Map (JSON) to generate some tree/graph displays of the structural information.

The code fragment below shows an example of processing a JSON hierarchy.

```
library(jsonlite)
library(RCurl)
library(data.tree)
url <- "http://socr.umich.edu/html/navigators/D3/xml/SOCR_HyperTree.json"
raw_data <- getURL(url)
document <- fromJSON(raw_data)
tree <- Node$new(document$name)
for(i in seq_len(length(document))) {
  tree$AddChild(document$children$name[[i]])
  for(j in seq_len(length(document$children$children[[i]]))) {
    tree$children[[i]]$AddChild(document$children$children[[i]]$name[[j]])
    for(k in seq_len(length(document$children$children[[i]]$children[[j]]))) {
      tree$children[[i]]$children[[j]]$AddChild((document$children$children[[i]]$children[[j]]$name[[k]]))
    }
  }
}
suppressMessages(library(igraph))
plot(as.igraph(tree, directed = T, direction = "climb"))

suppressMessages(library(networkD3))
treenetwork <- ToDataFrameNetwork(tree, "name")
simpleNetwork(treenetwork, font_size = 10)
```

4.7.3 Exploratory Data Analytics (EDA)

- Use SOCR Oil Gas Data to generate plots: (i) read data table, you may need to fill the inconsistent values with NAs; (ii) data preprocessing: select variables, type convert, etc.; (iii) generate two plots: the first plots includes two subplots, consumption plots and production plots; the second figure includes three subplots, for fossil, nuclear and renewable, respectively. To draw the subplots, you can use `facet_grid()`; (iv) all figures should have `year` as x axis; (v) the first figure should include three curves (fossil, nuclear and renewable) for each subplot; the second figure should include two curves (consumption and production) for each subplot.
- Use SOCR Ozone Data to generate a correlation plot with the variables `MTH_1`, `MTH_2`, ..., `MTH_12`. (Hint: you need to obtain the correlation matrix first, then apply the `corrplot` package. Try some alternative methods as well, `circle`, `pie`, `mixed` etc.)
- Use SOCR CA Ozone Data to generate a 3D surface plot (Using variables *Longitude*, *Latitude* and *O3*).
- Generate a sequence of random numbers from student *t* distribution. Draw the sample histogram and compare it with normal distribution. Try different degrees of freedom. What do you find? Does varying the *seed* and regenerating the student *t* sample change that conclusion?
- Use the SOCR Parkinson's Big Meta data (only rows with `time=0`) to generate a *heatmap plot*. Set `RowSideColors`, `ColSideColors` and `rainbow`. (Hint: you may need to select columns, properly convert the data, and normalize it.)
- Use SOCR 2011 US Jobs Ranking draw scatter plot `Overall_Score` vs. `Average_Income (USD)` include title and label the axes. Then try `qplot` for `Overall_Score` vs. `Average_Income (USD)`: (1) fill with the `Stress_Level`; (2) size the points according to `Hiring_Potential`; and (3) label using `Job_Title`.
- Use SOCR Turkiye Student Evaluation Data to generate trees and graphs, using `cutree()` and select any *k* you prefer. (Use variables Q1–Q28).

References

<http://www.statmethods.net/graphs/>
<http://www.springer.com/us/book/9783319497501>
www.r-graph-gallery.com

Chapter 5

Linear Algebra & Matrix Computing



Linear algebra is a branch of mathematics that studies linear associations using vectors, vector-spaces, linear equations, linear transformations, and matrices. It is generally challenging to visualize complex data, e.g., large vectors, tensors, and tables in n -dimensional Euclidian spaces ($n > 3$). Linear algebra allows us to mathematically represent, computationally model, statistically analyze, synthetically simulate, and visually summarize such complex data.

Virtually all natural processes permit first-order linear approximations. This is useful because linear equations are easy to write, interpret, and solve. These first order approximations may be useful to practically assess the process, determine general trends, identify potential patterns, and suggest associations in the data.

Linear equations represent the simplest type of models for many processes. Higher order models may include additional non-linear terms, e.g., Taylor-series expansions. Linear algebra provides the foundation for linear representation, analytics, computational solutions, inference, and visualization of first-order affine models. Linear algebra is a small part of the larger mathematics field of *functional analysis*, which is actually the infinite-dimensional version of linear algebra.

Specifically, *linear algebra* allows us to **computationally** manipulate, model, solve, and interpret complex systems of equations representing large numbers of dimensions and variables. Arbitrarily large problems can be mathematically transformed into simple matrix equations of the form $Ax = b$ or $Ax = \lambda x$.

In this chapter, we review the fundamentals of linear algebra, matrix manipulation and their applications to represent, model, and analyse real data. Specifically, we will cover (1) construction of matrices and matrix operations, (2) general matrix algebra notations, (3) eigenvalues and eigenvectors of linear operators, (4) least squares estimation, and (5) linear regression and variance-covariance matrices.

5.1 Matrices (Second Order Tensors)

5.1.1 Create Matrices

The easiest way to create a matrix is by using the `matrix()` function, which organizes the elements of a vector into specified positions into a matrix.

```
seq1<-seq(1:6)
m1<-matrix(seq1, nrow=2, ncol=3)
m1
##      [,1] [,2] [,3]
## [1,]     1     3     5
## [2,]     2     4     6

m2<-diag(seq1)
m2
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]     1     0     0     0     0     0
## [2,]     0     2     0     0     0     0
## [3,]     0     0     3     0     0     0
## [4,]     0     0     0     4     0     0
## [5,]     0     0     0     0     5     0
## [6,]     0     0     0     0     0     6

m3<-matrix(rnorm(20), nrow=5)
m3
##      [,1]      [,2]      [,3]      [,4]
## [1,]  0.4877535  0.22081284 -0.6067573 -0.8982306
## [2,] -0.1672924 -1.49020015  0.3038424 -0.1875045
## [3,] -0.4771204 -0.39004837  1.1160825 -0.6948070
## [4,] -0.9274687  0.08378863  0.3846627  0.2386284
## [5,]  0.8672767 -0.86752831  1.5536853  0.3222158
```

The function `diag()` is very useful. When the object is a vector, it creates a diagonal matrix with the vector in the principal diagonal.

```
diag(c(1, 2, 3))
##      [,1] [,2] [,3]
## [1,]     1     0     0
## [2,]     0     2     0
## [3,]     0     0     3
```

When the object is a matrix, `diag()` returns its principal diagonal.

```
diag(m1)
## [1] 1 4
```

When the object is a scalar, `diag(k)` returns a $k \times k$ identity matrix.

```
diag(4)
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

5.1.2 Adding Columns and Rows

Function `cbind()` and `rbind()` are used throughout the textbook.

```
c1<-1:5
m4<-cbind(m3, c1)
m4

##                                     c1
## [1,]  0.4877535  0.22081284 -0.6067573 -0.8982306  1
## [2,] -0.1672924 -1.49020015  0.3038424 -0.1875045  2
## [3,] -0.4771204 -0.39004837  1.1160825 -0.6948070  3
## [4,] -0.9274687  0.08378863  0.3846627  0.2386284  4
## [5,]  0.8672767 -0.86752831  1.5536853  0.3222158  5

r1<-1:4
m5<-rbind(m3, r1)
m5

##      [,1]      [,2]      [,3]      [,4]
##  0.4877535  0.22081284 -0.6067573 -0.8982306
## -0.1672924 -1.49020015  0.3038424 -0.1875045
## -0.4771204 -0.39004837  1.1160825 -0.6948070
## -0.9274687  0.08378863  0.3846627  0.2386284
##  0.8672767 -0.86752831  1.5536853  0.3222158
## r1  1.0000000  2.0000000  3.0000000  4.0000000
```

Note that `m5` has a row name `r1` in the fourth row. We can remove row/column names by naming them as `NULL`.

```
dimnames(m5)<-list(NULL, NULL)
m5

##      [,1]      [,2]      [,3]      [,4]
## [1,]  0.4877535  0.22081284 -0.6067573 -0.8982306
## [2,] -0.1672924 -1.49020015  0.3038424 -0.1875045
## [3,] -0.4771204 -0.39004837  1.1160825 -0.6948070
## [4,] -0.9274687  0.08378863  0.3846627  0.2386284
## [5,]  0.8672767 -0.86752831  1.5536853  0.3222158
## [6,]  1.0000000  2.0000000  3.0000000  4.0000000
```

5.2 Matrix Subscripts

Each element in a matrix has a location. $A[i, j]$ means the i th row and j th column in a matrix A . We can also access specific rows or columns using matrix subscripts.

```
m6<-matrix(1:12, nrow=3)
m6
##      [,1] [,2] [,3] [,4]
## [1,]     1     4     7    10
## [2,]     2     5     8    11
## [3,]     3     6     9    12

m6[1, 2]
## [1] 4
m6[1, ]
## [1] 1 4 7 10
m6[, 2]
## [1] 4 5 6
m6[, c(2, 3)]
##      [,1] [,2]
## [1,]     4     7
## [2,]     5     8
## [3,]     6     9
```

5.3 Matrix Operations

5.3.1 Addition

Elements in the same position are added to represent the result at the same location.

```
m7<-matrix(1:6, nrow=2)
m7
##      [,1] [,2] [,3]
## [1,]     1     3     5
## [2,]     2     4     6

m8<-matrix(2:7, nrow = 2)
m8
##      [,1] [,2] [,3]
## [1,]     2     4     6
## [2,]     3     5     7

m7+m8
##      [,1] [,2] [,3]
## [1,]     3     7    11
## [2,]     5     9    13
```

5.3.2 *Subtraction*

Similar to addition, matrix subtraction reflects differences between elements in same position.

```
m8-m7
##      [,1] [,2] [,3]
## [1,]     1     1     1
## [2,]     1     1     1
m8-1
##      [,1] [,2] [,3]
## [1,]     1     3     5
## [2,]     2     4     6
```

5.3.3 *Multiplication*

Multiplicative operations are different than additive operations. We can do elementwise multiplication or matrix multiplication. For matrix multiplication, the matrix dimensions have to match. That is, the number of columns in the first matrix must equal to the number of rows in the second matrix.

Elementwise Multiplication

Multiplication between elements in same position.

```
m8*m7
##      [,1] [,2] [,3]
## [1,]     2    12    30
## [2,]     6    20    42
```

Matrix Multiplication

The resulting matrix will have the same number of rows as the first matrix and the same number of columns as the second matrix.

```

dim(m8)
## [1] 2 3

m9<-matrix(3:8, nrow=3)
m9

##      [,1] [,2]
## [1,]     3     6
## [2,]     4     7
## [3,]     5     8

dim(m9)
## [1] 3 2

m8%*%m9

##      [,1] [,2]
## [1,]   52   88
## [2,]   64  109

```

We made a 2×2 matrix resulting from multiplying two matrices $2 \times 3 * 3 \times 2$.

The process of multiplying two vectors is called **outer product**. Assume we have two vectors u and v , in matrix multiplication their outer product is represented mathematically as uv^T . In R, the operator for outer product is `%o%`.

```

u<-c(1, 2, 3, 4, 5)
v<-c(4, 5, 6, 7, 8)
u%o%v

##      [,1] [,2] [,3] [,4] [,5]
## [1,]     4     5     6     7     8
## [2,]     8    10    12    14    16
## [3,]    12    15    18    21    24
## [4,]    16    20    24    28    32
## [5,]    20    25    30    35    40

u%*%t(v)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]     4     5     6     7     8
## [2,]     8    10    12    14    16
## [3,]    12    15    18    21    24
## [4,]    16    20    24    28    32
## [5,]    20    25    30    35    40

```

What are the differences between $u \% * \% t(v)$, $u \% * \% t(v)$, $u * t(v)$, and $u * v$?

5.3.4 Element-wise Division

Elementwise division is defined similarly to element-wise multiplication, however, this is different than multiplicative inversion.

```
m8/m7
##      [,1]     [,2]     [,3]
## [1,] 2.0 1.333333 1.200000
## [2,] 1.5 1.250000 1.166667

m8/2
##      [,1] [,2] [,3]
## [1,] 1.0  2.0  3.0
## [2,] 1.5  2.5  3.5
```

5.3.5 Transpose

The transpose of a matrix is a new matrix created by swapping the columns and the rows of the original matrix. Do this in a simple function `t()`.

```
m8
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    3    5    7

t(m8)
##      [,1] [,2]
## [1,]    2    3
## [2,]    4    5
## [3,]    6    7
```

Notice that the $[1, 2]$ element in `m8` is the $[2, 1]$ element in the transpose matrix `t(m8)`.

5.3.6 Multiplicative Inverse

The inverse of a matrix (A^{-1}) is its multiplicative inverse. That is, multiplying the original matrix (A) by its inverse (A^{-1}) yields an identity matrix that has 1's on the diagonal and 0's off the diagonal.

$$AA^{-1} = I$$

Assume we have the following 2×2 matrix:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Its matrix inverse is

$$\frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

For higher dimensions, the formula for computing the inverse matrix is more complex. In R, we can use the `solve()` function to calculate the matrix inverse, if it exists.

```
m10<-matrix(1:4, nrow=2)
m10

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

solve(m10)

##      [,1] [,2]
## [1,]   -2   1.5
## [2,]    1  -0.5

m10%*%solve(m10)

##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Note that only some matrices have inverses. These are square matrices, i.e., they have the same number of rows and columns, and are non-singular.

Another function that can help us compute the inverse of a matrix is the `ginv()` function under the MASS package, which reports the Moore-Penrose Generalized Inverse of a matrix.

```
require(MASS)

## Loading required package: MASS

ginv(m10)

##      [,1] [,2]
## [1,]   -2   1.5
## [2,]    1  -0.5
```

Also, the same function `solve()` can be used to solve matrix equations. `solve(A, b)` returns vector x in the equation $b = Ax$ (i.e., $x = A^{-1}b$).

```
s1<-diag(c(2, 4, 6, 8))
s2<-c(1, 2, 3, 4)
solve(s1, s2)

## [1] 0.5 0.5 0.5 0.5
```

The following Table 5.1 summarizes some basic matrix operation functions.

Table 5.1 Basic matrix operators in R

Expression	Explanation
<code>t(x)</code>	Transpose
<code>diag(x)</code>	Diagonal
<code>%^%</code>	Matrix multiplication
<code>solve(a, b)</code>	Solves a <code>%^% x = b</code> for x
<code>solve(a)</code>	Matrix inverse of a
<code>rowsum(x)</code>	Sum of rows for a matrix-like object. <code>rowSums(x)</code> is a faster version
<code>colSums(x), colSums(x)</code>	Id. for columns
<code>rowMeans(x)</code>	Fast version of row means
<code>colMeans(x)</code>	Id. for columns

```

mat1 <- cbind(c(1, -1/5), c(-1/3, 1))
mat1.inv <- solve(mat1)

mat1.identity <- mat1.inv %*% mat1
mat1.identity

##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1

b <- c(1, 2)
x <- solve (mat1, b)
x

## [1] 1.785714 2.357143

```

5.4 Matrix Algebra Notation

Let's introduce the basic matrix notation. The product AB between matrices A and B is defined only if the number of columns in A equals the number of rows in B . That is, we can multiply an $m \times n$ matrix A by an $n \times k$ matrix B and the result will be $AB_{m \times k}$ matrix. Each element of the product matrix, $(AB_{i,j})$, represents the product of the i th row in A and the j th column in B , which are of the same size n . Matrix multiplication is `row-by-column`.

5.4.1 Linear Models

Linear algebra notation simplifies the mathematical descriptions and manipulations of linear models, as well as coding in R.

The main point is to show how we can write linear models using matrix notation. Later, we'll explain how this is useful for solving the least squares matrix equation. Let's start by defining the notation and matrix multiplication.

5.4.2 Solving Systems of Equations

Linear algebra notation enables the mathematical analysis and the analytical solution of systems of linear equations:

$$\begin{aligned} a + b + 2c &= 6 \\ 3a - 2b + c &= 2. \\ 2a + b - c &= 3 \end{aligned}$$

It provides a generic machinery for solving these problems.

$$\underbrace{\begin{pmatrix} 1 & 1 & 2 \\ 3 & -2 & 1 \\ 2 & 1 & -1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_x = \underbrace{\begin{pmatrix} 6 \\ 2 \\ 3 \end{pmatrix}}_b.$$

That is: $Ax = b$, which yields a solution vector x :

$$x = \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 & 1 & 2 \\ 3 & -2 & 1 \\ 2 & 1 & -1 \end{pmatrix}^{-1} \begin{pmatrix} 6 \\ 2 \\ 3 \end{pmatrix}.$$

In other words, $A^{-1}Ax = x = A^{-1}b$.

Notice that this parallels the solution of simple (univariate) linear equations like:

$$\underbrace{2}_{\text{(design matrix) } A} \underbrace{x}_{\text{unknown } x} - \underbrace{3}_{\text{simple constant term}} = \underbrace{5}_{b}.$$

The constant term, -3 , can be simply joined with the right-hand-size, b , to form a new term $b' = 5 + 3 = 8$. Thus, the shifting factor is mostly ignored in linear models, or linear equations, to simplify the equation to:

$$\underbrace{2}_{\text{(design matrix) } A} \underbrace{x}_{\text{unknown } x} = \underbrace{5 + 3}_{b'} = \underbrace{8}_{b'}.$$

This (simple) linear equation is solved by multiplying both sides by the inverse (reciprocal) of the x multiplier, 2:

$$\frac{1}{2}2x = \frac{1}{2}8.$$

Thus, the unique solution is:

$$x = \frac{1}{2}8 = 4.$$

So, let's use exactly the same protocol to solve the corresponding matrix equation (linear equations, $Ax = b$) using R (the unknown is x , and the design matrix A and the constant vector b are known):

$$\underbrace{\begin{pmatrix} 1 & 1 & 2 \\ 3 & -2 & 1 \\ 2 & 1 & -1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_x = \underbrace{\begin{pmatrix} 6 \\ 2 \\ 3 \end{pmatrix}}_b.$$

```
A_matrix_values <- c(1, 1, 2, 3, -2, 1, 2, 1, -1)
A <- t(matrix(A_matrix_values, nrow=3, ncol=3))
b <- c(6, 2, 3)
# to solve Ax = b, x=A^{-1}*b
x <- solve(A, b)
# Ax = b ==> x = A^{-1} * b
x

## [1] 1.35 1.75 1.45

# Check the Solution x=(1.35 1.75 1.45)
LHS <- A %*% x
round (LHS-b)

##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0
```

How about if we want to triple-check the accuracy of the `solve` method to provide accurate solutions to matrix-based systems of linear equations?

We can generate the solution (x) to the equation $Ax = b$ using first principles:

$$x = A^{-1}b.$$

```
A.inverse <- solve(A) # the inverse matrix A^{-1}
x1 <- A.inverse %*% b
# check if X and x1 are the same
x; x1

## [1] 1.35 1.75 1.45

##      [,1]
## [1,] 1.35
## [2,] 1.75
## [3,] 1.45

round(x-x1,6)

##                  [,1]
## [1,] 0
## [2,] 0
## [3,] 0
```

5.4.3 The Identity Matrix

The identity matrix is the matrix analog to the multiplicative numeric identity, i.e., the number 1. Multiplying the identity matrix by any other matrix (B) does not change the matrix B . For this to happen, the multiplicative identity matrix must look like:

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

The identity matrix is always a square matrix with diagonal elements 1 and 0 at the off-diagonal elements.

If you follow the matrix multiplication rule above, you notice this works out:

$$\mathbf{X} \times \mathbf{I} = \begin{pmatrix} x_{1,1} & \dots & x_{1,p} \\ \vdots & & \\ x_{n,1} & \dots & x_{n,p} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} = \begin{pmatrix} x_{1,1} & \dots & x_{1,p} \\ \vdots & & \\ x_{n,1} & \dots & x_{n,p} \end{pmatrix}.$$

In R, you can form an identity matrix as follows:

```

n <- 3 #pick dimensions
I <- diag(n); I

##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1

A %*% I; I %*% A

##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    1   -2    1
## [3,]    2    1   -1

##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    1   -2    1
## [3,]    2    1   -1

```

5.5 Scalars, Vectors and Matrices

Let's look at this notation deeper. In the baseball player data, there are three quantitative variables: Heights, Weight, and Age. Suppose the variable Weight is represented as a response Y_1, \dots, Y_n random vector.

We can examine players' Weight as a function of Age and Height.

```
# Data: https://umich.instructure.com/courses/38100/files/folder/data
#01a_data.txt)
data <- read.table('https://umich.instructure.com/files/330381/download?downLoad_frd=1', as.is=T, header=T)
attach(data)
head(data)

##           Name Team      Position Height Weight   Age
## 1 Adam_Donachie  BAL     Catcher    74   180 22.99
## 2 Paul_Bako      BAL     Catcher    74   215 34.69
## 3 Ramon_Hernandez  BAL     Catcher    72   210 30.78
## 4 Kevin_Millar    BAL First_Baseman  72   210 35.43
## 5 Chris_Gomez     BAL First_Baseman  73   188 35.71
## 6 Brian_Roberts   BAL Second_Baseman 69   176 29.39
```

We can also use vector notation. We usually use bold to distinguish vectors from the individual elements:

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}.$$

The default representation of data vectors is as columns, i.e., we have dimension $n \times 1$, as opposed to $1 \times n$ rows.

Similarly, we can use math notation to represent the covariates or predictors: Age and Height. In a case with two predictors, we can represent them like this:

$$\mathbf{X}_1 = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} \text{ and } \mathbf{X}_2 = \begin{pmatrix} x_{1,2} \\ \vdots \\ x_{n,2} \end{pmatrix}.$$

Note that for the baseball players example, $x_{1,1} = \text{Age}_1$ and $x_{i,1} = \text{Age}_i$, with Age_i represent the Age of the i th player, and similarly, $x_{i,2} = \text{Height}_i$, represents the height of the i th player. These vectors are also thought of as $n \times 1$ matrices.

It is convenient to represent both covariates as a matrix:

$$\mathbf{X} = [\mathbf{X}_1 \mathbf{X}_2] = \begin{pmatrix} x_{1,1} & x_{1,2} \\ \vdots & \vdots \\ x_{n,1} & x_{n,2} \end{pmatrix}.$$

This matrix is of dimension $n \times 2$ and can be create in R this way:

```
X <- cbind(Age, Height)
head(X)

##           Age Height
## [1,] 22.99    74
## [2,] 34.69    74
## [3,] 30.78    72
## [4,] 35.43    72
## [5,] 35.71    73
## [6,] 29.39    69

dim(X)

## [1] 1034     2
```

We can also use this notation to denote an arbitrary number of covariates (k) with the following $n \times k$ matrix:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & \dots & x_{1,k} \\ x_{2,1} & \dots & x_{2,k} \\ \vdots & & \vdots \\ x_{n,1} & \dots & x_{n,k} \end{pmatrix}.$$

You can simulate such a matrix in R now using `matrix`, instead of `cbind`:

```
n <- 1034; k <- 5
X <- matrix(1:(n*k), n, k)
head(X)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1 1035 2069 3103 4137
## [2,]    2 1036 2070 3104 4138
## [3,]    3 1037 2071 3105 4139
## [4,]    4 1038 2072 3106 4140
## [5,]    5 1039 2073 3107 4141
## [6,]    6 1040 2074 3108 4142

dim(X)

## [1] 1034     5
```

By default, the matrices are filled in a column-by-column order; however using the `byrow=TRUE` argument allows us to change that order to row-by-row:

```
n <- 1034; k <- 5
X <- matrix(1:(n*k), n, k, byrow=TRUE)
head(X)

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
## [5,]   21   22   23   24   25
## [6,]   26   27   28   29   30

dim(X)

## [1] 1034     5
```

A scalar is just a univariate number, which is different from vectors and matrices, that is usually denoted by lower case letters.

5.5.1 Sample Statistics (Mean, Variance)

Mean

To compute the sample average and variance of a dataset, we use the formulas:

$$\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$$

and

$$\text{var}(Y) = \frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{Y})^2,$$

which can be represented with matrix multiplication.

Define a $n \times 1$ matrix made of 1's:

$$A = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

This implies that:

$$\frac{1}{n} \mathbf{A}^\top Y = \frac{1}{n} (1 \ 1 \ \dots \ 1) \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \frac{1}{n} \sum_{i=1}^n Y_i = \bar{Y}.$$

Note that we multiply matrices by scalars, like $\frac{1}{n}$, using the traditional multiplication operator `*`, whereas we multiply two matrices using this operator `%^%`:

```
# Using the Baseball dataset
y <- data$Height
print(mean(y))

## [1] 73.69729

n <- Length(y)
Y<- matrix(y, n, 1)
A <- matrix(1, n, 1)
barY=t(A)%^%Y / n
```

```

print(barY)
##      [,1]
## [1,] 73.69729
# double-check the result
mean(data$Height)
## [1] 73.69729

```

Note: Multiplying the transpose of a matrix with another matrix is very common in statistical modeling and computing. Thus, there is an R function for this operation, `crossprod()`:

```

barY=crossprod(A, Y) / n
print(barY)
##      [,1]
## [1,] 73.69729

```

Variance

For the variance, we note that if:

$$\mathbf{Y}' \equiv \begin{pmatrix} Y_1 - \bar{Y} \\ \vdots \\ Y_n - \bar{Y} \end{pmatrix}, \frac{1}{n-1} \mathbf{Y}'^\top \mathbf{Y}' = \frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{Y})^2.$$

A `crossprod` with only one matrix input computes: $\mathbf{Y}'^\top \mathbf{Y}'$ Thus, to compute the variance, we can simply type:

```

Y1 <- y - barY
crossprod(Y1)/(n-1) # Y1.man <- (1/(n-1))* t(Y1) %% Y1
##      [,1]
## [1,] 5.316798

```

Applications of Matrix Algebra: Linear Modeling

Let's use these matrices:

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \text{ and } \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

Then, we can write a simple linear model:

$$Y_i = \beta_0 + \beta_1 x_i + \epsilon_i, i = 1, \dots, n$$

as:

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

or simply:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

which is a brief way to write the same model equation.

The optimal solution is achieved when all residuals (ϵ_i) are as small as possible (indicating a good model fit). This corresponds to the least squares (LS) solution to this matrix equation ($\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$), which can be obtained by minimizing the residual square error:

$$\langle \boldsymbol{\epsilon}^T, \boldsymbol{\epsilon} \rangle = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^T \times (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

This can be achieved using the following cross-product:

$$\hat{\boldsymbol{\beta}} = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

We can determine the values of $\boldsymbol{\beta}$ by minimizing this expression, using calculus to find the minimum of the cost (objective) function, more about optimization is in Chap. 22.

Finding Function Extrema (Min/Max) Using Calculus

There are a series of rules that permit us to solve partial derivative equations in matrix notation. By setting the derivative of a cost function to zero and solving for the unknown parameter $\boldsymbol{\beta}$, we obtain a candidate solution(s). The derivative of the above equation is:

$$\begin{aligned} 2\mathbf{X}^\top (\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}) &= 0 \\ \mathbf{X}^\top \mathbf{X}\hat{\boldsymbol{\beta}} &= \mathbf{X}^\top \mathbf{Y} \\ \hat{\boldsymbol{\beta}} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}, \end{aligned}$$

which represents the desired solution. Hat notation (^) is used to denote estimates. For instance, the solution for the unknown $\boldsymbol{\beta}$ parameters is denoted by the (data-driven) estimate $\hat{\boldsymbol{\beta}}$.

The least squares minimization works because minimizing a function corresponds to finding the roots of its (first) derivative. In the ordinary least squares (OLS), we square the residuals:

$$(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}).$$

Notice that the minima of $f(x)$ and $f^2(x)$ are achieved at the same roots of $f'(x)$, as the derivative of $f^2(x)$ is $2f(x)f'(x)$.

5.5.2 Least Square Estimation

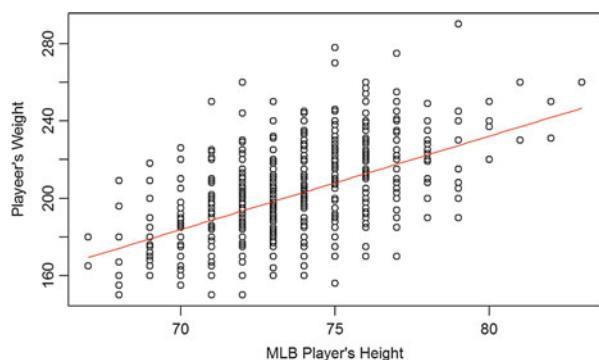
```
#x=cbind(data$Height, data$Age)
x=data$Height
y=data$Weight
X <- cbind(1, x)
beta_hat <- solve( t(X) %*% X ) %*% t(X) %*% y
###or
beta_hat <- solve( crossprod(X) ) %*% crossprod( X, y )
```

Now we can see the results of this by computing the estimated $\hat{\beta}_0 + \hat{\beta}_1 x$ for any value of x (Fig. 5.1):

```
newx <- seq(min(x), max(x), Len=100)
X <- cbind(1, newx)
fitted <- X%*%beta_hat
plot(x, y, xlab="MLB Player's Height", yLab="Player's Weight")
lines(newx, fitted, col=2)
```

$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$ is one of the most widely used results in data analytics. One of the advantages of this approach is that we can use it in many different situations.

Fig. 5.1 A linear model of player's weight as a function of their height overlayed on the paired scatterplot



The R `lm` Function

R has a very convenient function that fits these models. We will learn more about this function later, but here is a preview:

```
# X <- cbind(data$Height, data$Age) # more complicated model
X <- data$Height      # simple model
y <- data$Weight
fit <- lm(y ~ X);
fit
```

Note that we obtain the same estimates of the solution using either the built-in `lm()` function or using first-principles.

5.6 Eigenvalues and Eigenvectors

Eigen-spectrum decomposition of linear operators (matrices) into *eigenvalues* and *eigenvectors* enables us to easily understand linear transformations. The eigenvectors represent the “axes” (directions) along which a linear transformation acts by *stretching*, *compressing*, or *flipping*. The eigenvalues represent the amounts of this linear transformation into the specified eigenvector direction. In higher dimensions, there are more directions along which we need to understand the behavior of the linear transformation. The eigen-spectrum makes it easier to understand the linear transformation, especially when many (perhaps all) of the eigenvectors are linearly independent (orthogonal).

For a matrix A , if we have $A\vec{v} = \lambda\vec{v}$ then we say \vec{v} (a non-zero vector) is a right eigenvector of the matrix A , and the scale factor λ is the eigenvalue corresponding to that eigenvector.

With some calculations we know that $A\vec{v} = \lambda\vec{v}$ is the same as $(\lambda I_n - A)\vec{v} = \vec{0}$. Here I_n is the $n \times n$ identity matrix. So, when we solve this equation, we get our eigenvalues and eigenvectors. Of course, as this is a very common operation, we don’t need to do that by hand – the `eigen()` function in R help us with this calculation.

```
m11<-diag(nrow = 2, ncol=2)
m11

##      [,1] [,2]
## [1,]     1     0
## [2,]     0     1

eigen(m11)

## $values
## [1] 1 1
##
## $vectors
##      [,1] [,2]
## [1,]     0    -1
## [2,]     1     0
```

We can use R to prove that $(\lambda I_n - A)\vec{v} = \vec{0}$.

```
(eigen(m11)$values * diag(2) - m11) %*% eigen(m11)$vectors
##      [,1] [,2]
## [1,]     0     0
## [2,]     0     0
```

As we mentioned earlier, `diag(n)` creates an $n \times n$ identity matrix. Thus, `diag(2)` is the I_2 matrix in the equation. The output zero matrix proves that the equation $(\lambda I_n - A)\vec{v} = \vec{0}$ holds true.

Some of the many interesting applications of the eigen-spectrum are shown below.

5.7 Other Important Functions

Other useful matrix operation are listed in the following Table 5.2.

5.8 Matrix Notation (Another View)

Some flexible matrix operations can help us save time calculating row or column averages. For example, column averages can be calculated by the following matrix operation.

Table 5.2 Other matrix operators and operands

Functions	Math expression or explanation
<code>crossprod(A, B)</code>	$A^T B$ where A, B are matrices
<code>y <- svd(A)</code>	The Singular Value Decomposition output has the following components
<code>-y\$d</code>	Vector containing the singular values of A
<code>-y\$u</code>	Matrix with columns contain the left singular vectors of A
<code>-y\$v</code>	Matrix with columns contain the right singular vectors of A
<code>k <- qr(A)</code>	The output has the following components
<code>-k\$qr</code>	Has an upper triangle that contains the decomposition and a lower triangle that contains information on the Q decomposition.
<code>-k\$rank</code>	Is the rank of A
<code>-k\$qraux</code>	A vector which contains additional information on Q
<code>-k\$pivot</code>	Contains information on the pivoting strategy used.
<code>rowMeans(A) / colMeans(A)</code>	Returns vector of row/column means
<code>rowSums(A) / colSums(A)</code>	Returns vector of row/column sums

$$AX = \begin{pmatrix} \frac{1}{N} & \frac{1}{N} & \dots & \frac{1}{N} \end{pmatrix} \begin{pmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \dots & \dots & \dots \\ X_{N,1} & \dots & X_{N,p} \end{pmatrix} = (\bar{X}_1 \quad \bar{X}_2 \quad \dots \quad \bar{X}_N).$$

While row averages can be calculated by the next operation:

$$XB = \begin{pmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \dots & \dots & \dots \\ X_{N,1} & \dots & X_{N,p} \end{pmatrix} \begin{pmatrix} \frac{1}{p} \\ \frac{1}{p} \\ \frac{1}{p} \\ \frac{1}{p} \end{pmatrix} = \begin{pmatrix} \bar{X}_1 \\ \bar{X}_2 \\ \dots \\ \bar{X}_q \end{pmatrix}.$$

We see that fast calculations can be done by multiplying a matrix in the front or at the back of the original feature matrix. In general, multiplying a vector in front can give us the following equation.

$$AX = (a_1 \quad a_2 \quad \dots \quad a_N) \begin{pmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \dots & \dots & \dots \\ X_{N,1} & \dots & X_{N,p} \end{pmatrix} = \left(\sum_{i=1}^N a_i \bar{X}_{i,1} \quad \sum_{i=1}^N a_i \bar{X}_{i,2} \quad \dots \quad \sum_{i=1}^N a_i \bar{X}_{i,N} \right).$$

Now let's do an example to practice matrix notation. We will use genetic expression data including 8,793 different genes and 208 subjects. These gene expression data represents a microarray experiment—GSE5859—comparing Gene Expression Profiles from Lymphoblastoid cells. Specifically, the data compares the expression level of genes in lymphoblasts from individuals in three HapMap populations {CEU, CHB, JPT}. The study found that more than 1,000 genes were significantly different ($a < 0.05$) in mean expression level between the {CEU} and {CHB + JPT} samples.

The gene expression profiles data has two components:

- The gene expression intensities (exprs_GSE5859.csv) where the rows represent features on the microarray (e.g., genes), and columns represent different microarray samples,
- Meta-data about each of the samples (exprs_MetaData_GSE5859.csv) where rows represent samples, and columns represent meta-data (e.g., sex, age, treatment status, the date the sample processing).

```
gene<-read.csv("https://umich.instructure.com/files/2001417/download?downLo
d_frd=1", header = T) # exprs_GSE5859.csv
info<-read.csv("https://umich.instructure.com/files/2001418/download?downLo
d_frd=1", header=T) # exprs_MetaData_GSE5859.csv
```

Like the `lapply()` function that we will discuss in Chap. 7, the `sapply()` function can be used to calculate column and row averages. Let's compare the output by using `sapply` for first-principles matrix calculations.

```
colmeans<-sapply(gene[, -1], mean)
gene1<-as.matrix(gene[, -1])
# can also use built in functions
# colMeans <- colMeans(gene1)
colmeans.matrix<-crossprod(rep(1/nrow(gene1), nrow(gene1)), gene1)
colmeans[1:15]

##  GSM25581.CEL.gz  GSM25681.CEL.gz  GSM136524.CEL.gz  GSM136707.CEL.gz
##  5.703998      5.721779      5.726300      5.743632
##  GSM25553.CEL.gz  GSM136676.CEL.gz  GSM136711.CEL.gz  GSM136542.CEL.gz
##  5.835499      5.742565      5.751601      5.732211
##  GSM136535.CEL.gz  GSM25399.CEL.gz  GSM25552.CEL.gz  GSM25542.CEL.gz
##  5.741741      5.618825      5.805147      5.733117
##  GSM136544.CEL.gz  GSM25662.CEL.gz  GSM136563.CEL.gz
##  5.733175      5.716855      5.750600

colmeans.matrix[1:15]

## [1] 5.703998 5.721779 5.726300 5.743632 5.835499 5.742565 5.751601
## [8] 5.732211 5.741741 5.618825 5.805147 5.733117 5.733175 5.716855
## [15] 5.750600
```

The same output is reported. Here, we use `rep(1/nrow(gene1), nrow(gene1))` to create the vector

$$\left(\frac{1}{N} \quad \frac{1}{N} \quad \cdots \quad \frac{1}{N} \right)$$

needed to obtain the column averages. We may visualize the column means using a histogram (Fig. 5.2).

```
colmeans<-as.matrix(colmeans)
hist(colmeans)
```

The histogram shows that the distribution is mostly symmetric and bell shaped. We can address harder problems using matrix notation. For example, let's calculate the differences between genders for each gene. First, we need to get the gender information for each subject.

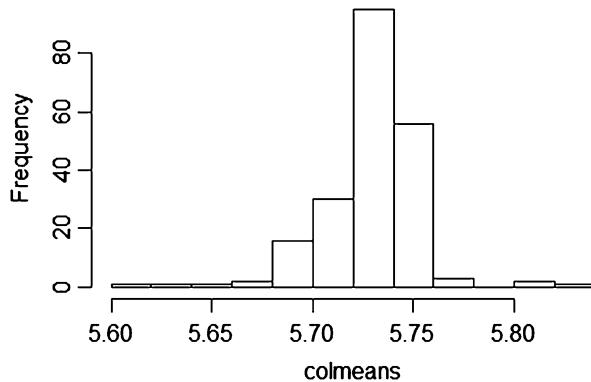
```
gender<-info[, c(3, 4)]
rownames(gender)<-gender$filename
```

Then, we have to reorder the columns to make them consistent with the feature matrix `gene1`.

```
gender<-gender[, colnames(gene1), ]
```

After that, we will construct the design matrix and multiply it by the feature matrix. The plan is to multiply the following two matrices.

Fig. 5.2 Histogram of the column means of the gene expression data



$$\begin{pmatrix} X_{1,1} & \dots & X_{1,p} \\ X_{2,1} & \dots & X_{2,p} \\ \dots & \dots & \dots \\ X_{N,1} & \dots & X_{N,p} \end{pmatrix} \begin{pmatrix} \frac{1}{p} & a_1 \\ \frac{1}{p} & a_2 \\ \frac{1}{p} & \dots \\ \frac{1}{p} & a_p \end{pmatrix} = \begin{pmatrix} \bar{X}_1 & \text{gender.diff}_1 \\ \bar{X}_2 & \text{gender.diff}_2 \\ \dots & \dots \\ \bar{X}_q & \text{gender.diff}_N \end{pmatrix}.$$

where $a_i = -1/N_F$ if the subject is female and $a_i = 1/N_M$ if the subject is male. Thus, we gave each female and male the same weight before the subtraction. We average each gender and get their difference. \bar{X}_i represents the average across both genders and gender.diff_i represents the gender difference for the i th gene.

```
table(gender$sex)

##
##      F      M
##  86 122

gender$vector<-ifelse(gender$sex=="F", -1/86, 1/122)
vec1<-as.matrix(data.frame(rowavg=rep(1/ncol(gene1), ncol(gene1)),
gender.diff=gender$vector))
gender.matrix<-gene1%*%vec1
gender.matrix[1:15, ]

##           rowavg  gender.diff
## [1,] 6.383263 -0.003209464
## [2,] 7.091630 -0.031320597
## [3,] 5.477032  0.064806978
## [4,] 7.584042 -0.001300152
## [5,] 3.197687  0.015265502
## [6,] 7.338204  0.078434938
## [7,] 4.232132  0.008437864
## [8,] 3.716460  0.018235650
## [9,] 2.810554 -0.038698101
## [10,] 5.208787  0.020219666
## [11,] 6.498989  0.025979654
## [12,] 5.292992 -0.029988980
## [13,] 7.069081  0.038575442
## [14,] 5.952406  0.030352616
## [15,] 7.247116  0.046020066
```

5.9 Multivariate Linear Regression

As we mentioned earlier, the formula for multivariate linear regression can be written as

$$Y_i = \beta_0 + X_{i,1}\beta_1 + \cdots + X_{i,p}\beta_p + \epsilon_i, i = 1, \dots, N.$$

We can rewrite this in a matrix form.

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \beta_0 + \begin{pmatrix} X_{1,1} \\ X_{2,1} \\ \vdots \\ X_{N,1} \end{pmatrix} \beta_1 + \cdots + \begin{pmatrix} X_{1,p} \\ X_{2,p} \\ \vdots \\ X_{N,p} \end{pmatrix} \beta_p + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix}.$$

Which is the same as $Y = X\beta + \epsilon$ or

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix} = \begin{pmatrix} 1 & X_{1,1} & \cdots & X_{1,p} \\ 1 & X_{2,1} & \cdots & X_{2,p} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & X_{N,1} & \cdots & X_{N,p} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix}.$$

$Y = X\beta + \epsilon$ implies that $X^T Y \sim X^T (X\beta) = (X^T X)\beta$, and thus the solution for β is obtained by multiplying both hand sides by $(X^T X)^{-1}$:

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

Matrix calculation would be faster than fitting a regression model. Let's apply this to the Lahman baseball data representing yearly stats and standings. Let's download the baseball.data (https://umich.instructure.com/files/2018445/download?download_frd=1) and put it in the R working directory. We can use the `load()` function to load a local RData object. For this example, we subset the dataset by `G==162` and `yearID<2002`. Also, we create a new feature named `Singles` that is equal to `H`(Hits by batters) - `X2B`(Doubles) - `X3B`(Triples) - `HR`(Homeruns by batters). Finally, we only pick some features: `R`(Runs scored), `Singles`, `HR`(Homeruns by batters) and `BB`(Walks by batters).

```
#If you downloaded the .RData locally first, then you can easily load it
into the R workspace by:
# load("Teams.RData")

# Alternatively you can also download the data in CSV format from
# http://umich.instructure.com/courses/38100/files/folder/data (teamsData.csv)
Teams <- read.csv('https://umich.instructure.com/files/2798317/download?download_frd=1', header=T)

dat<-Teams[Teams$G==162&Teams$yearID<2002, ]
dat$Singles<-dat$H-dat$X2B-dat$X3B-dat$HR
dat<-dat[, c("R", "Singles", "HR", "BB")]
head(dat)
```

```

##      R SingLes  HR  BB
## 439  505     997 11 344
## 1367 683     989 90 580
## 1368 744     902 189 681
## 1378 652     948 156 516
## 1380 707    1017  92 620
## 1381 632    1020 126 504

```

Now let's do a simple example. We will use runs scored (R) as the response variable and batters walks (BB) as the independent variable. Also, we need to add a column of 1's to the X matrix.

```

Y<-dat$R
X<-cbind(rep(1, n=nrow(dat)), dat$BB)
X[1:10, ]

```

```

##      [,1] [,2]
## [1,]    1 344
## [2,]    1 580
## [3,]    1 681
## [4,]    1 516
## [5,]    1 620
## [6,]    1 504
## [7,]    1 498
## [8,]    1 502
## [9,]    1 493
## [10,]   1 556

```

Let's solve for the effect-sizes (the beta coefficients) by

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

```

beta<-solve(t(X)%%X)%%t(X)%%Y
beta

```

```

##      [,1]
## [1,] 326.8241628
## [2,]  0.7126402

```

To examine this manual calculation, we refit the linear equation using the `lm()` function. After comparing the time used for computations, we may notice that matrix calculation are more time efficient.

```

fit<-lm(R~BB, data=dat)
# fit<-lm(R~., data=dat)
# '.' indicates all other variables, very useful when fitting models
with many predictors
fit
## 
## Call:
## lm(formula = R ~ BB, data = dat)
## 

```

```

## Coefficients:
## (Intercept)      BB
## 326.8242       0.7126

summary(fit)

## Call:
## lm(formula = R ~ BB, data = dat)
## 

## Residuals:
##    Min     1Q  Median     3Q    Max 
## -187.788 -53.977 -2.995  55.649 258.614 
## 

## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 326.82416  22.44340 14.56   <2e-16 ***
## BB          0.71264   0.04157 17.14   <2e-16 ***
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 

## Residual standard error: 76.95 on 661 degrees of freedom
## Multiple R-squared:  0.3078, Adjusted R-squared:  0.3068 
## F-statistic: 294 on 1 and 661 DF,  p-value: < 2.2e-16

system.time(fit<-lm(R~BB, data=dat))

##    user  system elapsed
##      0      0      0

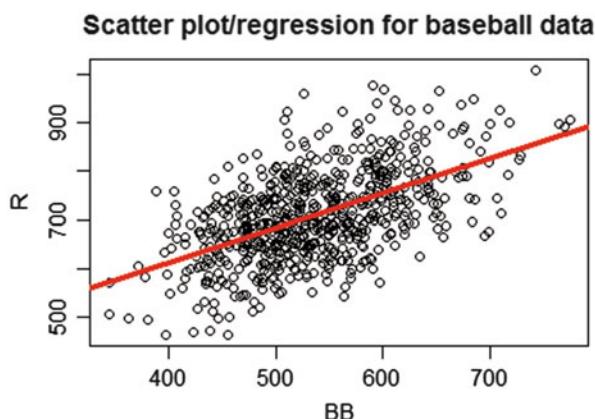
system.time(beta<-solve(t(X)%%X)%%t(X)%%Y)

##    user  system elapsed
##      0      0      0

```

We can visualize the relationship between R and BB by drawing a scatter plot (Fig. 5.3).

Fig. 5.3 Scatterplot and model of walks (BB) and runs (R) by batters, using the MLB dataset



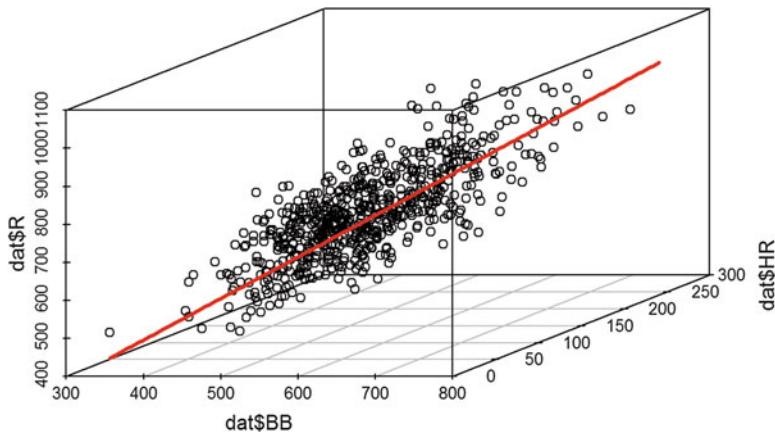


Fig. 5.4 3D Scatterplot of walks (BB), homeruns (HR), and runs (R) by batters, using the baseball dataset

```
plot(dat$BB, dat$R, xlab = "BB", ylab = "R", main = "Scatter plot/regression for baseball data")
abline(beta[1, 1], beta[2, 1], lwd=4, col="red")
```

On Fig. 5.3, the red line is our regression line calculated by matrix calculation. Matrix calculation can still work if we have multiple independent variables. Next, we will add another variable, HR, to the model, Fig. 5.4.

```
X<-cbind(rep(1, n=nrow(dat)), dat$BB, dat$HR)
beta<-solve(t(X)%%X)%*%t(X)%%Y
beta

## [1,] 287.7226756
## [2,] 0.3897178
## [3,] 1.5220448

#install.packages("scatterplot3d")
library(scatterplot3d)
scatterplot3d(dat$BB, dat$HR, dat$R)
```

5.10 Sample Covariance Matrix

We can also obtain the covariance matrix for our features using matrix operations. Suppose

$$X_{N \times K} = \begin{pmatrix} X_{1,1} & \dots & X_{1,K} \\ X_{2,1} & \dots & X_{2,K} \\ \dots & \dots & \dots \\ X_{N,1} & \dots & X_{N,K} \end{pmatrix} = [X_1, X_2, \dots, X_N]^T.$$

Then the covariance matrix is:

$$\Sigma = (\Sigma_{i,j}),$$

where $\Sigma_{i,j} = \text{Cov}(X_i, X_j) = E((X_i - \mu_i)(X_j - \mu_j)), 1 \leq i, j \leq N$.

The sample covariance matrix is:

$$\Sigma_{i,j} = \frac{1}{N-1} \sum_{m=1}^N (x_{m,i} - \bar{x}_i)(x_{m,j} - \bar{x}_j),$$

where

$$\bar{x}_i = \frac{1}{N} \sum_{m=1}^N x_{m,i}, \quad i = 1, \dots, K.$$

In general,

$$\Sigma = \frac{1}{n-1} (X - \bar{X})^T (X - \bar{X}).$$

Assume we want to get the sample covariance matrix of the following 5×3 feature matrix x .

```
x<-matrix(c(4.0, 4.2, 3.9, 4.3, 4.1, 2.0, 2.1, 2.0, 2.1, 2.2, 0.60, 0.59,
0.58, 0.62, 0.63), ncol=3)
x
##      [,1] [,2] [,3]
## [1,] 4.0  2.0  0.60
## [2,] 4.2  2.1  0.59
## [3,] 3.9  2.0  0.58
## [4,] 4.3  2.1  0.62
## [5,] 4.1  2.2  0.63
```

Notice that we have *three* features and *five* observations in this matrix. Let's get the column means first.

```
vec2<-matrix(c(1/5, 1/5, 1/5, 1/5, 1/5), ncol=5)
#column means
x.bar<-vec2%*%x
x.bar
##      [,1] [,2] [,3]
## [1,] 4.1  2.08 0.604
```

Then, we repeat each column mean 5 times to match the layout of feature matrix. Finally, we are able to plug everything in the formula above.

```

x.bar<-matrix(rep(x.bar, each=5), nrow=5)
S<-1/4*t(x-x.bar)%%(x-x.bar)
S

##          [,1]      [,2]      [,3]
## [1,] 0.02500 0.00750 0.00175
## [2,] 0.00750 0.00700 0.00135
## [3,] 0.00175 0.00135 0.00043

```

In the covariance matrix, $S[i, i]$ is the variance of the i th feature and $S[i, j]$ is the covariance of i th and j th features.

Compare this to the automated calculation of the variance-covariance matrix.

```

autoCov <- cov(x)
autoCov

##          [,1]      [,2]      [,3]
## [1,] 0.02500 0.00750 0.00175
## [2,] 0.00750 0.00700 0.00135
## [3,] 0.00175 0.00135 0.00043

```

5.11 Assignments: 5. Linear Algebra & Matrix Computing

5.11.1 How Is Matrix Multiplication Defined?

Validate that $(A_{k,n} \times B_{n,m})^T = (B_{m,n}^T) \times (A_{n,k}^T)$, by using math notation, as well as by using R functions.

5.11.2 Scalar Versus Matrix Multiplication

Demonstrate the differences between the scalar multiplication (*) and matrix multiplication (% * %) for numbers, vectors, and matrices (second-order tensors).

5.11.3 Matrix Equations

Write a simple matrix solver ($b = Ax$, i.e., $x = A^{-1}b$) and validate its accuracy using the R command `solve(A, b)`. Solve this equation:

$$\begin{aligned}
 2a - b + 2c &= 5 \\
 -a - 2b + c &= 3 \\
 a + b - c &= 2
 \end{aligned}$$

5.11.4 Least Square Estimation

Use the SOCR Knee Pain dataset, extract the `RB = Right-Back` locations (x, y) , and fit in a linear model for vertical locations (y) in terms of the horizontal locations (x) . Display the linear model on top of the scatter plot of the paired data. Comment on the model you obtain.

5.11.5 Matrix Manipulation

Create a matrix A with elements `seq(1, 15, length = 6)` and argument `nrow = 3`. Then, add a row to this matrix and add two columns to A to obtain a matrix $C_{4, 4}$. Next, generate a diagonal matrix D with `dim = 4` and elements `rnorm(4, 0, 1)`. Apply elementwise addition, subtraction, multiplication, and division to the matrix C and D ; apply matrix multiplication to D and C ; obtain the inverse of the C and compare it with the generalized inverse, `MASS::ginv()`.

5.11.6 Matrix Transpose

Validate the multiplication transposition formula, $(A_{k,n} \cdot B_{n,m})^T = B_{n,m}^T \cdot A_{k,n}^T$, by using math notation, as well as computationally using R and some example matrices. E.g. you can try

```
A = matrix(1:6,nrow=3); B = matrix(2:7, nrow = 2)
```

5.11.7 Sample Statistics

Use the SOCR Data Iris Sepal Petal Classes and extract the rows of `setosa` flowers. Compute the sample mean and variance of each variables; then calculate sample covariance and correlation between sepal width and sepal height.

5.11.8 Least Square Estimation

Use the SOCR Knee Pain dataset, extract the `RB = Right-Back` locations (x, y) , and fit in a linear model for vertical location (y) in terms of the horizontal

location (x). Display the linear model on top of the scatter plot of the paired data. Comment on the model you obtained.

5.11.9 Eigenvalues and Eigenvectors

Generate a random matrix with $A = \text{matrix}(\text{rnorm}(9, 0, 1), \text{nrow} = 3)$, compute eigenvalues and eigenvectors for A ; then try to solve this equation $\det(A - \lambda I) = 0$, where λ is a vector of length 3. Compare λ and the eigenvalues you solved above.

Example of manual and automated calculations of eigen-spectra (eigenvalues and eigenvectors):

```
# A <- matrix(rnorm(9,0,1),nrow = 3); A
# define a random design matrix, may generate complex solutions
A <- matrix(c(0,1/4,1/4,3/4,0,1/4,1/4,3/4,1/2),3,3,byrow=T); A
eigen_spectrum <- eigen(A); eigen_spectrum

# compute the eigen spectrum (eigen-values, $l$, and eigen-vectors, $v$),
# $ A \times v = l \times v$.
B <- A-eigen(A)$values*diag(3); B

# compute B = (A - eigen_value \times I)
det(A-eigen(A)$values*diag(3))

# verify that the det(A-eigen(A)$values*diag(3)) is not trivial ($0$)
A%*%eigen(A)$vector - eigen(A)$value*diag(3)

# validate that $ A \times v = l \times v$.
all.equal(A, eigen(A)$vector %% diag(eigen(A)$values) %%%
solve(eigen(A)$vector)) # compare A = v*l*inv(v)
all.equal(diag(3), A%*%eigen(A)$vector - eigen(A)$values * eigen(A)$vector)
# The Last Line compares I == AV - Lambda*v, mind the $* and
# %% scalar and matrix operators
```

References

- <http://www.statmethods.net/advstats/matrix.html>
 Vinod, Hrishikesh D. (2011) *Hands-On Matrix Algebra Using R: Active and Motivated Learning with Applications*, World Scientific Publishing, ISBN 981310080X, 9789813100800.
 Gentle, James E. (2007) *Matrix Algebra: Theory, Computations, and Applications in Statistics*, Springer, ISBN 0387708723, 9780387708720.

Chapter 6

Dimensionality Reduction



Now that we have most of the fundamentals covered in the previous chapters, we can delve into the first data analytic method, *dimension reduction*, which reduces the number of features when modeling a very large number of variables. Dimension reduction can help us extract a set of “uncorrelated” principal variables and reduce the complexity of the data. We are not simply picking some of the original variables. Rather, we are constructing new “uncorrelated” variables as functions of the old features.

Dimensionality reduction techniques enable exploratory data analyses by reducing the complexity of the dataset, still approximately preserving important properties, such as retaining the distances between cases or subjects. If we are able to reduce the complexity down to a few dimensions, we can then plot the data and untangle its intrinsic characteristics.

We will (1) start with a synthetic example demonstrating the reduction of a 2D data into 1D; (2) explain the notion of rotation matrices; (3) show examples of principal component analysis (PCA), singular value decomposition (SVD), independent component analysis (ICA) and factor analysis (FA); and (4) present a Parkinson’s disease case-study at the end. The supplementary DSPA electronic materials for this chapter also include the theory and practice of t-Distributed Stochastic Neighbor Embedding (t-SNE), which represents high-dimensional data via projections into non-linear low-dimensional manifolds.

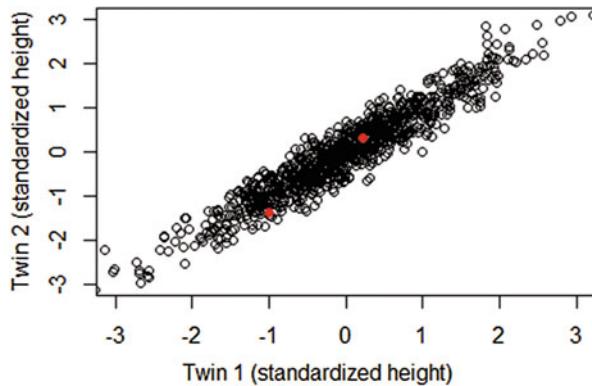
6.1 Example: Reducing 2D to 1D

We consider an example looking at twin heights. Suppose we simulate 1000 2D points that representing normalized individual heights, i.e., number of standard deviations from the mean height. Each 2D point represents a pair of twins. We will simulate this scenario using Bivariate Normal Distribution (Table 6.1 and Fig. 6.1).

Table 6.1 Schematic data structure representation and indexing of twin heights

<i>Twin1_{Height}</i>	<i>Twin2_{Height}</i>
<i>y</i> [1, 1]	<i>y</i> [1, 2]
<i>y</i> [2, 1]	<i>y</i> [2, 2]
<i>y</i> [3, 1]	<i>y</i> [3, 2]
...	...
<i>y</i> [500, 1]	<i>y</i> [500, 2]

Fig. 6.1 Scatterplot of paired twin heights. The red points show the heights of the first two pairs of twins



```
library(MASS)
set.seed(1234)
n <- 1000
y=t(mvrnorm(n, c(0, 0), matrix(c(1, 0.95, 0.95, 1), 2, 2)))
```

$$y_{2 \times 500}^T = \begin{bmatrix} y[1,] = \text{Twin1}_{\text{Height}} \\ y[2,] = \text{Twin2}_{\text{Height}} \end{bmatrix} = \text{BVN} \left(\mu = \begin{bmatrix} \text{Twin1}_{\text{Height}} \\ \text{Twin2}_{\text{Height}} \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.95 \\ 0.95 & 1 \end{bmatrix} \right).$$

```
plot(y[1, ], y[2, ], xlab="Twin 1 (standardized height)",
      ylab="Twin 2 (standardized height)", xlim=c(-3, 3), ylim=c(-3, 3))
points(y[1, 1:2], y[2, 1:2], col=2, pch=16) # plot the first 2 points
```

These data may represent a fraction of the information included in a high-throughput neuroimaging genetics study of twins, like the pediatric study example shown here (http://wiki.socr.umich.edu/index.php/SOCR_Data_Oct2009_ID_NI).

Tracking the distances between any two samples can be accomplished using the `dist` function. For example, here is the distance between the two RED points in the Fig. 6.1:

```
d=dist(t(y))
as.matrix(d)[1, 2]
## [1] 2.100187
```

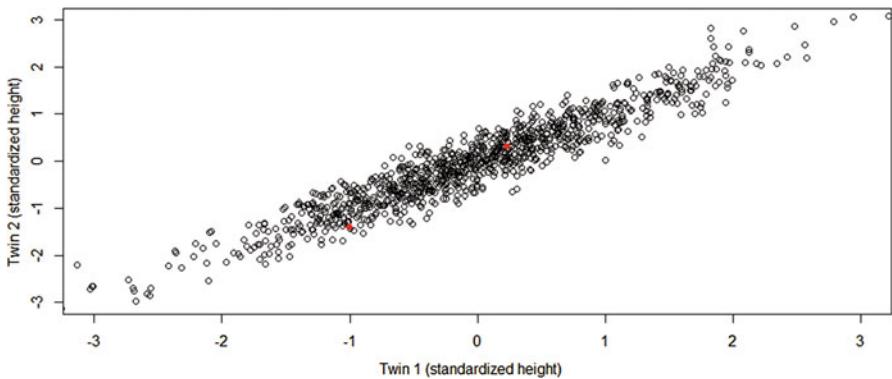


Fig. 6.2 Scatterplots of the raw twin heights

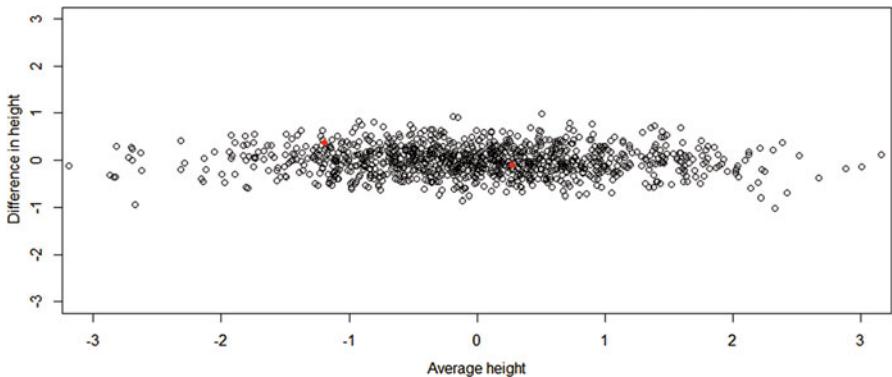


Fig. 6.3 Scatterplots of the transformed twin heights, compare to Fig. 6.2

To transform the 2D data to a simpler 1D plot, we can reduce the data to a 1D matrix (vector) approximately preserving the distances between the 2D points.

The 2D plot shows the Euclidean distance between the pair of red points, Fig. 6.2. The length of this line is the distance between the two points. In 2D, these lines tend to go along the direction of the diagonal. If we rotate the plot so that the diagonal is in the x-axis (Fig. 6.3):

```

z1 = (y[1, ]+y[2, ])/2 # the sum (or rather average)
z2 = (y[1, ]-y[2, ]) # the difference

z = rbind(z1, z2) #matrix now same dimensions as y

thelim <- c(-3, 3)
# par(mar=c(1, 2))
# par(mfrow=c(2,1))
plot(y[1, ], y[2, ], xLab="Twin 1 (standardized height)",
     yLab="Twin 2 (standardized height)",
     xlim=thelim, ylim=thelim)
points(y[1, 1:2], y[2, 1:2], col=2, pch=16)

```

```
par(mfrow=c(1,1))
plot(z[1, ], z[2, ], xlim=thelim, ylim=thelim, xLab="Average height", yLab="Difference in height")
points(z[1, 1:2], z[2, 1:2], col=2, pch=16)
```

Of course, matrix linear algebra notation can be used to represent this affine transformation of the data. Here we can see that to get z we multiplied y by the matrix:

$$A = \begin{pmatrix} 1/2 & 1/2 \\ 1 & -1 \end{pmatrix} \Rightarrow z = A \times y.$$

We can invert this transform by multiplying the result by the inverse matrix A^{-1} as follows:

$$A^{-1} = \begin{pmatrix} 1 & 1/2 \\ 1 & -1/2 \end{pmatrix} \Rightarrow y = A^{-1} \times z.$$

You can try this in R:

```
A <- matrix(c(1/2, 1, 1/2, -1), nrow=2, ncol=2); A  # define a matrix
##      [,1] [,2]
## [1,]  0.5  0.5
## [2,]  1.0 -1.0
A_inv <- solve(A); A_inv  # inverse
##      [,1] [,2]
## [1,]    1  0.5
## [2,]    1 -0.5
A %*% A_inv # Verify result
##      [,1] [,2]
## [1,]    1     0
## [2,]    0     1
```

Note that this matrix transformation *did not* preserve distances, i.e., the matrix A is not a simple rotation in 2D:

```
d=dist(t(y)); as.matrix(d)[1, 2]  # distance between first two points of Y
## [1] 2.100187
d1=dist(t(z)); as.matrix(d1)[1, 2] # distance between first two points of
Z=A*Y
## [1] 1.541323
```

6.2 Matrix Rotations

One important question to ask is how we can identify transformations that preserve distances. In mathematics, transformations between metric spaces that are distance-preserving are called *isometries* (or congruences or congruent transformations).

First, let's test the MA transformation we used above (Fig. 6.3):

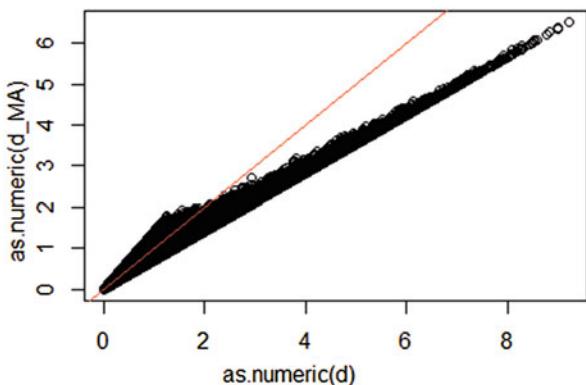
$$\left| \begin{array}{l} M = Y_1 - Y_2 \\ A = \frac{Y_1 + Y_2}{2}. \end{array} \right.$$

```
MA <- matrix(c(1/2, 1, 1/2, -1), 2, 2)
MA_z <- MA%*%y
d <- dist(t(y))
d_MA <- dist(t(MA_z))

plot(as.numeric(d), as.numeric(d_MA))
abline(0, 1, col=2)
```

Observe that this MA transformation is not an isometry – the distances are not preserved. Here is one example with $v1 = \begin{bmatrix} v1_x = 0 \\ v1_y = 1 \end{bmatrix}$, $v2 = \begin{bmatrix} v2_x = 1 \\ v2_y = 0 \end{bmatrix}$, which are distance $\sqrt{2}$ apart in their native space, but separated further by the transformation MA , $d(MA(v1), MA(v2)) = 2$.

Fig. 6.4 The above MA transformation is not an isometry. This scatterplot shows that the relation between the transformed (y-axis) and the native-space (x-axis) twin-pairs distances are not preserved



```

MA; t(MA); solve(MA); t(MA) - solve(MA)

##      [,1] [,2]
## [1,]  0.5  0.5
## [2,]  1.0 -1.0

##      [,1] [,2]
## [1,]  0.5   1
## [2,]  0.5  -1

##      [,1] [,2]
## [1,]    1  0.5
## [2,]    1 -0.5

##      [,1] [,2]
## [1,] -0.5  0.5
## [2,] -0.5 -0.5

v1 <- c(0,1); v2 <- c(1,0); rbind(v1,v2)

##      [,1] [,2]
## v1     0     1
## v2     1     0

euc.dist <- function(x1, x2) sqrt(sum((x1 - x2) ^ 2))
euc.dist(v1,v2)

## [1] 1.414214

v1_t <- MA %*% v1; v2_t <- MA %*% v2
euc.dist(v1_t,v2_t)

## [1] 2

```

More generally, if

$$\begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}\right).$$

Then,

$$Z = AY + \eta \sim BVN(\eta + A\mu, A\Sigma A^T).$$

Where BVN denotes bivariate normal distribution (see <http://socr.umich.edu/HTML5/BivariateNormal/>),

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, Y = (Y_1, Y_2)^T, \mu = (\mu_1, \mu_2), \Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{12} & \sigma_2^2 \end{pmatrix}.$$

You can verify this by using the change of variable theorem. Thus, affine transformations preserve bivariate normality. However, in general, there is no means to guarantee isometry.

The question now is: *Under what additional conditions for a transformation matrix A, can we guarantee an isometry?*

Notice that,

$$d^2(P_i, P_j) = \sum_{k=1}^T (P_{jk} - P_{ik})^2 = \|P\|^2 = P^T P,$$

where $P = (P_{j,1} - P_{i,1}, \dots, P_{j,T} - P_{i,T})^T$, P_i and P_j is any two points in T dimensions.

Thus, the only requirement we need is $(AY)^T(AY) = Y^T Y$, i.e., $A^T A = I$, which implies that A is an orthogonal (rotational) matrix.

Let's use a two dimension orthogonal matrix to illustrate this concept. Set $A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. It's easy to verify that A is an orthogonal (2D rotation) matrix.

The simplest way to test the isometry is to perform the linear transformation directly (Fig. 6.5).

```
A <- 1/sqrt(2)*matrix(c(1, 1, 1, -1), 2, 2)
z <- A%*%y
d <- dist(t(y))
d2 <- dist(t(z))

plot(as.numeric(d), as.numeric(d2))
abline(0, 1, col=2)
```

We can observe that the distances computed using the original data are preserved after the transformation. This transformation is called a rotation (isometry) of y . Note the difference compared to the earlier plot, Fig. 6.4.

An alternative method is to simulate from the joint distribution of $Z = (Z_1, Z_2)^T$. As we have mentioned above:

$$Z = AY + \eta \sim BVN(\eta + A\mu, A\Sigma A^T),$$

where $\eta = (0, 0)^T$, $\Sigma = \begin{pmatrix} 1 & 0.95 \\ 0.95 & 1 \end{pmatrix}$, $A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$.

Fig. 6.5 The matrix A transformation above is distance preserving (i.e., an isometry), as illustrated by the perfect linear relation between the native-space and the transformed pairs of twin height distances

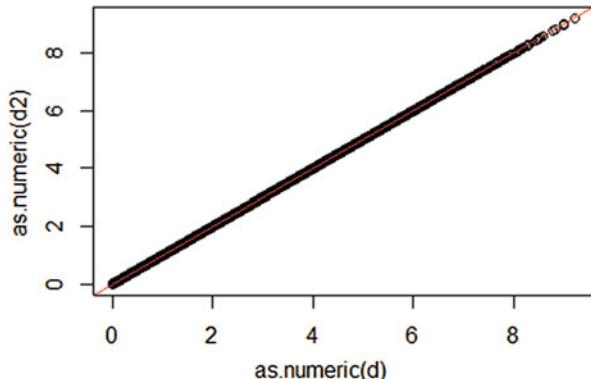
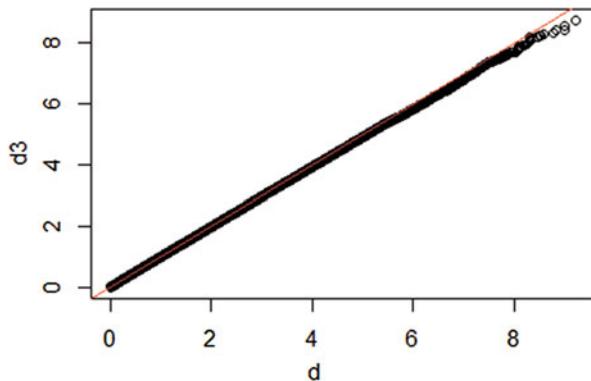


Fig. 6.6 QQ-plot of the distances between twin heights (d) and distances between the simulated bivariate Normal distribution data ($d3$)



We can compute $A\Sigma A^T$ by hand or by using matrix multiplication in R:

```
sig <- matrix(c(1, 0.95, 0.95, 1), nrow=2)
A %*% sig %*% t(A)
##      [,1] [,2]
## [1,] 1.95  0.00
## [2,] 0.00  0.05
```

$A\Sigma A^T$ represents the transformed variance-covariance matrix, $\text{cov}(z_1, z_2) = 0$. We can simulate z_1, z_2 independently from $z_1 \sim N(0, 1.95)$ and $z_2 \sim N(0, 0.05)$ (Note: independence and uncorrelation are equivalent for bivariate normal distribution) (Fig. 6.6).

```
set.seed(2017)
zz1 = rnorm(1000, 0, sd = sqrt(1.95))
zz2 = rnorm(1000, 0, sd = sqrt(0.05))
zz = rbind(zz1, zz2)
d3 = dist(t(zz))
qqplot(d, d3)
abline(a = 0, b=1, col=2)
```

We can observe that the distances computed using the original data and the simulated data are the same (Figs. 6.7 and 6.8).

```
thelim <- c(-3, 3)
#par(mfrow=c(2,1))
plot(y[1, ], y[2, ], xlab="Twin 1 (standardized height)",
      ylab="Twin 2 (standardized height)",
      xlim=thelim, ylim=thelim)
points(y[1, 1:2], y[2, 1:2], col=2, pch=16)
```

```
plot(z[1, ], z[2, ], xlim=thelim, ylim=thelim, xlab="Average height",
      ylab="Difference in height")
points(z[1, 1:2], z[2, 1:2], col=2, pch=16)
```

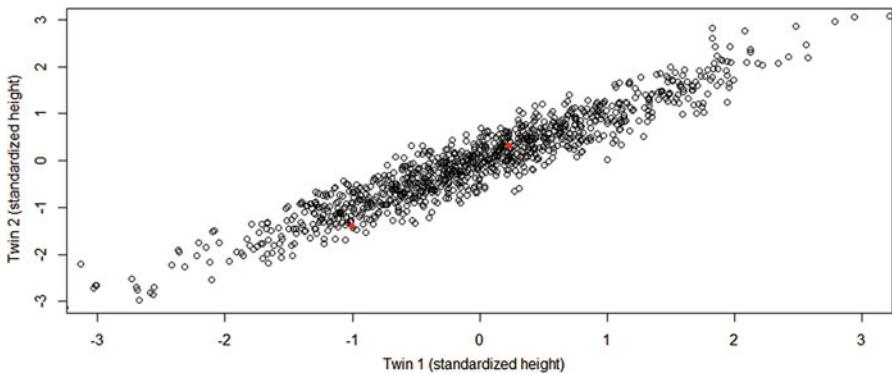


Fig. 6.7 Twin height scatterplot before rotation

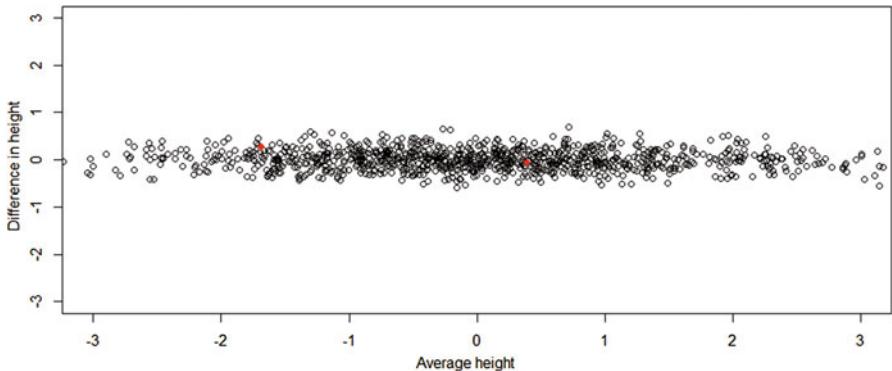


Fig. 6.8 Twin height scatterplot after the rotation

We applied this transformation and observed that the distances between points were unchanged after the rotation. This rotation achieves the goals of:

- Preserving the distances between points, and
- Reducing the dimensionality of the data (see plot reducing 2D to 1D).

Removing the second dimension and recomputing the distances, we get (Fig. 6.9):

```
d4 = dist(z[, 1]) ##distance computed using just the first dimension
plot(as.numeric(d), as.numeric(d4))
abline(0, 1)
```

The 1D distances provide a very good approximation to the actual 2D distances. This first dimension of the transformed data is called the first principal component. In general, this idea motivates the use of principal component analysis (PCA) and the singular value decomposition (SVD) to achieve dimensionality reduction.

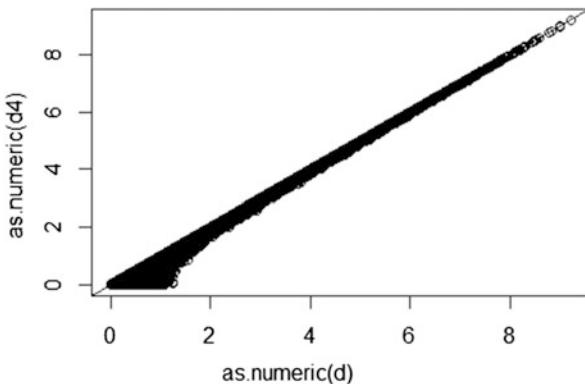


Fig. 6.9 Comparing the twin distances, computed using just one dimension, following the rotation transformation against the actual twin pair height distances. The strong linear relation suggests that measuring distances in the native space is equivalent to measuring distances in the transformed space, where we reduced the dimension of the data from 2D to 1D

6.3 Notation

In the notation above, the rows represent variables and columns represent cases. In general, rows represent cases and columns represent variables. Hence, in our example shown here, Y would be transposed to be a $N \times 2$ matrix. This is the most common way to represent the data: individuals in the rows, features in the columns. In genomics, it is more common to represent subjects/SNPs/genes in the columns. For example, genes are rows and samples are columns. The sample covariance matrix usually denoted with $\mathbf{X}^T \mathbf{X}$ and has cells representing covariance between two units. Yet, for this to be the case, we need the rows of \mathbf{X} to represent the subjects and the columns to represent the variables, or features. Here, we have to compute, $\mathbf{Y} \mathbf{Y}^T$ instead following the rescaling.

6.4 Summary (PCA vs. ICA vs. FA)

Principle Component Analysis (PCA), Independent Component Analysis (ICA), and Factor Analysis (FA) are similar strategies, seeking to identify a new basis (vectors representing the principal directions) that the data is projected against to maximize certain (specific to each technique) objective functions. These basis functions, or vectors, are just linear combinations of the original features in the data/signal.

The singular value decomposition (SVD), discussed later in this chapter, provides a specific matrix factorization algorithm that can be employed in various techniques to decompose a data matrix $X_{m \times n}$ as $U \Sigma V^T$, where U is an $m \times m$ real or complex unitary matrix ($U U^T = U^T U = I$), Σ is a $m \times n$ rectangular diagonal matrix of *singular values*, representing non-negative values on the diagonal, and V is an $n \times n$ unitary matrix (Table 6.2).

Table 6.2 Summary of some dimensionality reduction methods

Method	Assumptions	Cost function optimization	Applications
<i>PCA</i>	Gaussian signals	Aims to explain the variance in the original signal. Minimizes the covariance of the data and yields high-energy orthogonal vectors in terms of the signal variance. PCA looks for an orthogonal linear transformation that maximizes the variance of the variables	Relies on first and second moments of the measured data, which makes it useful when data features are close to Gaussian
<i>ICA</i>	No Gaussian signal assumptions	Minimizes higher-order statistics (e.g., third and fourth order skewness and kurtosis), effectively minimizing the <i>mutual information</i> of the transformed output. ICA seeks a linear transformation where the basis vectors are statistically independent, but neither Gaussian, orthogonal or ranked in order	Applicable for non-Gaussian, very noisy, or mixture processes composed of simultaneous input from multiple sources
<i>FA</i>	Approximately Gaussian data	Objective function relies on second order moments to compute likelihoods. FA <i>factors</i> are linear combinations that maximize the shared portion of the variance underlying <i>latent variables</i> , which may use a variety of optimization strategies (e.g., maximum likelihood)	PCA-generalization used to test a theoretical model of latent factors causing the observed features

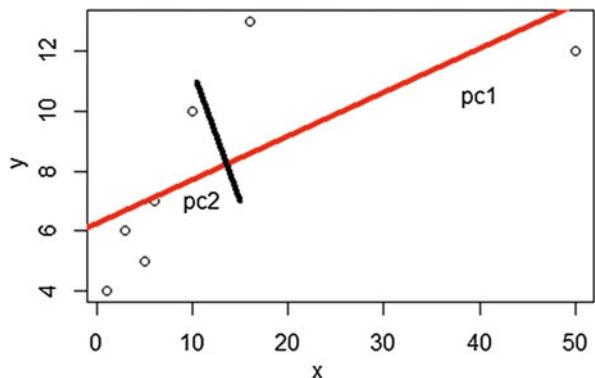
6.5 Principal Component Analysis (PCA)

PCA (principal component analysis) is a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables through a process known as orthogonal transformation.

6.5.1 Principal Components

Let's consider the simplest situation where we have n observations $\{p_1, p_2, \dots, p_n\}$ with two features $p_i = (x_i, y_i)$. When we draw them on a plot, we use the x -axis and y -axis for positioning. However, we can make our own coordinate system by principal components (Fig. 6.10).

Fig. 6.10 Schematic representation of the first two principal components (simulated data)



```
ex<-data.frame(x=c(1, 3, 5, 6, 10, 16, 50), y=c(4, 6, 5, 7, 10, 13, 12))
reg1<-lm(y~x, data=ex)
plot(ex)
abline(reg1, col='red', lwd=4)
text(40, 10.5, "pc1")
segments(10.5, 11, 15, 7, lwd=4)
text(11, 7, "pc2")
```

Illustrated on the graph, the first PC, pc_1 is a minimum distance fit in the feature space. The second PC is a minimum distance fit to a line perpendicular to the first PC. Similarly, the third PC would be a minimum distance fit to all previous PCs. In our case of a 2D space, two PC's is the most we can have. In higher dimensional spaces, we have to figure out how many PCs are needed to make the best performance.

In general, the formula for the first PC is $pc_1 = a_1^T X = \sum_{i=1}^N a_{i,1} X_i$ where X_i is a $n \times 1$ vector representing a column of the matrix X (complete design matrix with a total of n observations and N features). The weights $a_1 = \{a_{1,1}, a_{2,1}, \dots, a_{N,1}\}$ are chosen to maximize the variance of pc_1 . According to this rule, the k th PC $pc_k = a_k^T X = \sum_{i=1}^N a_{i,k} X_i$, where $a_k = \{a_{1,k}, a_{2,k}, \dots, a_{N,k}\}$ has to be constrained by more conditions:

1. Variance of pc_k is maximized
2. $Cov(pc_k, pc_l) = 0, \forall 1 \leq l < k$
3. $a_k^T a_k = 1$ (the weights vectors are unitary).

Let's figure out how to find a_1 . To begin, we need to express the variance of our first principal component using the variance covariance matrix of X :

$$\begin{aligned} Var(pc_1) &= E(pc_1^2) - (E(pc_1))^2 = \\ &\sum_{i,j=1}^N a_{i,1} a_{j,1} E(x_i x_j) - \sum_{i,j=1}^N a_{i,1} a_{j,1} E(x_i) E(x_j) = \\ &\sum_{i,j=1}^N a_{i,1} a_{j,1} S_{i,j}, \end{aligned}$$

where $S_{i,j} = E(x_i x_j) - E(x_i) E(x_j)$.

This implies $\text{Var}(pc_1) = a_1^T S a_1$, where $S = S_{i,j}$ is the covariance matrix of $X = \{X_1, \dots, X_N\}$. Since a_1 maximized $\text{Var}(pc_1)$ and the constraint $a_1^T a_1 = 1$ holds, we can rewrite a_1 as:

$$a_1 = \arg \max_{a_1} (a_1^T S a_1 - \lambda(a_1^T a_1 - 1)).$$

Where the part after the subtraction should be trivial. Take the derivative of this expression w.r.t. a_1 and set the derivative to zero, which yields $(S - \lambda I_N)a_1 = 0$.

In Chap. 5 we showed that a_1 will correspond to the largest eigenvalue of S , the variance covariance matrix of X . Hence, pc_1 retains the largest amount of variation in the sample. Likewise, a_k is the k th largest eigenvalue of S .

PCA requires data matrix to have zero empirical means for each column. That is, the sample mean of each column has been shifted to zero.

Let's use a subset ($N = 33$) of Parkinson's Progression Markers Initiative (PPMI) database to demonstrate the relationship between S and PC loadings. First, we need to import the dataset into R and delete the patient ID column.

```
library(rvest)
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SMHS_PCA_ICA_FA")
html_nodes(wiki_url, "#content")
pd.sub <- html_table(html_nodes(wiki_url, "table"))[[1]]
summary(pd.sub)

##      Patient_ID      Top_of_SN_Voxel_Intensity_Ratio
##  Min.   :3001      Min.   :1.058
##  1st Qu.:3012      1st Qu.:1.334
##  Median :3029      Median :1.485
##  Mean   :3204      Mean   :1.532
##  3rd Qu.:3314      3rd Qu.:1.755
##  Max.   :3808      Max.   :2.149
##      Side_of_SN_Voxel_Intensity_Ratio      Part_IA      Part_IB
##  Min.   :0.9306      Min.   :0.000      Min.   : 0.000
##  1st Qu.:0.9958      1st Qu.:0.000      1st Qu.: 2.000
##  Median :1.1110      Median :1.000      Median : 5.000
##  Mean   :1.1065      Mean   :1.242      Mean   : 4.909
##  3rd Qu.:1.1978      3rd Qu.:2.000      3rd Qu.: 7.000
##  Max.   :1.3811      Max.   :6.000      Max.   :13.000
##      Part_II      Part_III
##  Min.   : 0.000      Min.   : 0.00
##  1st Qu.: 0.000      1st Qu.: 2.00
##  Median : 2.000      Median :12.00
##  Mean   : 4.091      Mean   :13.39
##  3rd Qu.: 6.000      3rd Qu.:20.00
##  Max.   :17.000      Max.   :36.00

pd.sub<-pd.sub[, -1]
```

Then, we need to center the `pdsub` by subtracting the average of all column means from each element. Next we change `pd.sub` to a matrix and get its variance covariance matrix, S . Now, we are able to calculate the eigenvalues and eigenvectors of S .

```

mu<-apply(pd.sub, 2, mean)
mean(mu)

## [1] 4.379068

pd.center<-as.matrix(pd.sub)-mean(mu)
S<-cov(pd.center)
eigen(S)

## $values
## [1] 1.315073e+02 1.178340e+01 6.096920e+00 1.424351e+00 6.094592e-02
## [6] 8.035403e-03
##
## $vectors
## [,1]           [,2]           [,3]           [,4]           [,5]
## [1,] -0.007460885 -0.0182022093  0.016893318  0.02071859  0.97198980
## [2,] -0.005800877  0.0006155246  0.004186177  0.01552971  0.23234862
## [3,]  0.080839361 -0.0600389904 -0.027351225  0.99421646 -0.02352324
## [4,]  0.229718933 -0.2817718053 -0.929463536 -0.06088782  0.01466136
## [5,]  0.282109618 -0.8926329596  0.344508308 -0.06772403 -0.01764367
## [6,]  0.927911126  0.3462292153  0.127908417 -0.05068855  0.01305167
## [,6]
## [1,] -0.232667561
## [2,]  0.972482080
## [3,] -0.009618592
## [4,]  0.003019008
## [5,]  0.006061772
## [6,]  0.002456374

```

The next step is to calculate the PCs using the `prcomp()` function in R. Note that we will use the uncentered version of the data and use `center=T` option. We stored the model information into `pca1`. Then `pca1$rotation` provides the loadings for each PC.

```

pca1<-prcomp(as.matrix(pd.sub), center = T)
summary(pca1)

## Importance of components:
##                               PC1        PC2        PC3        PC4        PC5        PC6
## Standard deviation    11.4677  3.4327  2.46919  1.19346  0.2469  0.08964
## Proportion of Variance 0.8716  0.0781  0.04041  0.00944  0.0004  0.00005
## Cumulative Proportion  0.8716  0.9497  0.99010  0.99954  1.0000  1.00000

pca1$rotation

##                               PC1        PC2        PC3
## Top_of_SN_Voxel_Intensity_Ratio 0.007460885 -0.0182022093  0.016893318
## Side_of_SN_Voxel_Intensity_Ratio 0.005800877  0.0006155246  0.004186177
## Part_IA                         -0.080839361 -0.0600389904 -0.027351225
## Part_IB                         -0.229718933 -0.2817718053 -0.929463536

```

```

## Part_II          -0.282109618 -0.8926329596  0.344508308
## Part_III         -0.927911126  0.3462292153  0.127908417
##                           PC4      PC5      PC6
## Top_of_SN_Voxel_Intensity_Ratio  0.02071859 -0.97198980 -0.232667561
## Side_of_SN_Voxel_Intensity_Ratio  0.01552971 -0.23234862  0.972482080
## Part_IA          0.99421646  0.02352324 -0.009618592
## Part_IB          -0.06088782 -0.01466136  0.003019008
## Part_II          -0.06772403  0.01764367  0.006061772
## Part_III         -0.05068855 -0.01305167  0.002456374

```

The loadings are just the eigenvectors times -1. This actually represents the same line in 6D dimensional space (we have six columns for the original data). The multiplier -1 represents the opposite direction in the same line. For further comparisons, we can load the `factoextra` package to get the eigenvalues of PCs.

```

# install.packages("factoextra")
library("factoextra")

eigen<-get_eigenvalue(pca1); eigen

##           eigenvalue variance.percent cumulative.variance.percent
## Dim.1 1.315073e+02      87.159638589          87.15964
## Dim.2 1.178340e+01       7.809737384          94.96938
## Dim.3 6.096920e+00       4.040881920          99.01026
## Dim.4 1.424351e+00       0.944023059          99.95428
## Dim.5 6.094592e-02       0.040393390          99.99467
## Dim.6 8.035403e-03       0.005325659          100.00000

```

The eigenvalues correspond to the amount of the variation explained by each principal component (PC), which represents the eigenvalues for the S matrix.

To see detailed information about the variances that each PC explains, we utilize the `plot()` function. We can also visualize the PC loadings (Figs. 6.11, 6.12, and 6.13).

Fig. 6.11 Scree plot of the magnitude of the eigenvalues corresponding to the principal components

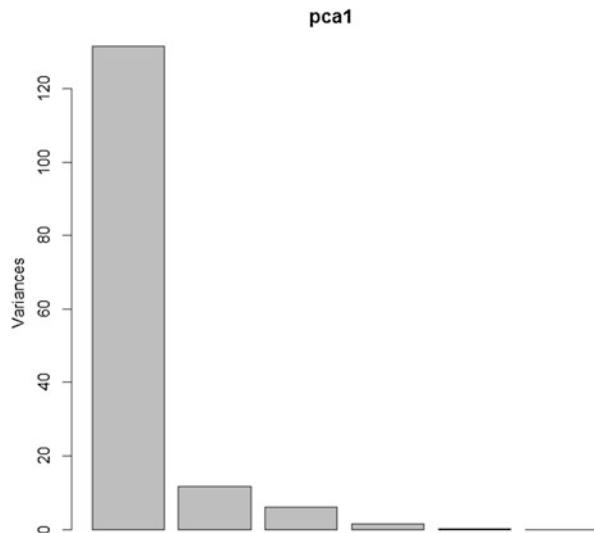


Fig. 6.12 A biplot, enhanced scatterplot, showing both points and vectors representing structure of the data in terms of the projections of the features onto the main two principal component directions

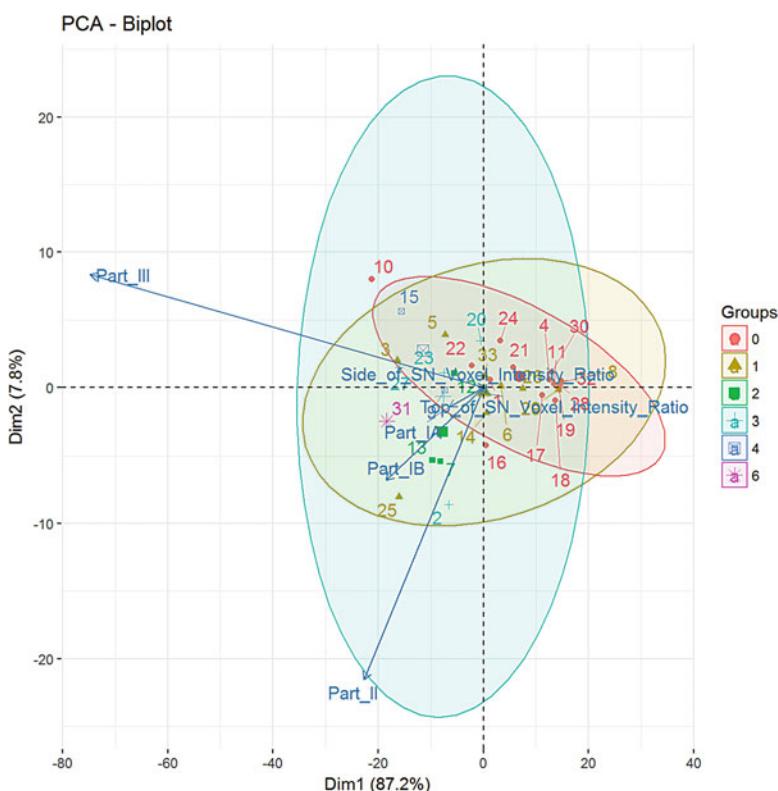
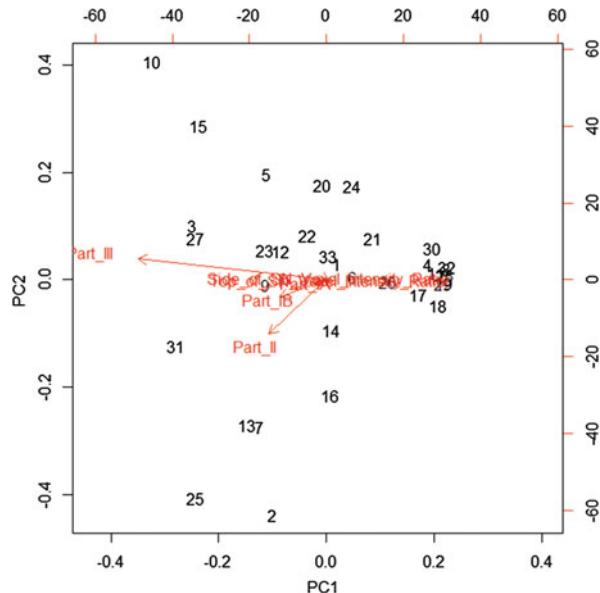


Fig. 6.13 A more elaborate biplot of the same Parkinson's disease dataset

```

plot(pca1)

library(graphics)
biplot(pca1, choices = 1:2, scale = 1, pc.biplot = F)

library("factoextra")
# Data for the supplementary qualitative variables
qualit_vars <- as.factor(pd.sub$Part_IA)
head(qualit_vars)

## [1] 0 3 1 0 1 1
## Levels: 0 1 2 3 4 6

# for plots of individuals
# fviz_pca_ind(pca1, habillage = qualit_vars, addEllipses = TRUE,
ellipse.level = 0.68) +
# theme_minimal()
# for Biplot of individuals and variables
fviz_pca_biplot(pca1, axes = c(1, 2), geom = c("point", "text"),
col.ind = "black", col.var = "steelblue", label = "all",
invisble = "none", repel = T, habillage = qualit_vars,
palette = NULL, addEllipses = TRUE, title = "PCA - Biplot")

```

The histogram plot has a clear “elbow” point at the second PC. Two PCs explains about 95% of the total variation. Thus, we can use the first 2 PCs to represent the data. In this case, the dimension of the data is substantially reduced.

Here, biplot uses PC1 and PC2 as the axes and red vectors to represent the direction of variables after adding loadings as weights. It help us to visualize how the loadings are used to rearrange the structure of the data.

Next, let's try to obtain a bootstrap test for the confidence interval of the explained variance (Fig. 6.14).

```

set.seed(12)
num_boot = 1000
bootstrap_it = function(i) {
  data_resample = pd.sub[sample(1:nrow(pd.sub), nrow(pd.sub), replace=TRUE), ]
  p_resample = princomp(data_resample, cor = T)
  return(sum(p_resample$sdev[1:3]^2)/sum(p_resample$sdev^2))
}

pco = data.frame(per=sapply(1:num_boot, bootstrap_it))
quantile(pco$per, probs = c(0.025, 0.975))

# specify 95-th % Confidence Interval
##      2.5%      97.5%
## 0.8134438 0.9035291

corpp = sum(pca1$sdev[1:3]^2)/sum(pca1$sdev^2)
require(ggplot2)
plot = ggplot(pco, aes(x=pco$per)) +
  geom_histogram() + geom_vline(xintercept=corpp, color='yellow')+
  labs(title = "Percent Var Explained by the first 3 PCs") +
  theme(plot.title = element_text(hjust = 0.5))+
  labs(x='perc of var')
show(plot)

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

```

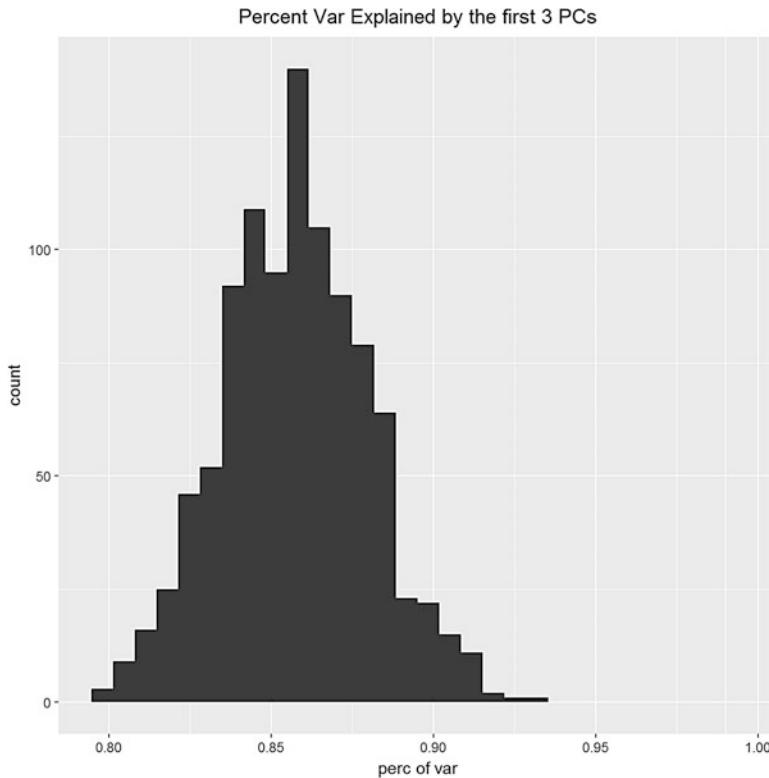


Fig. 6.14 A histogram plot illustrating the proportion of the energy of the original dataset accounted for by the first three principal components

6.6 Independent Component Analysis (ICA)

ICA aims to find basis vectors representing independent components of the original data. For example, this may be achieved by maximizing the norm of the fourth order normalized kurtosis, which iteratively projects the signal on a new basis vector, computes the objective function (e.g., the norm of the kurtosis) of the result, slightly adjusts the basis vector (e.g., by gradient ascent), and recomputes the kurtosis again. The end of this iterative process generates a basis vector corresponding to the highest (residual) kurtosis representing the next independent component.

The process of Independent Component Analysis is to maximize the statistical independence of the estimated components. Assume that each variable X_i is generated by a sum of n independent components.

$$X_i = a_{i,1}s_1 + \cdots + a_{i,n}s_n.$$

Here, X_i is generated by $s_1 : s_n$ and $a_{i,1} : a_{i,n}$ are the corresponding weights. Finally, we rewrite X as

$$X = As,$$

where $X = (X_1, \dots, X_n)^T$, $A = (a_1, \dots, a_n)^T$, $a_i = (a_{i,1}, \dots, a_{i,n})$ and $s = (s_1, \dots, s_n)^T$. Note that s is obtained by maximizing the independence of the components. This procedure is done by maximizing some *independence* objective function.

ICA assumes all of its components (s_i) are non-Gaussian and independent of each other.

We will now introduce the `fastICA` function in R.

```
fastICA(X, n.comp, alg.typ, fun, rownorm, maxit, tol)
```

- **X**: data matrix
- **n.comp**: number of components,
- **alg.type**: components extracted simultaneously (`alg.typ == "parallel"`) or one at a time (`alg.typ == "deflation"`)
- **fun**: functional form of F to approximate to neg-entropy,
- **rownorm**: whether rows of the data matrix X should be standardized beforehand
- **maxit**: maximum number of iterations
- **tol**: a positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged.

Let's generate a correlated X matrix.

```
S <- matrix(runif(10000), 5000, 2)
S[1:10, ]

##          [,1]      [,2]
## [1,] 0.19032887 0.92326457
## [2,] 0.64582044 0.36716717
## [3,] 0.09673674 0.51115358
## [4,] 0.24813471 0.03997883
## [5,] 0.51746238 0.03503276
## [6,] 0.94568595 0.86846372
## [7,] 0.29500222 0.76227787
## [8,] 0.93488888 0.97061365
## [9,] 0.89622932 0.62092241
## [10,] 0.33758057 0.84543862

A <- matrix(c(1, 1, -1, 3), 2, 2, byrow = TRUE)
X <- S %*% A # In R, "*" and "%*%" indicate "scalar" and matrix multiplication, respectively!
cor(X)

##          [,1]      [,2]
## [1,] 1.0000000 -0.4563297
## [2,] -0.4563297 1.0000000
```

The correlation between two variables is -0.4563297 . Then we can start to fit the ICA model.

```
# install.packages("fastICA")
library(fastICA)
a <- fastICA(X, 2, alg.typ = "parallel", fun = "Logcosh", alpha = 1,
               method = "C", row.norm = FALSE, maxit = 200,
               tol = 0.0001)
```

To visualize how *correlated* the pre-processed data is and how *independent* the resulting S is, we can draw the following two plots (Fig. 6.15).

```
par(mfrow = c(1, 2))
plot(a$X, main = "Pre-processed data")
plot(a$S, main = "ICA components")
```

Finally, we can check the correlation of two components in the ICA result, S ; it is nearly 0.

```
cor(a$S)
## [,1]      [,2]
## [1,] 1.000000e+00 -7.677818e-16
## [2,] -7.677818e-16 1.000000e+00
```

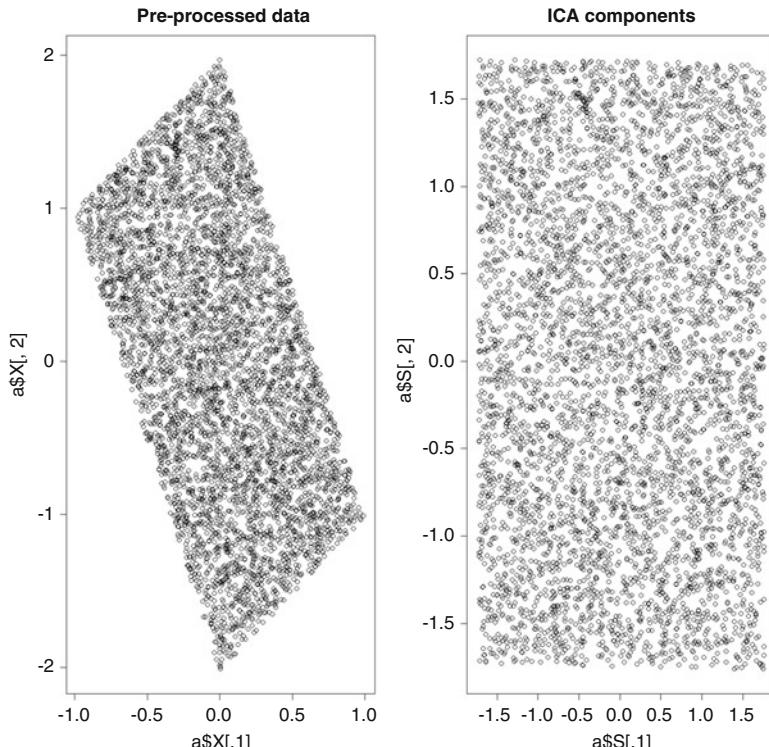


Fig. 6.15 Scatterplots of the raw data (left) illustrating intrinsic relation in the simulated bivariate data and the ICA-transformed data (right) showing random scattering

To do a more interesting example, we can use the `pd.sub` dataset (Parkinson's disease). It has six variables and the correlation is relatively high. After fitting the ICA model, the components are nearly independent.

```
cor(pd.sub)

##                                     Top_of_SN_Voxel_Intensity_Ratio
## Top_of_SN_Voxel_Intensity_Ratio          1.00000000
## Side_of_SN_Voxel_Intensity_Ratio        0.54747225
## Part_IA                                -0.10144191
## Part_IB                                -0.26966299
## Part_II                                 -0.04358545
## Part_III                                -0.33921790
##                                     Side_of_SN_Voxel_Intensity_Ratio
## Top_of_SN_Voxel_Intensity_Ratio          0.5474722
## Side_of_SN_Voxel_Intensity_Ratio        1.0000000
## Part_IA                                -0.2157587
## Part_IB                                -0.4438992
## Part_II                                 -0.3766388
## Part_III                                -0.5226128
##                                     Part_IA      Part_IB      Part_II
## Top_of_SN_Voxel_Intensity_Ratio  -0.1014419 -0.2696630 -0.04358545
## Side_of_SN_Voxel_Intensity_Ratio  -0.2157587 -0.4438992 -0.37663875
## Part_IA                                1.0000000  0.4913169  0.50378157
## Part_IB                                0.4913169  1.0000000  0.57987562
## Part_II                                 0.5037816  0.5798756  1.00000000
## Part_III                                0.5845831  0.6735584  0.63901337
##                                     Part_III
## Top_of_SN_Voxel_Intensity_Ratio  -0.3392179
## Side_of_SN_Voxel_Intensity_Ratio  -0.5226128
## Part_IA                                0.5845831
## Part_IB                                0.6735584
## Part_II                                 0.6390134
## Part_III                                1.0000000

a1<-fastICA(pd.sub, 2, alg.typ = "parallel", fun = "Logcosh", alpha = 1,
             method = "C", row.norm = FALSE, maxit = 200,
             tol = 0.0001)
par(mfrow = c(1, 2))
cor(a1$x)

##          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 1.0000000  0.5474722 -0.1014419 -0.2696630 -0.04358545 -0.3392179
## [2,] 0.54747225 1.0000000 -0.2157587 -0.4438992 -0.37663875 -0.5226128
## [3,] -0.10144191 -0.2157587 1.0000000  0.4913169  0.50378157  0.5845831
## [4,] -0.26966299 -0.4438992  0.4913169  1.0000000  0.57987562  0.6735584
## [5,] -0.04358545 -0.3766388  0.5037816  0.5798756  1.00000000  0.6390134
## [6,] -0.33921790 -0.5226128  0.5845831  0.6735584  0.63901337  1.0000000

cor(a1$x)

##          [,1]      [,2]
## [1,] 1.000000e+00 1.088497e-15
## [2,] 1.088497e-15 1.000000e+00
```

Notice that we only have two ICA components instead of six variables, successfully reducing the dimension of the data.

6.7 Factor Analysis (FA)

Similar to ICA and PCA, FA tries to find components in the data. As a generalization of PCA, FA requires that the number of components is smaller than the original number of variables (or columns of the data matrix). FA optimization relies on iterative perturbations with full-dimensional Gaussian noise and maximum-likelihood estimation where every observation in the data represents a sample point in a subspace. Whereas PCA assumes the noise is spherical, Factor Analysis allows the noise to have an arbitrary diagonal covariance matrix and estimates the subspace as well as the noise covariance matrix.

Under FA, the centered data can be expressed in the following form:

$$x_i - \mu_i = l_{i,1}F_1 + \cdots + l_{i,k}F_k + \epsilon_i = LF + \epsilon_i,$$

where $i \in 1, \dots, p, j \in 1, \dots, k, k < p$ and ϵ_i are independently distributed error terms with zero mean and finite variance.

Let's do FA in R with function `factanal()`. According to PCA, our `pd`.`sub` dataset can explain 95% of variance with the first two principal components. This suggest that we might need two factors in FA. We can double check that by the following commands (Fig. 6.16).

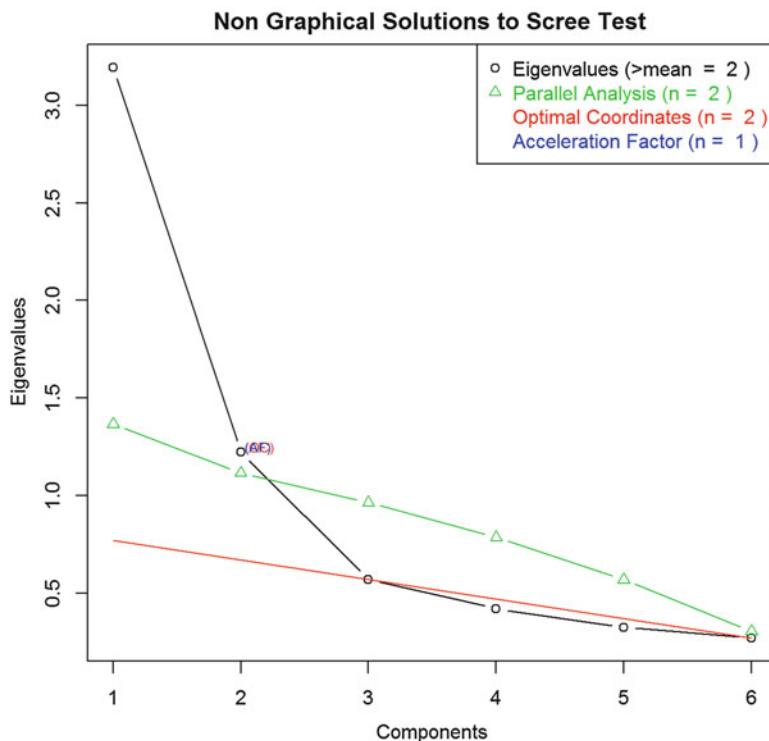


Fig. 6.16 Scree plots of various solutions

```

## Report For a nScree Class
##
## Details: components
##
## Eigenvalues Prop Cumu Par.Analysis Pred.eig      OC Acc.factor      AF
## 1      3   1   1           1      1      NA (< AF)
## 2      1   0   1           1      1 (< OC)      1
## 3      1   0   1           1      0      1
## 4      0   0   1           1      0      0
## 5      0   0   1           1      NA      0
## 6      0   0   1           0      NA      NA
##
## Number of factors retained by index
##
## noc naf nparallel nkaiser
## 1   2   1       2       2

```

Three out of four rules in Cattell's Scree test summary suggest we should use two factors. Thus, in function `factanal()` we use `factors=2` and the `varimax` rotation as performing arithmetic to obtain a new set of factor loadings. Oblique `promax` and `Procrustes` rotation (projecting the loadings to a target matrix with a simple structure) are two other commonly used matrix rotations.

```

fit<-factanal(pd.sub, factors=2, rotation="varimax")
# fit<-factanal(pd.sub, factors=2, rotation="promax") # the most popular oblique rotation; And fitting a simple structure
fit

## Call:
## factanal(x = pd.sub, factors = 2, rotation = "varimax")
## 

## Uniquenesses:
##   Top_of_SN_Voxel_Intensity_Ratio Side_of_SN_Voxel_Intensity_Ratio
##                               0.018                      0.534
##                               Part_IA                  Part_IB
##                               0.571                      0.410
##                               Part_II                  Part_III
##                               0.392                      0.218
## 
## Loadings:
##   Factor1 Factor2
##   Top_of_SN_Voxel_Intensity_Ratio      0.991
##   Side_of_SN_Voxel_Intensity_Ratio -0.417   0.540
##   Part_IA                      0.650
##   Part_IB                      0.726 -0.251
##   Part_II                      0.779
##   Part_III                      0.825 -0.318
## 
##   Factor1 Factor2
##   SS Loadings      2.412   1.445
##   Proportion Var  0.402   0.241
##   Cumulative Var  0.402   0.643
## 
## Test of the hypothesis that 2 factors are sufficient.
## The chi square statistic is 1.35 on 4 degrees of freedom.
## The p-value is 0.854

```

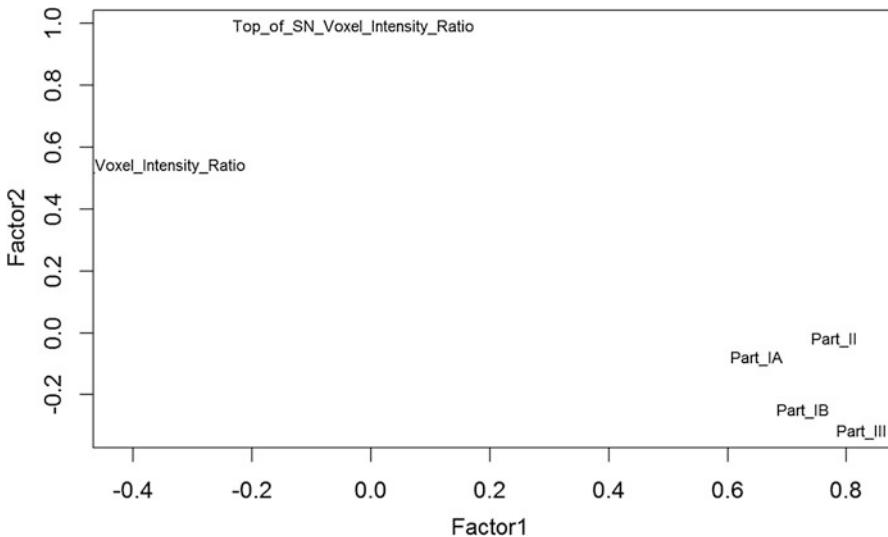


Fig. 6.17 Factor analysis results projecting the key features on the first two factor dimensions

Here the p -value 0.854 is very large, suggesting that we failed to reject the null-hypothesis that two factors are sufficient. We can also visualize the loadings for all the variables (Fig. 6.17).

```
Load <- fit$loadings
plot(Load, type="n") # set up plot
text(Load, labels=colnames(pd.sub), cex=.7) # add variable names
```

This plot displays *factors 1* and *2* on the *x*-axis and *y*-axis, respectively.

6.8 Singular Value Decomposition (SVD)

SVD is a factorization of a real or complex matrix. If we have a data matrix X with n observation and p variables, it can be factorized into the following form:

$$X = UDV^T,$$

where U is a $n \times p$ unitary matrix, that $U^T U = I$, D is a $p \times p$ diagonal matrix, and V^T is a $p \times p$ unitary matrix, which is the conjugate transpose of the $n \times n$ unitary matrix, V . Thus, we have $V^T V = I$.

SVD is closely linked to PCA (when correlation matrix is used for calculation). U are the left singular vectors. D are the singular values. U gives PCA scores. V are the right singular vectors-PCA loadings.

We can compare the output from the `svd()` function and the `princomp()` function (another R function for PCA). Still, we are using the `pd.sub` dataset. Before the SVD, we need to scale our data matrix.

```
#SVD output
df<-nrow(pd.sub)-1
zvars<-scale(pd.sub)
z.svd<-svd(zvars)
z.svd$d</sqrt(df)

## [1] 1.7878123 1.1053808 0.7550519 0.6475685 0.5688743 0.5184536
z.svd$v

##          [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
## [1,]  0.2555204  0.71258155 -0.37323594  0.10487773 -0.4773992  0.22073161
## [2,]  0.3855208  0.47213743  0.35665523 -0.43312945  0.5581867  0.04564469
## [3,] -0.3825033  0.37288211  0.70992668  0.31993403 -0.2379855 -0.22728693
## [4,] -0.4597352  0.09803466 -0.11166513 -0.79389290 -0.2915570 -0.22647775
## [5,] -0.4251107  0.34167997 -0.46424927  0.26165346  0.5341197 -0.36505061
## [6,] -0.4976933  0.06258370  0.03872473 -0.01769966  0.1832789  0.84438182

#PCA output
pca2<-princomp(pd.sub, cor=T)
pca2

## Call:
## princomp(x = pd.sub, cor = T)
##
## Standard deviations:
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
## 1.7878123 1.1053808 0.7550519 0.6475685 0.5688743 0.5184536
##
## 6 variables and 33 observations.

Loadings(pca2)

## 
## Loadings:
##                                         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6
## Top_of_SN_Voxel_Intensity_Ratio -0.256 -0.713 -0.373 -0.105 0.477 -0.221
## Side_of_SN_Voxel_Intensity_Ratio -0.386 -0.472  0.357  0.433 -0.558
## Part_IA                      0.383 -0.373  0.710 -0.320 0.238  0.227
## Part_IB                      0.460   -0.112  0.794  0.292  0.226
## Part_II                      0.425 -0.342 -0.464 -0.262 -0.534  0.365
## Part_III                     0.498   -0.183 -0.844
##                                         Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6
## SS Loadings      1.000  1.000  1.000  1.000  1.000
## Proportion Var  0.167  0.167  0.167  0.167  0.167  0.167
## Cumulative Var  0.167  0.333  0.500  0.667  0.833  1.000
```

When the correlation matrix is used for calculation (`cor=T`), the V matrix of SVD contains the loadings of the PCA.

6.9 SVD Summary

Intuitively, the SVD approach $X = UDV^T$ represents a composition of the (centered!) data into three geometrical transformations: a rotation or reflection (U), a scaling (D), and a rotation or reflection (V). Here we assume that the data X stores samples/cases in rows and variables/features in columns. If these are reversed, then the interpretations of the U and V matrices reverse as well.

- The columns of V represent the directions of the principal axes, the columns of UD are the principal components, and the singular values in D are related to the eigenvalues of data variance-covariance matrix (Σ) via $\lambda_i = \frac{d_i^2}{n-1}$, where the eigenvalues λ_i capture the magnitude of the data variance in the respective PCs.
- The standardized scores are given by columns of $\sqrt{n-1}U$ and the corresponding loadings are given by columns of $\frac{1}{n-1}VD$. However, these “loadings” are not the principal directions. The requirement for X to be centered is needed to ensure that the covariance matrix $Cov(X) = \frac{1}{n-1}X^T X$.
- Alternatively, to perform PCA on the *correlation matrix* (instead of the covariance matrix), the columns of X need to be *scaled* (centered and standardized).
- To reduce the data dimensionality from p to $k < p$, we multiply the first k columns of U by the $k \times k$ upper-left corner of the matrix D to get an $n \times k$ matrix $U_k D_k$ containing the first k PCs.
- Multiplying the first k PCs by their corresponding principal directions V_k^T reconstructs the original data from the first k PCs, $X_k = U_k D_k V_k^T$, with the lowest possible reconstruction error.
- Typically, we have more subjects/cases (n) than variables/features ($p < n$). As $U_{n \times n}$ and $V_{p \times p}$, the last $n - p > 0$ columns of U may be trivial (zeros). It’s customary to drop the zero columns of U for $n \gg p$ to avoid dealing with unnecessarily large (trivial) matrices.

6.10 Case Study for Dimension Reduction (Parkinson’s Disease)

Step 1: Collecting Data

The data we will be using in this case study is the Clinical, Genetic and Imaging Data for Parkinson’s Disease in the SOCR website. A detailed data explanation is available online http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_BiomedBigMetadata. Let’s import the data into R.

```

# Loading required package: xml2
wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_Bio
medBigMetadata")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...
pd_data <- html_table(html_nodes(wiki_url, "table")[[1]])
head(pd_data); summary(pd_data)

##   Cases L_caudate_ComputeArea L_caudate_Volume R_caudate_ComputeArea
## 1      2                  597                  767                  855
## 2      2                  597                  767                  855
## 3      2                  597                  767                  855
## 4      2                  597                  767                  855
## 5      3                  604                  873                  935
## 6      3                  604                  873                  935
...
##   chr17_rs11868035_GT UPDRS_part_I UPDRS_part_II UPDRS_part_III Time
## 1                  0                  1                  12                  1     0
## 2                  0                  1                  12                  1     6
## 3                  0                  1                  12                  1    12
## 4                  0                  1                  12                  1    18
## 5                  1                  0                  19                  22     0
## 6                  1                  0                  19                  22     6
...
##   Cases      L_caudate_ComputeArea L_caudate_Volume
##  Min.   : 2.0  Min.   :525.0      Min.   :719.0
##  1st Qu.:158.0 1st Qu.:582.0      1st Qu.:784.0
##  Median :363.5 Median :600.0      Median :800.0
##  Mean   :346.1 Mean   :600.4      Mean   :800.3
##  3rd Qu.:504.0 3rd Qu.:619.0      3rd Qu.:819.0
##  Max.   :692.0  Max.   :667.0      Max.   :890.0
...
##   Min.   : 0.0
##  1st Qu.: 4.5
##  Median : 9.0
##  Mean   : 9.0
##  3rd Qu.:13.5
##  Max.   :18.0

```

Step 2: Exploring and Preparing the Data

To make sure that the data is ready for further modeling, we need to fix a few things. First, the Dx variable, or diagnosis, is a factor. We need to change it to a numeric variable. Second, we don't need the patient ID and time variable in the dimension reduction procedures.

```

pd_data$Dx <- gsub("PD", 1, pd_data$Dx)
pd_data$Dx <- gsub("HC", 0, pd_data$Dx)
pd_data$Dx <- gsub("SWEDD", 0, pd_data$Dx)
pd_data$Dx <- as.numeric(pd_data$Dx)
attach(pd_data)
pd_data<-pd_data[, -c(1, 33)]

```

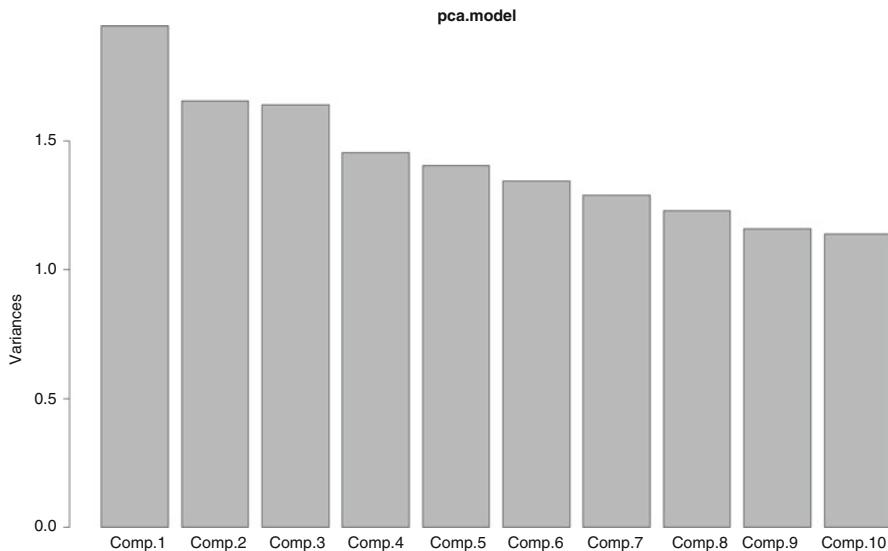


Fig. 6.18 Barplot illustrating the decay of the eigenvectors corresponding to the PCA linear transformation of the variables in the Parkinson's disease dataset (Figs. 6.19 and 6.20)

Step 3: Training a Model on the Data

1. PCA

Now we start the process of fitting a PCA model. Here we will use the `princomp()` function and use the correlation rather than the covariance matrix for calculation.

```

pca.model <- princomp(pd_data, cor=TRUE)
summary(pca.model) # pc loadings (i.e., eigenvector columns)

## Importance of components:
##                                         Comp.1      Comp.2      Comp.3      Comp.4
## Standard deviation      1.39495952 1.28668145 1.28111293 1.2061402
## Proportion of Variance 0.06277136 0.05340481 0.05294356 0.0469282
## Cumulative Proportion  0.06277136 0.11617617 0.16911973 0.2160479
##                                         Comp.5      Comp.6      Comp.7      Comp.8      Comp.9
## Standard deviation      1.18527282 1.15961464 1.135510 1.10882348 1.0761943
## Proportion of Variance 0.04531844 0.04337762 0.041593 0.03966095 0.037361
## Cumulative Proportion  0.26136637 0.30474399 0.346337 0.38599794 0.423359
##                                         Comp.10     Comp.11     Comp.12     Comp.13
## Standard deviation      1.06687730 1.05784209 1.04026215 1.03067437
## Proportion of Variance 0.03671701 0.03609774 0.03490791 0.03426741
## Cumulative Proportion  0.46007604 0.49617378 0.53108169 0.56534910
##                                         Comp.14     Comp.15     Comp.16     Comp.17
## Standard deviation      1.0259684 0.99422375 0.97385632 0.96688855
## Proportion of Variance 0.0339552 0.03188648 0.03059342 0.03015721
## Cumulative Proportion  0.5993043 0.63119078 0.66178421 0.69194141
##                                         Comp.18     Comp.19     Comp.20     Comp.21

```

```

## Standard deviation 0.92687735 0.92376374 0.89853718 0.88924412
## Proportion of Variance 0.02771296 0.02752708 0.02604416 0.02550823
## Cumulative Proportion 0.71965437 0.74718145 0.77322561 0.79873384
## Comp.22 Comp.23 Comp.24 Comp.25
## Standard deviation 0.87005195 0.86433816 0.84794183 0.82232529
## Proportion of Variance 0.02441905 0.02409937 0.02319372 0.02181351
## Cumulative Proportion 0.82315289 0.84725226 0.87044598 0.89225949
## Comp.26 Comp.27 Comp.28 Comp.29
## Standard deviation 0.80703739 0.78546699 0.77505522 0.76624322
## Proportion of Variance 0.02100998 0.01990188 0.01937776 0.01893963
## Cumulative Proportion 0.91326947 0.93317135 0.95254911 0.97148875
## Comp.30 Comp.31
## Standard deviation 0.68806884 0.64063259
## Proportion of Variance 0.01527222 0.01323904
## Cumulative Proportion 0.98676096 1.00000000

plot(pca.model)

biplot(pca.model)

fviz_pca_biplot(pca.model, axes = c(1, 2), geom = "point",
col.ind = "black", col.var = "steelblue", label = "all",
invisible = "none", repel = F, habillage = pd_data$Sex,
palette = NULL, addEllipses = TRUE, title = "PCA - Biplot")

```

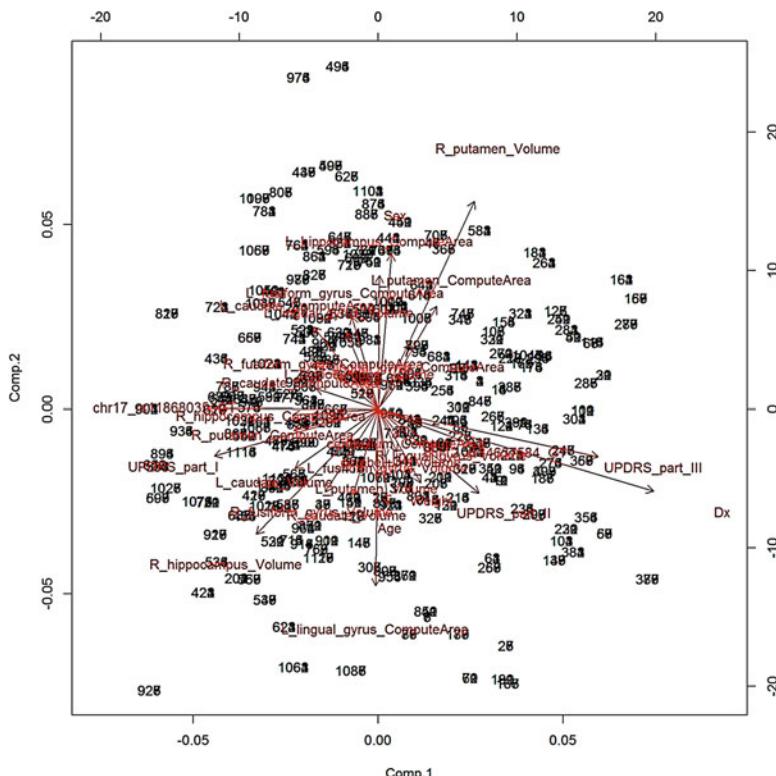


Fig. 6.19 Biplot of the PD variables onto the first two principle axes

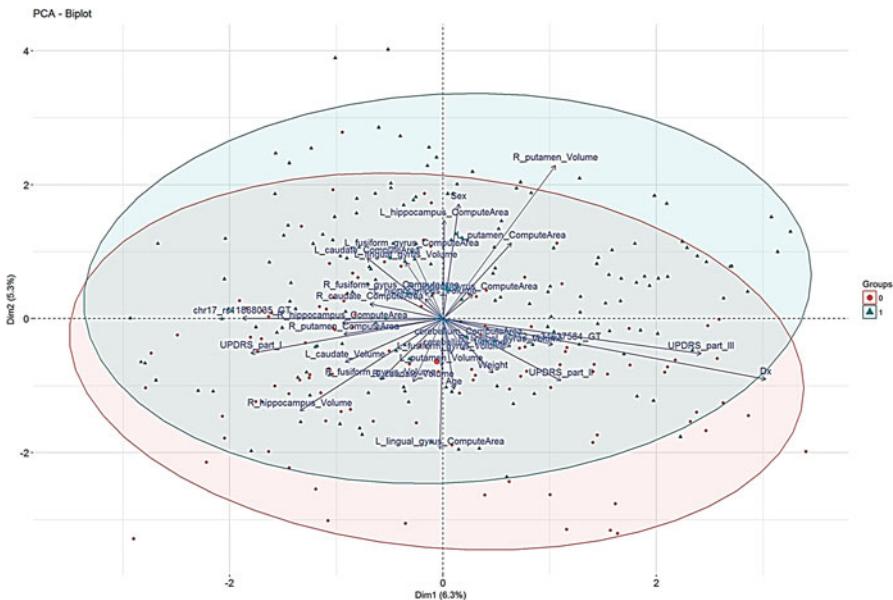


Fig. 6.20 Enhanced biplot of the PD data explicitly labeling the patients and control volunteers

We can see that in real world examples PCs do not necessarily have an “elbow” in the scree plot (Fig. 6.18). In our model, each PC explains about the same amount of variation. Thus, it is hard to tell how many PCs, or factors, we need to pick. This would be an ad hoc decision.

2. FA

Let's set up a Cattell's Scree test to determine the number of factors first.

```
ev <- eigen(cor(pd_data)) # get eigenvalues
ap <- parallel(subject=nrow(pd_data), var=ncol(pd_data), rep=100, cent=.05)
nS <- nScree(x=ev$values, aparallel=ap$eigen$qevpea)
summary(nS)

## Report For a nScree Class
##
## Details: components
##
##   Eigenvalues Prop Cumu Par.Analysis Pred.eig      OC Acc.factor      AF
## 1          2   0     0           1      2 (< OC)      NA (< AF)
## 2          2   0     0           1      2                   0
## 3          2   0     0           1      1                   0
## 4          1   0     0           1      1                   0
## 5          1   0     0           1      1                   0
...
...
```

```

## 30          0    0    1          1      NA          0
## 31          0    0    1          1      NA      NA
##
##
## Number of factors retained by index
##
## noc naf nparallel nkaiser
## 1   1   1      14      14

```

Although the Cattell's Scree test suggest that we should use 14 factors, the real fit shows 14 is not enough. Previous PCA results suggest we need around 20 PCs to obtain a cumulative variance of 0.6. After a few trials, we find that 19 factors can pass the chi square test for sufficient number of factors at 0.05 level.

```

fa.model<-factanal(pd_data, 19, rotation="varimax")
fa.model

##
## Call:
## factanal(x = pd_data, factors = 19, rotation = "varimax")
##
## Uniquenesses:
##          L_caudate_ComputeArea          L_caudate_Volume
##                0.840                      0.005
##          R_caudate_ComputeArea          R_caudate_Volume
##                0.868                      0.849
##          L_putamen_ComputeArea          L_putamen_Volume
##                0.791                      0.702
##          R_putamen_ComputeArea          R_putamen_Volume
##                0.615                      0.438
##          L_hippocampus_ComputeArea      L_hippocampus_Volume
##                0.476                      0.777
##          R_hippocampus_ComputeArea      R_hippocampus_Volume
##                0.798                      0.522
##          cerebellum_ComputeArea        cerebellum_Volume
##                0.137                      0.504
##          L_lingual_gyrus_ComputeArea    L_lingual_gyrus_Volume
##                0.780                      0.698
##          R_lingual_gyrus_ComputeArea    R_lingual_gyrus_Volume
##                0.005                      0.005
##          L_fusiform_gyrus_ComputeArea    L_fusiform_gyrus_Volume
##                0.718                      0.559
##          R_fusiform_gyrus_ComputeArea    R_fusiform_gyrus_Volume
##                0.663                      0.261
##          Sex                          Weight
##                0.829                      0.005
##          Age                          Dx
##                0.005                      0.005
##          chr12_rs34637584_GT          chr17_rs11868035_GT
##                0.638                      0.721
##          UPDRS_part_I                UPDRS_part_II
##                0.767                      0.826
##          UPDRS_part_III               0.616
##
```

```

## Loadings:
## Factor1 Factor2 Factor3 Factor4 Factor5
## L_caudate_ComputeArea
## L_caudate_Volume
## R_caudate_ComputeArea
## R_caudate_Volume
## L_putamen_ComputeArea
## L_putamen_Volume
## R_putamen_ComputeArea
## R_putamen_Volume
## L_hippocampus_ComputeArea
## L_hippocampus_Volume
## R_hippocampus_ComputeArea -0.102
## R_hippocampus_Volume
## cerebellum_ComputeArea
## cerebellum_Volume
## L_lingual_gyrus_ComputeArea 0.107
## L_lingual_gyrus_Volume
## R_lingual_gyrus_ComputeArea
## R_lingual_gyrus_Volume 0.989
## L_fusiform_gyrus_ComputeArea
## L_fusiform_gyrus_Volume
## R_fusiform_gyrus_ComputeArea
## R_fusiform_gyrus_Volume 0.983
## Sex -0.111
## Weight 0.983
## Age
## Dx 0.965
## chr12_rs34637584_GT 0.124
## chr17_rs11868035_GT -0.303
## UPDRS_part_I -0.260
## UPDRS_part_II
## UPDRS_part_III 0.332 0.104
## Factor6 Factor7 Factor8 Factor9 Factor10
## L_caudate_ComputeArea -0.101
## L_caudate_Volume
...
## Factor1 Factor2 Factor3 Factor4 Factor5 Factor6 Factor7
## SS Loadings 1.282 1.029 1.026 1.019 1.013 1.011 0.921
## Proportion Var 0.041 0.033 0.033 0.033 0.033 0.033 0.030
## Cumulative Var 0.041 0.075 0.108 0.140 0.173 0.206 0.235
## Factor8 Factor9 Factor10 Factor11 Factor12 Factor13
## SS Loadings 0.838 0.782 0.687 0.647 0.615 0.587
## Proportion Var 0.027 0.025 0.022 0.021 0.020 0.019
## Cumulative Var 0.263 0.288 0.310 0.331 0.351 0.370
## Factor14 Factor15 Factor16 Factor17 Factor18 Factor19
## SS Loadings 0.569 0.566 0.547 0.507 0.475 0.456
## Proportion Var 0.018 0.018 0.018 0.016 0.015 0.015
## Cumulative Var 0.388 0.406 0.424 0.440 0.455 0.470
## Test of the hypothesis that 19 factors are sufficient.
## The chi square statistic is 54.51 on 47 degrees of freedom.
## The p-value is 0.211

```

This data matrix has relatively low correlation. Thus, it is not suitable for ICA.

```
cor(pd_data)[1:10, 1:10]
```

	<i>L_caudate_ComputeArea</i>	<i>L_caudate_Volume</i>
## <i>L_caudate_ComputeArea</i>	1.00000000	0.05794916
## <i>L_caudate_Volume</i>	0.057949162	1.00000000
## <i>R_caudate_ComputeArea</i>	-0.060576361	0.01076372
## <i>R_caudate_Volume</i>	0.043994457	0.07245568
## <i>L_putamen_ComputeArea</i>	0.009640983	-0.06632813
## <i>L_putamen_Volume</i>	-0.064299184	-0.11131525
## <i>R_putamen_ComputeArea</i>	0.040808105	0.04504867
## <i>R_putamen_Volume</i>	0.058552841	-0.11830387
## <i>L_hippocampus_ComputeArea</i>	-0.037932760	-0.04443615
## <i>L_hippocampus_Volume</i>	-0.042033469	-0.04680825
...		
## <i>L_caudate_ComputeArea</i>	0.04080810	0.058552841
## <i>L_caudate_Volume</i>	0.04504867	-0.118303868
## <i>R_caudate_ComputeArea</i>	0.07864348	0.007022844
## <i>R_caudate_Volume</i>	0.05428747	-0.094336376
## <i>L_putamen_ComputeArea</i>	0.09049611	0.176353726
## <i>L_putamen_Volume</i>	0.09093926	-0.057687648
## <i>R_putamen_ComputeArea</i>	1.00000000	0.052245264
## <i>R_putamen_Volume</i>	0.05224526	1.000000000
## <i>L_hippocampus_ComputeArea</i>	-0.05508472	0.131800075
## <i>L_hippocampus_Volume</i>	-0.08866344	-0.001133570
## <i>L_hippocampus_ComputeArea</i>	-0.037932760	-0.04203347
## <i>L_caudate_Volume</i>	-0.044436146	-0.04680825
## <i>R_caudate_ComputeArea</i>	0.051359613	0.08578833
## <i>R_caudate_Volume</i>	0.006123355	-0.07791361
## <i>L_putamen_ComputeArea</i>	0.094604791	-0.06442537
## <i>L_putamen_Volume</i>	0.025303302	0.04041557
## <i>R_putamen_ComputeArea</i>	-0.055084723	-0.08866344
## <i>R_putamen_Volume</i>	0.131800075	-0.00113357
## <i>L_hippocampus_ComputeArea</i>	1.000000000	-0.02633816
## <i>L_hippocampus_Volume</i>	-0.026338163	1.000000000

6.11 Assignments: 6. Dimensionality Reduction

6.11.1 Parkinson's Disease Example

Apply principal component analysis (PCA), singular value decomposition (SVD), independent component analysis (ICA), and factor analysis (FA) to reduce the dimensionality of the PD data. Interpret the results.

6.11.2 Allometric Relations in Plants Example

Load Data

Load Allometric Relations in Plants data and perform a proper type conversion, e.g., convert “Province” and “Born”.

Dimensionality Reduction

- Apply Principal Component Analysis protocol.
 - Generate a data summary
 - Apply `prcomp`
 - Report the rotations (scores)
 - Display screen plot
 - Select the number of PCs and employ a bootstrap test
 - Apply `factoextra` to draw biplot and grouped by Province/Sites
- Perform SVD and ICA and compare the results of PCA.
 - Use these three variables `L`, `M`, `D` to perform ICA and show pair plots before ICA and after ICA. (Hint: `scatter3dplot()` may be helpful, which you saw in Chap. 5.)
- Perform factor analysis.
 - Use `require(nFactors)` to determine the number of the factors and show a scree plot as stated in notes
 - Use `factanal()` to apply FA and compare the rotation `varimax` and `promax`
 - Report the loadings and consider an appropriate visualization method.
- Interpret the findings in the context of the case-study.

References

- Jolliffe, I.T. (2002) Principal Component Analysis, Springer.
Karhunen, J. and Hyvärinen, A. (2001) Independent Component Analysis, Wiley-Interscience.
Cattell, R.B. (1952) Factor analysis. New York: Harper.

Chapter 7

Lazy Learning: Classification Using Nearest Neighbors



In the next several Chapters, we will concentrate on various progressively advanced machine learning, classification and clustering techniques. There are two categories of learning techniques we will explore: supervised (human-guided) classification and unsupervised (fully-automated) clustering. In general, supervised *classification* aims to identify or predict predefined classes and label new objects as members of specific classes. Whereas, unsupervised *clustering* attempts to group objects into sets, without knowing *a priori* labels, and determine relationships between objects.

In the context of machine learning, classification refers to supervised learning and clustering to unsupervised learning.

Unsupervised classification refers to methods where the outcomes (groupings with common characteristics) are automatically derived based on intrinsic affinities and associations in the data without prior human indication of clustering. Unsupervised learning is purely based on input data (X) without corresponding output labels. The goal is to model the underlying structure, affinities, or distribution in the data in order to learn more about its intrinsic characteristics. It is called unsupervised learning because there are no *a priori* correct answers and there is no human guidance. Algorithms are left to their own devices to discover and present the interesting structure in the data. *Clustering* (discovers the inherent groupings in the data) and *association* (discovers association rules that describe the data) represent the core unsupervised learning problems. The **k-means** clustering and the **Apriori association rule** provide solutions to unsupervised learning problems.

Supervised classification methods utilize user provided labels representative of specific classes associated with concrete observations, cases, or units. These training classes/outcomes are used as references for the classification. Many problems can be addressed by decision-support systems utilizing combinations of supervised and unsupervised classification processes. Supervised learning involves input variables (X) and an outcome variable (Y) to learn mapping functions from the input to the output: $Y = f(X)$. The goal is to approximate the mapping function so that when it is applied to new (validation) data (Z) it (accurately) predicts the (expected) outcome variables (Y). It is called supervised learning because the learning process is

Table 7.1 Summary of supervised classification and unsupervised clustering techniques

Inference	Outcome	Supervised	Unsupervised
Classification & prediction	Binary	Classification-rules, OneR, kNN, NaiveBayes, Decision-Tree, C5.0, AdaBoost, XGBoost, LDA/QDA, Logit/Poisson, SVM	<i>Apriori</i> , Association-rules, k-Means, NaiveBayes
Classification & prediction	Categorical	Regression modeling & forecasting	<i>Apriori</i> , Association-rules, k-Means, NaiveBayes
Regression modeling	Real quantitative	LDA/QDA, SVM, Decision-Tree, NeuralNet	(MLR) Regression modeling, Regression modeling tree, <i>Apriori</i> /Association-rules

supervised by initial training labels guiding and correcting the learning until the algorithm achieves an acceptable level of performance.

Regression (output variable is a real value) and *classification* (output variable is a category) problems represent the two types of supervised learning. Examples of supervised machine learning algorithms include *Linear regression* and *Random forest*. Both provide solutions for regression problems, but *Random forest* also provides solutions to classification problems.

Just like categorization of exploratory data analytics (Chap. 4) is challenging, so is systematic codification of machine learning techniques. Table 7.1 attempts to provide a rough representation of common machine learning methods. However, it is not really intended to be a gold-standard protocol for choosing the best analytical method. Before you settle on a specific strategy for data analysis, you should always review the data characteristics in light of the assumptions of each technique and assess the potential to gain new knowledge or extract valid information from applying a specific technique (Table 7.1).

Many of these will be discussed in later Chapters. In this Chapter, we will present step-by-step the *k-nearest neighbor* (*kNN*) algorithm. Specifically, we will show (1) data retrieval and normalization; (2) splitting the data into *training* and *testing* sets; (3) fitting models on the training data; (4) evaluating model performance on testing data; (5) improving model performance; and (6) determining optimal values of *k*.

In Chap. 14, we will present detailed strategies, and evaluation metrics, to assess the performance of all clustering and classification methods.

7.1 Motivation

Classification tasks could be very difficult when the features and target classes are numerous, complicated, or extremely difficult to understand. In those scenarios where the items of similar class type tend to be homogeneous, nearest neighbor classifying method are well-suited because assigning unlabeled examples to most similar labeled examples would be fairly easy.

Such classification methods can help us to understand the story behind the complicated case-studies. This is because machine learning methods generally have no distribution assumptions. However, this non-parametric manner makes the methods rely heavily on large and representative training datasets.

7.2 The kNN Algorithm Overview

The kNN algorithm involves the following steps:

1. Create a training dataset that has classified examples labeled by nominal variables and different features in ordinal or numerical variables.
 2. Create a test dataset containing unlabeled examples with similar features as the training data.
 3. Given a predetermined number k , match each test record with k training records that are “nearest” in similarity.
 4. Assigning the class that contains the majority of the k training records to the test record.

The Fig. 7.1 demonstration shows the dynamic classification of the mouse location (x, y) coordinates that are used as new data. You can specify the number of points (n) and the number of nearest neighbors (k). The app automatically computes the neighborhood size and the corresponding label (color) for the mouse location and draws the connecting edges to the nearest neighbors showing the dynamic classification process.

7.2.1 Distance Function and Dummy Coding

How to measure the similarity between records? We can measure the similarity as the geometric distance between the two records. There are many distance functions to choose from. Traditionally, we use *Euclidean distance* as our distance function.

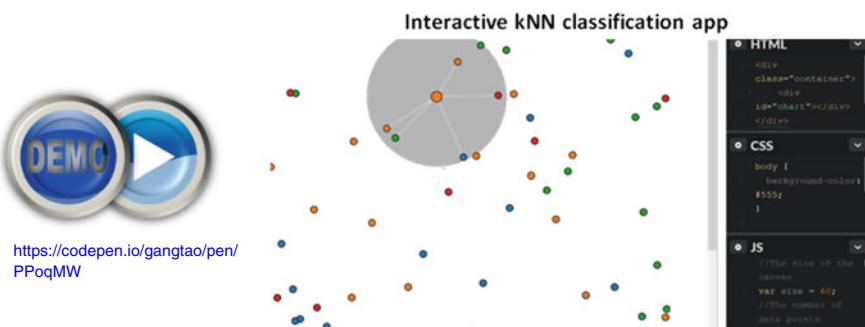


Fig. 7.1 Live Demo: k-nearest neighbor classification webapp

If we use a line to link the two dots created by the test record and the training record in n dimensional space, the length of the line is the Euclidean distance. Suppose a, b both have n features with coordinates (a_1, a_2, \dots, a_n) and (b_1, b_2, \dots, b_n) . A simple Euclidian distance could be defined by:

$$dist(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \dots + (a_n - b_n)^2}.$$

When we have nominal features, it requires a little trick to apply the Euclidean distance formula. We could create dummy variables as indicators of the nominal feature. The dummy variable would equal to one when we have the feature and zero otherwise. We show two examples:

$$\begin{aligned} Gender &= \begin{cases} 0 & X = \text{male} \\ 1 & X = \text{female} \end{cases}, \\ Cold &= \begin{cases} 0 & \text{Temp} \geq 37F \\ 1 & \text{Temp} < 37F \end{cases}. \end{aligned}$$

This allows only binary expressions. If we have multiple nominal categories, just make each one as a dummy variable and apply the Euclidean distance.

7.2.2 Ways to Determine k

The parameter k could be neither too large nor too small. If our k is too large, the test record tends to be classified as the most popular class in the training records rather than the most similar one. On the other hand, if the k is too small, outliers or noisy data, like mislabeling the training data, might lead to errors in predictions.

A common practice is to calculate the square root of the number of training examples and use that number as k .

A more robust way would be to choose several k 's and select the one with best classifying performance.

7.2.3 Rescaling of the Features

Different features might have different scales. For example, we can have a measure of pain scaling from one to ten or one to one hundred. They could be transferred into the same scale. Re-scaling can make each feature contribute to the distance in a relatively equal manner.

7.2.4 Rescaling Formulas

1. min-max normalization

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}.$$

After re-scaling, X_{new} would range between 0 and 1. It measures the distance between each value and its minimum as a percentage. The larger a percentage the further a value is from the minimum. 100% means that the value is at the maximum.

2. z-Score Standardization

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{SD}(X)}.$$

This is based on the properties of normal distribution that we have talked about in Chap. 3. After z-score standardization, the re-scaled feature will have unbounded range. This is different from the min-max normalization that has a limited range from 0 to 1. However, after z-score standardization, the new X is assumed to follow a standard normal distribution.

7.3 Case Study

7.3.1 Step 1: Collecting Data

The data we are using for this case study is the “Boys Town Study of Youth Development”, which is the second case study, CaseStudy02_Boystown_Data.csv.

Variables:

- **ID:** Case subject identifier.
- **Sex:** dichotomous variable (1 = male, 2 = female).
- **GPA:** Interval-level variable with range of 0–5 (0- "A" average, 1- "B" average, 2- "C" average, 3- "D" average, 4- "E", 5- "F").
- **Alcohol use:** Interval level variable from 0 to 11 (drink everyday - never drank).
- **Attitudes on drinking in the household:** Alcatt- Interval level variable from 0 to 6 (totally approve - totally disapprove).
- **DadJob:** 1-yes, dad has a job: and 2- no.
- **MomJob:** 1-yes and 2-no.
- **Parent closeness** (example: In your opinion, does your mother make you feel close to her?)
 - Dadclose: Interval level variable 0–7 (usually-never)
 - Momclose: interval level variable 0–7 (usually-never).

- Delinquency:
 - larceny (how many times have you taken things >\$50?): Interval level data 0–4 (never - many times),
 - vandalism: Interval level data 0–7 (never - many times).

7.3.2 Step 2: Exploring and Preparing the Data

First, we need to load in the data and do some data manipulation. We are using the Euclidean distance, so dummy variable should be used. The following code transfers sex, dadjob and momjob into dummy variables.

```
boystown<-read.csv("https://umich.instructure.com/files/399119/download?down
load_frd=1", sep=" ")
boystown$sex<-boystown$sex-1
boystown$dadjob<-1*(boystown$dadjob-2)
boystown$momjob<-1*(boystown$momjob-2)
str(boystown)

## 'data.frame': 200 obs. of 11 variables:
## $ id      : int 1 2 3 4 5 6 7 8 9 10 ...
## $ sex     : num 0 0 0 0 1 1 0 0 1 1 ...
## $ gpa     : int 5 0 3 2 3 3 1 5 1 3 ...
## $ Alcoholuse: int 2 4 2 2 6 3 2 6 5 2 ...
## $ alcatt   : int 3 2 3 1 2 0 0 3 0 1 ...
## $ dadjob   : num 1 1 1 1 1 1 1 1 1 1 ...
## $ momjob   : num 0 0 0 0 1 0 0 0 1 1 ...
## $ dadclose : int 1 3 2 1 2 1 3 6 3 1 ...
## $ momclose : int 1 4 2 2 1 2 1 2 3 2 ...
## $ Larceny  : int 1 0 0 3 1 0 0 0 1 1 ...
## $ vandalism: int 3 0 2 2 2 0 5 1 4 0 ...
```

The `str()` function reports that we have 200 observations and 11 variables. However, the ID variable is not important in this case study so we can delete it. The variable of most interest is GPA. We can classify it into two categories. Whoever gets a "C" or higher will be classified into the "above average" category; Students who have average score below "C" will be in the "average or below" category. These two are the classes of interest for this case study.

```
boystown<-boystown[, -1]
table(boystown$gpa)

##
## 0 1 2 3 4 5
## 30 50 54 40 14 12

boystown$grade<-boystown$gpa %in% c(3, 4, 5)
boystown$grade<-factor(boystown$grade, levels=c(F, T), labels = c("above_avg
", "avg_or_below"))
table(boystown$grade)

##
##      above_avg avg_or_below
##                  134             66
```

Let's look at the proportions for the two categorizes.

```
round(prop.table(table(boystown$grade))*100, digits=1)
##      above_avg avg_or_below
##             67            33
```

We can see that most of the students are above average (67%).

The remaining ten features are all numeric but with different scales. If we use these features directly, the ones with larger scale will have a greater impact on the classification performance. Therefore, re-scaling is needed in this scenario.

```
summary(boystown[c("Alcoholuse", "Larceny", "vandalism")])
##      Alcoholuse      Larceny      vandalism
##  Min.   : 0.00   Min.   :0.00   Min.   :0.0
##  1st Qu.: 2.00   1st Qu.:0.00   1st Qu.:1.0
##  Median : 4.00   Median :1.00   Median :2.0
##  Mean   : 3.87   Mean   :0.92   Mean   :1.9
##  3rd Qu.: 5.00   3rd Qu.:1.00   3rd Qu.:3.0
##  Max.   :11.00   Max.   :4.00   Max.   :7.0
```

7.3.3 Normalizing Data

First let's create a function of our own using the min-max normalization formula. We can check the function using some trial vectors.

```
normalize<-function(x){
  # be careful, the denominator may be trivial!
  return((x-min(x))/(max(x)-min(x)))
}

# some test examples:
normalize(c(1, 2, 3, 4, 5))
## [1] 0.00 0.25 0.50 0.75 1.00

normalize(c(1, 3, 6, 7, 9))
## [1] 0.000 0.250 0.625 0.750 1.000
```

After confirming that it is working properly, we use the `lapply()` function to apply the normalization to each element in a “list.” First, we need to make our dataset into a list. The `as.data.frame()` function converts our data into a data frame, which is a list of equal-length column vectors. Thus, each feature is an element in the list that we can apply the normalization function to.

```
boystown_n<-as.data.frame(lapply(boystown[-11], normalize))
```

Let's see one of the features that have been normalized.

```
summary(boystown_n$Alcoholuse)
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.0000 0.1818 0.3636 0.3518 0.4545 1.0000
```

This looks great! Now we can move to the next step.

7.3.4 Data Preparation: Creating Training and Testing Datasets

We have 200 observations in this dataset. The more data we use to train the algorithm, the more precise the prediction would be. We can use 3/4 of the data for training and the remaining 1/4 for testing.

```
# Ideally, we want to randomly split the raw data into training and testing
# For example: 80% training + 20% testing
# subset_int <- sample(nrow(boystown_n), floor(nrow(boystown_n)*0.8))
# bt_train<- boystown_n [subset_int, ]; bt_test<-boystown_n[-subset_int, ]
# Below, we use a simpler 3:1 split for simplicity
bt_train<-boystown_n[1:150, -11]
bt_test<-boystown_n[151:200, -11]
```

The following step is to extract the labels or classes (column = 11, Delinquency in terms of reoccurring vandalism) for our two subsets.

```
bt_train_labels<-boystown[1:150, 11]
bt_test_labels<-boystown[151:200, 11]
```

7.3.5 Step 3: Training a Model On the Data

We are using the `class` package for the kNN algorithm in R.

```
#install.packages('class', repos = "http://cran.us.r-project.org")
library(class)
```

The function `knn()` has following components:

```
p<-knn(train, test, class, k)
```

- `train`: data frame containing numeric training data (features)
- `test`: data frame containing numeric testing data (features)
- `class/cl`: class for each observation in the training data
- `k`: predetermined integer indication the number of nearest neighbors

The first k we chose should be the square root of our number of observations: $\sqrt{200} \approx 14$.

```
bt_test_pred<-knn(train=bt_train, test=bt_test, cl=bt_train_labels, k=14)
```

7.3.6 Step 4: Evaluating Model Performance

We utilize the `CrossTable()` function in Chap. 3 to evaluate the kNN model. We have two classes in this example. The goal is to create a 2×2 table that shows the

matched true and predicted classes, as well as the unmatched ones. However chi-square values are not needed, so we use option `prop.chisq=False` to suppress reporting them.

```
# install.packages("gmodels", repos="http://cran.us.r-project.org")
library(gmodels)
CrossTable(x=bt_test_labels, y=bt_test_pred, prop.chisq = F)

## 
## 
##   Cell Contents
##   |-----|-----|
##   |           N |-----|
##   |           N / Row Total |-----|
##   |           N / Col Total |-----|
##   |           N / Table Total |-----|
##   |-----|-----|
## 
## 
## Total Observations in Table:  50
## 
## 
##           | bt_test_pred
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       30 |       0 |       30 |
##   |       1.000 |       0.000 |       0.600 |
##   |       0.769 |       0.000 |       |
##   |       0.600 |       0.000 |       |
## -----|-----|-----|-----|
##   avg_or_below |       9 |      11 |       20 |
##   |       0.450 |       0.550 |       0.400 |
##   |       0.231 |       1.000 |       |
##   |       0.180 |       0.220 |       |
## -----|-----|-----|-----|
##   Column Total |      39 |      11 |      50 |
##   |      0.780 |      0.220 |      |
## -----|-----|-----|-----|
```

From the table, the diagonal first row first cell and the second row second cell contain the counts for records that have predicted classes matching the true classes. The other two cells are the counts for unmatched cases. The accuracy in this case is calculated by $\frac{cell[1,1]+cell[2,2]}{total}$. This accuracy will vary each time we run the algorithm. In this situation, we got $accuracy = \frac{cell[1,1]+cell[2,2]}{total} = \frac{41}{50} = 0.82$, however, a previous run generated an $accuracy = \frac{cell[1,1]+cell[2,2]}{total} = \frac{38}{50} = 0.76$.

7.3.7 Step 5: Improving Model Performance

The above Normalization may be suboptimal. We can try an alternative standardization method, e.g., standard Z-score centralization and normalization (via `scale()` method). Let's give it a try:

```
bt_z<-as.data.frame(scale(boystown[, -11]))
summary(bt_z$Alcoholuse)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -2.04800 -0.98960  0.06879  0.00000  0.59800  3.77300
```

The `summary()` shows the re-scaling is working properly. Then, we can proceed to next steps (retraining the kNN, predicting and assessing the accuracy of the results):

```
bt_train<-bt_z[1:150, -11]
bt_test<-bt_z[151:200, -11]
bt_train_labels<-boystown[1:150, 11]
bt_test_labels<-boystown[151:200, 11]
bt_test_pred<-knn(train=bt_train, test=bt_test,
                   cl=bt_train_labels, k=14)
CrossTable(x=bt_test_labels, y=bt_test_pred, prop.chisq = F)

##
##
## Cell Contents
## |-----|
## |           N |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
##
##
## Total Observations in Table:  50
##
##
##          | bt_test_pred
## bt_test_labels |  above_avg | avg_or_below | Row Total |
## -----|-----|-----|-----|
##   above_avg |      30 |       0 |      30 |
##   |      1.000 |  0.000 |  0.600 |
##   |      0.769 |  0.000 |  |
##   |      0.600 |  0.000 |  |
## -----|-----|-----|-----|
##   avg_or_below |      9 |      11 |      20 |
##   |      0.450 |  0.550 |  0.400 |
##   |      0.231 |  1.000 |  |
##   |      0.180 |  0.220 |  |
## -----|-----|-----|-----|
##   Column Total |      39 |      11 |      50 |
##   |      0.780 |  0.220 |  |
## -----|-----|-----|-----|
```

Under the z-score method, the prediction result is similar to the previous run.

7.3.8 Testing Alternative Values of k

Originally, we used the square root of 200 as our k . However, this might not be the best k in this case study. We can test different k 's for their predicting performance.

```
bt_train<-boystown_n[1:150, -11]
bt_test<-boystown_n[151:200, -11]
bt_train_labels<-boystown[1:150, 11]
bt_test_labels<-boystown[151:200, 11]
bt_test_pred1<-knn(train=bt_train, test=bt_test,
                     cl=bt_train_labels, k=1)
bt_test_pred5<-knn(train=bt_train, test=bt_test,
                     cl=bt_train_labels, k=5)
bt_test_pred11<-knn(train=bt_train, test=bt_test,
                      cl=bt_train_labels, k=11)
bt_test_pred21<-knn(train=bt_train, test=bt_test,
                      cl=bt_train_labels, k=21)
bt_test_pred27<-knn(train=bt_train, test=bt_test,
                      cl=bt_train_labels, k=27)
ct_1<-CrossTable(x=bt_test_labels, y=bt_test_pred1, prop.chisq = F)

##      Cell Contents
## |-----|
## |           N |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## 
## 
## Total Observations in Table:  50
## 
## 
##           | bt_test_pred1
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       27 |        3 |      30 |
##   |       0.900 |      0.100 |      0.600 |
##   |       0.818 |      0.176 |      |
##   |       0.540 |      0.060 |      |
## -----|-----|-----|-----|
##   avg_or_below |       6 |       14 |      20 |
##   |       0.300 |      0.700 |      0.400 |
##   |       0.182 |      0.824 |      |
##   |       0.120 |      0.280 |      |
## -----|-----|-----|-----|
##   Column Total |      33 |       17 |      50 |
##   |      0.660 |      0.340 |      |
## -----|-----|-----|-----|
## 
## 
##           | bt_test_pred5
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       27 |        3 |      30 |
##   |       0.900 |      0.100 |      0.600 |
##   |       0.818 |      0.176 |      |
##   |       0.540 |      0.060 |      |
## -----|-----|-----|-----|
##   avg_or_below |       6 |       14 |      20 |
##   |       0.300 |      0.700 |      0.400 |
##   |       0.182 |      0.824 |      |
##   |       0.120 |      0.280 |      |
## -----|-----|-----|-----|
##   Column Total |      33 |       17 |      50 |
##   |      0.660 |      0.340 |      |
## -----|-----|-----|-----|
## 
## 
##           | bt_test_pred11
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       27 |        3 |      30 |
##   |       0.900 |      0.100 |      0.600 |
##   |       0.818 |      0.176 |      |
##   |       0.540 |      0.060 |      |
## -----|-----|-----|-----|
##   avg_or_below |       6 |       14 |      20 |
##   |       0.300 |      0.700 |      0.400 |
##   |       0.182 |      0.824 |      |
##   |       0.120 |      0.280 |      |
## -----|-----|-----|-----|
##   Column Total |      33 |       17 |      50 |
##   |      0.660 |      0.340 |      |
## -----|-----|-----|-----|
## 
## 
##           | bt_test_pred21
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       27 |        3 |      30 |
##   |       0.900 |      0.100 |      0.600 |
##   |       0.818 |      0.176 |      |
##   |       0.540 |      0.060 |      |
## -----|-----|-----|-----|
##   avg_or_below |       6 |       14 |      20 |
##   |       0.300 |      0.700 |      0.400 |
##   |       0.182 |      0.824 |      |
##   |       0.120 |      0.280 |      |
## -----|-----|-----|-----|
##   Column Total |      33 |       17 |      50 |
##   |      0.660 |      0.340 |      |
## -----|-----|-----|-----|
## 
## 
##           | bt_test_pred27
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## -----|-----|-----|-----|
##   above_avg |       27 |        3 |      30 |
##   |       0.900 |      0.100 |      0.600 |
##   |       0.818 |      0.176 |      |
##   |       0.540 |      0.060 |      |
## -----|-----|-----|-----|
##   avg_or_below |       6 |       14 |      20 |
##   |       0.300 |      0.700 |      0.400 |
##   |       0.182 |      0.824 |      |
##   |       0.120 |      0.280 |      |
## -----|-----|-----|-----|
##   Column Total |      33 |       17 |      50 |
##   |      0.660 |      0.340 |      |
## -----|-----|-----|-----|
```

```

## 
## 
## 
##   Total Observations in Table:  50
## 
## 
##          | bt_test_pred5
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## ----- |----- |----- |----- |
##   above_avg |      30 |        0 |      30 |
##   |      1.000 |        0.000 |        0.600 |
##   |      0.857 |        0.000 |        0.600 |
##   |      0.600 |        0.000 |        0.600 |
## ----- |----- |----- |----- |
##   avg_or_below |      5 |      15 |      20 |
##   |      0.250 |      0.750 |      0.400 |
##   |      0.143 |      1.000 |        0.400 |
##   |      0.100 |      0.300 |        0.400 |
## ----- |----- |----- |----- |
##   Column Total |      35 |      15 |      50 |
##   |      0.700 |      0.300 |        0.300 |
## ----- |----- |----- |----- |
## 
## 
##   Cell Contents
## |----- |----- |----- |
## |           N |           |           |
## |           N / Row Total |           |           |
## |           N / Col Total |           |           |
## |           N / Table Total |           |           |
## |----- |----- |----- |
## 
##   Total Observations in Table:  50
## 
##          | bt_test_pred11
## bt_test_labels |   above_avg | avg_or_below |   Row Total |
## ----- |----- |----- |----- |
##   above_avg |      30 |        0 |      30 |
##   |      1.000 |        0.000 |        0.600 |
##   |      0.769 |        0.000 |        0.600 |
##   |      0.600 |        0.000 |        0.600 |
## ----- |----- |----- |----- |
##   avg_or_below |      9 |      11 |      20 |
##   |      0.450 |      0.550 |      0.400 |
##   |      0.231 |      1.000 |        0.400 |
##   |      0.180 |      0.220 |        0.400 |
## ----- |----- |----- |----- |

```

```

##   Column Total |      39 |      11 |      50 |
##   |      0.780 |      0.220 |      |
##   |-----|-----|-----|-----|
ct_21<-CrossTable(x=bt_test_labels, y=bt_test_pred21,
prop.chisq = F)

##   Cell Contents
##   |-----|
##   |      N |
##   |      N / Row Total |
##   |      N / Col Total |
##   |      N / Table Total |
##   |-----|
##   |
##   ## Total Observations in Table: 50
##
##
##          | bt_test_pred21
## bt_test_labels | above_avg | avg_or_below | Row Total |
##   |-----|-----|-----|-----|
##   above_avg |      30 |      0 |      30 |
##   |      1.000 |      0.000 |      0.600 |
##   |      0.714 |      0.000 |      |
##   |      0.600 |      0.000 |      |
##   |-----|-----|-----|-----|
##   avg_or_below |      12 |      8 |      20 |
##   |      0.600 |      0.400 |      0.400 |
##   |      0.286 |      1.000 |      |
##   |      0.240 |      0.160 |      |
##   |-----|-----|-----|-----|
##   Column Total |      42 |      8 |      50 |
##   |      0.840 |      0.160 |      |
##   |-----|-----|-----|-----|
##   |
##   ct_27<-CrossTable(x=bt_test_labels, y=bt_test_pred27,prop.chisq = F)

##
##
##   Cell Contents
##   |-----|
##   |      N |
##   |      N / Row Total |
##   |      N / Col Total |
##   |      N / Table Total |
##   |-----|
##   |
##   ## Total Observations in Table: 50
##
##
##          | bt_test_pred27
## bt_test_labels | above_avg | avg_or_below | Row Total |
##   |-----|-----|-----|-----|
##   above_avg |      30 |      0 |      30 |
##   |      1.000 |      0.000 |      0.600 |
##   |      0.682 |      0.000 |      |
##   |      0.600 |      0.000 |      |
##   |-----|-----|-----|-----|
##   avg_or_below |      14 |      6 |      20 |
##   |      0.700 |      0.300 |      0.400 |
##   |      0.318 |      1.000 |      |
##   |      0.280 |      0.120 |      |
##   |-----|-----|-----|-----|
##   Column Total |      44 |      6 |      50 |
##   |      0.880 |      0.120 |      |
##   |-----|-----|-----|-----|

```

The choice of k in kNN clustering is very important.

```

# install.packages("e1071")
library(e1071)
knntuning = tune.knn(x= bt_train, y = bt_train_labels, k = 1:30)
knntuning

## 
## Parameter tuning of 'knn.wrapper':
## 
## - sampling method: 10-fold cross validation
## 
## - best parameters:
##   k
##   9
## 
## - best performance: 0.1733333

summary(knntuning)

## Parameter tuning of 'knn.wrapper':
## - sampling method: 10-fold cross validation
## - best parameters:
##   k
##   9
## - best performance: 0.1733333
## - Detailed performance results:
##   k      error dispersion
## 1  1 0.2400000 0.08432740
## 2  2 0.2800000 0.10795518
## 3  3 0.2133333 0.10327956
## 4  4 0.2466667 0.04499657
## 5  5 0.1866667 0.10327956
## 6  6 0.1866667 0.11243654
## 7  7 0.1800000 0.10446808
## 8  8 0.1866667 0.12090196
## 9  9 0.1733333 0.10976968
## 10 10 0.2133333 0.12090196
## 11 11 0.2266667 0.12649111
## 12 12 0.2066667 0.11088867
## 13 13 0.2133333 0.11243654
## 14 14 0.2266667 0.13033670
## 15 15 0.2133333 0.12090196
## 16 16 0.2133333 0.09838197
## 17 17 0.2200000 0.10909278
## 18 18 0.2266667 0.11842589
## 19 19 0.2200000 0.10909278
## 20 20 0.2333333 0.11439589
## 21 21 0.2333333 0.11439589
## 22 22 0.2200000 0.08916623
## 23 23 0.2533333 0.10327956
## 24 24 0.2466667 0.10446808
## 25 25 0.2466667 0.11779874
## 26 26 0.2600000 0.11088867
## 27 27 0.2533333 0.11674600
## 28 28 0.2666667 0.10886621
## 29 29 0.2866667 0.11352924
## 30 30 0.2800000 0.11674600

```

It's useful to visualize the error rate against the value of k . This can help us select a k parameter that minimizes the cross-validation (CV) error (Fig. 7.2).

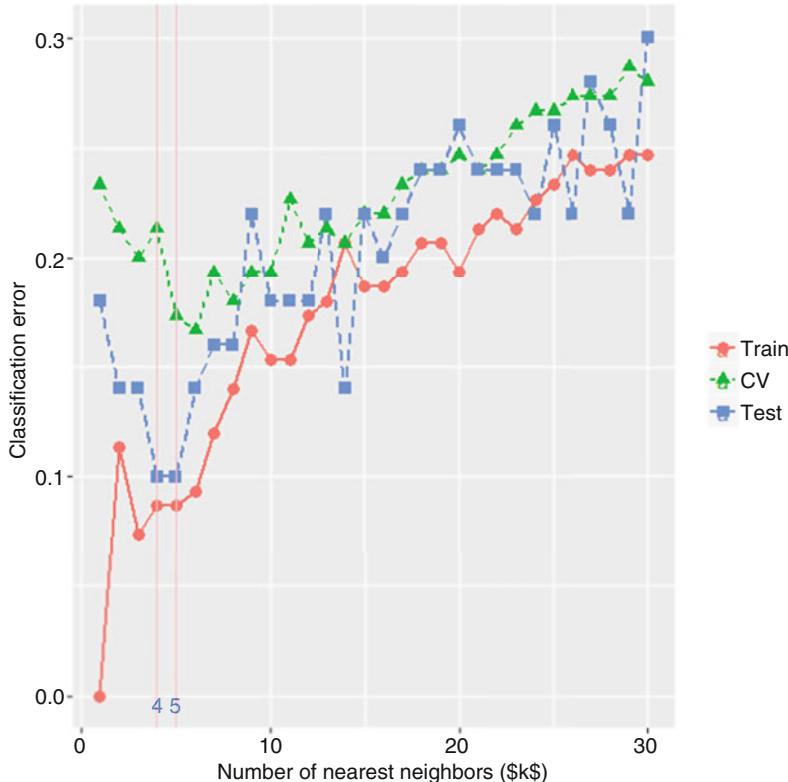


Fig. 7.2 Classification error plots (y-axis) for training data (red), internal statistical cross-validation (green) and external out of box data (blue) against different k -parameters of the kNN method

```

library(class)
library(ggplot2)

# define a function that generates CV folds
cv_partition <- function(y, num_folds = 10, seed = NULL) {
  if(!is.null(seed)) {
    set.seed(seed)
  }
  n <- length(y)

  folds <- split(sample(seq_len(n), n), gl(n = num_folds, k=1, length=n))
  folds <- lapply(folds, function(fold) {
    list(
      training = which(!seq_along(y) %in% fold),
      test = fold
    )
  })
  names(folds) <- paste0("Fold", names(folds))
  return(folds)
}
# Generate 10-folds of the data
folds = cv_partition(bt_train_labels, num_folds = 10)

```

```

# Define a trainingset_cv_error calculation function
train_cv_error = function(K) {
  #Train error
  knnbt = knn(train = bt_train, test = bt_train,
              cl = bt_train_labels, k = K)
  train_error = mean(knnbt != bt_train_labels)

  #CV error
  cverrbt = sapply(folds, function(fold) {
    mean(bt_train_labels[fold$test] != knn(train=bt_train[fold$training,],
    cl = bt_train_labels[fold$training], test = bt_train[fold$test,], k=K))
  })
  cv_error = mean(cverrbt)

  #Test error
  knn.test = knn(train = bt_train, test = bt_test,
                  cl = bt_train_labels, k = K)
  test_error = mean(knn.test != bt_test_labels)
  return(c(train_error, cv_error, test_error))
}

k_err = sapply(1:30, function(k) train_cv_error(k))
df_errs = data.frame(t(k_err), 1:30)
colnames(df_errs) = c('Train', 'CV', 'Test', 'K')

require(ggplot2)
library(reshape2)

dataal <- melt(df_errs, id="K")
ggplot(dataal, aes_string(x="K", y="value", colour="variable",
  group="variable", linetype="variable", shape="variable")) +
  geom_line(size=0.8) + labs(x = "Number of nearest neighbors ($k$)",
  y = "Classification error",
  colour="", group="",
  linetype="", shape="") +
  geom_point(size=2.8) +
  geom_vline(xintercept=4:5, colour = "pink")+
  geom_text(aes(4,0,label = "4", vjust = 1)) +
  geom_text(aes(5,0,label = "5", vjust = 1))

```

7.3.9 Quantitative Assessment (Tables 7.2 and 7.3)

The reader should first review the fundamentals of hypothesis testing inference. Table 7.2 shows the basic components of binary classification, and Table 7.3 reports the results of the classification for several k values.

Table 7.2 Basic evaluation metrics of binary classification

kNN fails to reject	TN	FN
kNN rejects	FP	TP
	Specificity: TN/(TN + FP)	Sensitivity: TP/(TP + FN)

Table 7.3 Summary results of the kNN classification for different values of the parameter k

k value	Total unmatched counts	Accuracy
1	9	0.82
5	5	0.90
11	9	0.82
21	12	0.76
27	14	0.72

Suppose we want to evaluate the kNN model (5) as to how well it predicts the **below-average** boys. Let's report manually some of the accuracy metrics for `model5`. Combining the results, we get the following sensitivity and specificity:

```
# bt_test_pred5<-knn(train=bt_train, test=bt_test, cl=bt_train_labels, k=5)
# ct_5<-CrossTable(x=bt_test_labels, y=bt_test_pred5, prop.chisq = F)
mod5_TN <- ct_5$prop.row[1, 1]
mod5_FP <- ct_5$prop.row[1, 2]
mod5_FN <- ct_5$prop.row[2, 1]
mod5_TP <- ct_5$prop.row[2, 2]

mod5_sensi <- mod5_TN/(mod5_TN+mod5_FP)
mod5_speci <- mod5_TP/(mod5_TP+mod5_FN)
print(paste0("mod5_sensi=", mod5_sensi))

## [1] "mod5_sensi=1"

print(paste0("mod5_speci=", mod5_speci))
## [1] "mod5_speci=0.75"
```

Therefore, **model5** yields a good choice for the number of clusters $k = 5$. Nevertheless, we can always examine further near 5 to get potentially better choices of k .

Another strategy for model validation and improvement involves the use of the `confusionMatrix()` method, which reports several complementary metrics quantifying the performance of the prediction model.

Let's focus on `model5` power to predict `Delinquency` in terms of reoccurring **vandalism**.

```

corr5 <- cor(as.numeric(bt_test_labels), as.numeric(bt_test_pred5))
corr5
## [1] 0.8017837

# plot(as.numeric(bt_test_labels), as.numeric(bt_test_pred5))

# install.packages("caret")
library("caret")

## Loading required package: lattice

# compute the accuracy, LOR, sensitivity/specificity of 3 kNN models

# Model 1: bt_test_pred1
confusionMatrix(as.numeric(bt_test_labels), as.numeric(bt_test_pred1))

## Confusion Matrix and Statistics
##
##          Reference
## Prediction 1 2
##          1 27 3
##          2  6 14
##
##          Accuracy : 0.82
## 95% CI : (0.6856, 0.9142)
##  No Information Rate : 0.66
##  P-Value [Acc > NIR] : 0.009886
##
##          Kappa : 0.6154
##  Mcnemar's Test P-Value : 0.504985
##
##          Sensitivity : 0.8182
##          Specificity : 0.8235
##  Pos Pred Value : 0.9000
##  Neg Pred Value : 0.7000
##          Prevalence : 0.6600
##          Detection Rate : 0.5400
##  Detection Prevalence : 0.6000
##          Balanced Accuracy : 0.8209
##
##          'Positive' Class : 1
## 

# Model 5: bt_test_pred5
confusionMatrix(as.numeric(bt_test_labels), as.numeric(bt_test_pred5))

## Confusion Matrix and Statistics
##
##          Reference
## Prediction 1 2
##          1 30 0
##          2  5 15
##
##          Accuracy : 0.9
## 95% CI : (0.7819, 0.9667)
##  No Information Rate : 0.7
##  P-Value [Acc > NIR] : 0.0007229
##
##          Kappa : 0.7826
##  Mcnemar's Test P-Value : 0.0736383
##

```

```

##           Sensitivity : 0.8571
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 0.7500
##           Prevalence : 0.7000
##           Detection Rate : 0.6000
##           Detection Prevalence : 0.6000
##           Balanced Accuracy : 0.9286
##
##           'Positive' Class : 1
##
# Model 11: bt_test_pred11
confusionMatrix(as.numeric(bt_test_labels), as.numeric(bt_test_pred11))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##           1 30  0
##           2  9 11
##
##           Accuracy : 0.82
##           95% CI : (0.6856, 0.9142)
##           No Information Rate : 0.78
##           P-Value [Acc > NIR] : 0.313048
##
##           Kappa : 0.5946
##           Mcnemar's Test P-Value : 0.007661
##
##           Sensitivity : 0.7692
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 0.5500
##           Prevalence : 0.7800
##           Detection Rate : 0.6000
##           Detection Prevalence : 0.6000
##           Balanced Accuracy : 0.8846
##
##           'Positive' Class : 1
##

```

Finally, we can use a 3D plot to display the results of model5 (mod5_TN, mod5_FN, mod5_FP, mod5_TP), Fig. 7.3.

```

# install.packages("scatterplot3d")
library(scatterplot3d)
grid_xy <- matrix(c(0, 1, 1, 0), nrow=2, ncol=2)
intensity <- matrix(c(mod5_TN, mod5_FN, mod5_FP, mod5_TP), nrow=2, ncol=2)

# scatterplot3d(grid_xy, intensity, pch=16, highlight.3d=TRUE, type="h",
# main="3D Scatterplot")

s3d.dat <- data.frame(cols=as.vector(col(grid_xy)),
                       rows=as.vector(row(grid_xy)),
                       value=as.vector(intensity))
scatterplot3d(s3d.dat, pch=16, highlight.3d=TRUE, type="h", xLab="real",
yLab="predicted", zLab="Agreement", main="3D Scatterplot: Model5 Results
(FP, FN, TP, TN)")

```

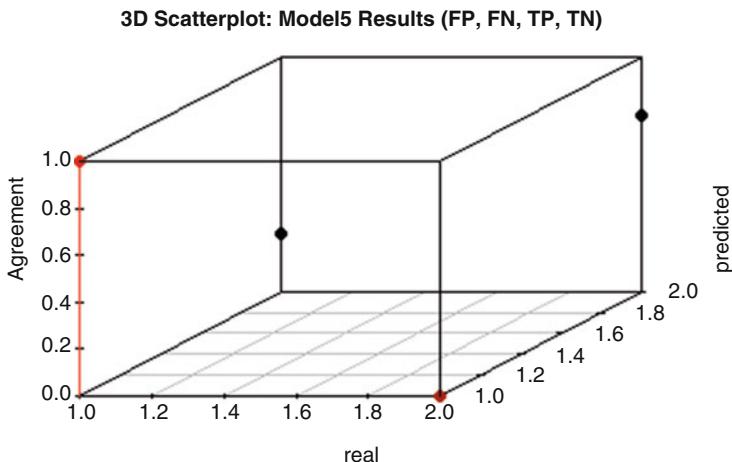


Fig. 7.3 5-NN classification metrics

```
# scatterplot3d(s3d.dat, type="h", lwd=5, pch=" ", xlab="real", ylab="predicted", zlab="Agreement", main="Model5 Results (FP, FN, TP, TN)")
```

7.4 Assignments: 7. Lazy Learning: Classification Using Nearest Neighbors

7.4.1 Traumatic Brain Injury (TBI)

Use the kNN algorithm to provide a classification of the data in the TBI case study, (CaseStudy11_TBI). Determine an appropriate k , train, and evaluate the performance of the classification model on the data. Report some model quality statistics for a couple of different values of k and use these to rank-order (and perhaps plot the classification results of) the models.

7.4.2 Parkinson's Disease

Use 05_PPMI_top_UPDRS_Integrated_LongFormat1 data to practice KNN classification.

7.4.3 KNN Classification in a High Dimensional Space

- **Preprocess the data:** delete the index and ID columns; convert the response variable ResearchGroup to binary 0-1 factor; detect NA (missing) values (impute if necessary)
- **Summarize the dataset:** use `str`, `summary`, `cor`, `ggpairs`
- **Scale/Normalize the data:** As appropriate, scale to [0, 1]; transform $\log(x + 1)$; discretize (0 or 1), etc.
- **Partition data into training and testing sets:** use `set.seed` and `random.sample`, `train:test = 2:1`
- **Select the optimal k for each of the scaled data:** Plot an error graph for k , including three lines: `training_error`, `cross-validation error`, and `testing error`, respectively
- **What is the impact of k ?** Formulate a hypothesis about the relation between k and the error rates. You can try to use `knn.tunning` to verify the results (Hint: select the same folds, all you may obtain a result slightly different)
- **Interpret the results:** Hint: Considering the number of dimension of the data, how many points are necessary to obtain the same density result for 100 dimensional space compared to a 1 dimensional space?
- **Report the error rates** for both the training and the testing data. What do you find?

7.4.4 KNN Classification in a Lower Dimensional Space

Try all the above again but select only the variables: `UPDRS_Part_I_Summary_Score_Baseline`, `UPDRS_Part_I_Summary_Score_Month_24`, `UPDRS_Part_II_Patient_Questionnaire_Summary_Score_Baseline`, `UPDRS_Part_II_Patient_Questionnaire_Summary_Score_Month_24`, `UPDRS_Part_III_Summary_Score_Baseline`, `UPDRS_Part_III_Summary_Score_Month_24`, as predictors. Now, what about the specific k you select and the error rates for each kind of data (original data, normalized data, log-transformed data, and binary data). Comment on any interesting observations.

References

- Kidwell, David A. (2013) *Lazy Learning*, Springer Science & Business Media, ISBN 9401720533, 9789401720533
- Interactive kNN webapp: <https://codepen.io/gangtao/pen/PPoqMW>
- Aggarwal, Charu C. (ed.) (2015) *Data Classification: Algorithms and Applications*, Chapman & Hall/CRC, ISBN 1498760589, 9781498760584

Chapter 8

Probabilistic Learning: Classification Using Naive Bayes



The introduction to Chap. 7 presented the types of machine learning methods and described lazy classification for numerical data. What about nominal features or textual data? In this Chapter, we will begin to explore some classification techniques for categorical data. Specifically, we will (1) present the Naive Bayes algorithm; (2) review its assumptions; (3) discuss Laplace estimation; and (4) illustrate the Naive Bayesian classifier on a Head and Neck Cancer Medication case-study.

Later, in Chap. 20, we will also discuss text mining and natural language processing of unstructured text data.

8.1 Overview of the Naive Bayes Algorithm

Start by reviewing the basics of probability theory and Bayesian inference.

Bayes classifiers use training data to calculate an observed probability of each class based on all the features. The probability links feature values to classes like a map. When labeling the test data, we utilize the feature values in the test data and the “map” to classify our test data with the most likely class. This idea seems simple but the corresponding algorithmic implementations might be very sophisticated.

The best scenario of accurately estimating the probability of an outcome-class map is when all features in Bayes classifiers attribute to the class simultaneously. The Naive Bayes algorithm is frequently used for text classifications. The maximum *a posteriori* assignment to the class label is based on obtaining the conditional probability density function for each feature given the value of the class variable.

8.2 Assumptions

Naive Bayes is named for its “naive” assumptions. Its most important assumption is that all of the features are *equally important* and *independent*. This rarely happens in real world data. However, sometimes even when the assumptions are violated, Naive Bayes still performs fairly accurately, particularly when the number of features p is large. This is why the Naive Bayes algorithm may be used as a powerful text classifier.

There are interesting relations between QDA (Quadratic Discriminant Analysis), LDA (Linear Discriminant Analysis), and Naive Bayes classification. Additional information about LDA and QDA is available online (http://wiki.socr.umich.edu/index.php/SMHS_BigDataBigSci_CrossVal_LDA_QDA).

8.3 Bayes Formula

Let’s first define the set-theoretic Bayes formula. We assume that B_i ’s are mutually exclusive events, for all $i = 1, 2, \dots, n$, where n represents the number of features. If A and B are two events, the Bayes conditional probability formula is as follows:

$$\text{Posterior Probability} = \frac{\text{likelihood} \times \text{Prior Probability}}{\text{Marginal Likelihood}}$$

Symbolically,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

When B_i ’s represent a partition of the event space, $S = \cup B_i$ and $B_i \cap B_j = \emptyset \forall i \neq j$. So we have:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B|B_1) \times P(B_1) + P(B|B_2) \times P(B_2) \dots + P(B|B_n) \times P(B_n)}.$$

Now, let’s represent the Bayes formula in terms of classification using observed features. Having observed n features, F_i , for each of K possible class outcomes, C_k . The Bayesian model may be reformulate to make it more tractable using the Bayes’ theorem, by decomposing the conditional probability.

$$P(C_k | F_1, \dots, F_n) = \frac{P(F_1, \dots, F_n | C_k)P(C_k)}{P(F_1, \dots, F_n)}.$$

In the above expression, only the numerator depends on the class label, C_k , as the values of the features F_i are observed (or imputed) making the denominator constant. Let's focus on the numerator.

The numerator essentially represents the joint probability model:

$$P(F_1, \dots, F_n | C_k) P(C_k) = \underbrace{P(F_1, \dots, F_n, C_k)}_{\text{joint model}}$$

Repeatedly using the chain rule and the definition of conditional probability simplifies this to:

$$\begin{aligned} P(F_1, \dots, F_n, C_k) &= P(F_1 | F_2, \dots, F_n, C_k) \times P(F_2, \dots, F_n, C_k) = \\ &= P(F_1 | F_2, \dots, F_n, C_k) \times P(F_2 | F_3, \dots, F_n, C_k) \times P(F_3, \dots, F_n, C_k) = \\ &= P(F_1 | F_2, \dots, F_n, C_k) \times P(F_2 | F_3, \dots, F_n, C_k) \times P(F_3 | F_4, \dots, F_n, C_k) \\ &\quad \times P(F_4, \dots, F_n, C_k) = \\ &= \dots = \\ &= P(F_1 | F_2, \dots, F_n, C_k) \times P(F_2 | F_3, \dots, F_n, C_k) \times P(F_3 | F_4, \dots, F_n, C_k) \times \dots \\ &\quad \times P(F_n | C_k) \times P(C_k) \end{aligned}$$

Note that the “naive” qualifier in the *Naive Bayes classifier* name is attributed to the oversimplification of the conditional probability. Assuming each feature F_i is conditionally statistical independent of every other feature F_j , $\forall j \neq i$, given the category C_k , we get:

$$P(F_i | F_{i+1}, \dots, F_n, C_k) = P(F_i | C_k).$$

This reduces the joint probability model to:

$$P(F_1, \dots, F_n, C_k) = P(F_1 | C_k) \times P(F_2 | C_k) \times P(F_3 | C_k) \times \dots \times P(F_n | C_k) \times P(C_k)$$

Therefore, the joint model is:

$$P(F_1, \dots, F_n, C_k) = P(C_k) \prod_{i=1}^n P(F_i | C_k)$$

Essentially, we express the probability of class level L given an observation, represented as a set of *independent features* F_1, F_2, \dots, F_n . Then the posterior probability that the observation is in class L is equal to:

$$P(C_L | F_1, \dots, F_n) = \frac{P(C_L) \prod_{i=1}^n P(F_i | C_L)}{\prod_{i=1}^n P(F_i)},$$

where the denominator, $\prod_{i=1}^n P(F_i)$, is a scaling factor that represents the marginal probability of observing all features jointly.

For a given case $X = (F_1, F_2, \dots, F_n)$, i.e., given vector of *features*, the naive Bayes classifier assigns the **most likely class** \hat{C} by calculating $\frac{P(C_L) \prod_{i=1}^n P(F_i|C_L)}{\prod_{i=1}^n P(F_i)}$ for all class labels L , and then assigning the class \hat{C} corresponding to the maximum posterior probability. Analytically, \hat{C} is defined by:

$$\hat{C} = \arg \max_L \frac{P(C_L) \prod_{i=1}^n P(F_i|C_L)}{\prod_{i=1}^n P(F_i)}.$$

As the denominator is static for L , the posterior probability above is maximized when the numerator is maximized, i.e., $\hat{C} = \operatorname{argmax}_L P(C_L) \prod_{i=1}^n P(F_i|C_L)$.

The contingency table below illustrates schematically how the Bayesian, marginal, conditional, and joint probabilities may be calculated for a finite number of features (columns) and classes (rows).

Features/ Classes	F_1	F_2	...	F_n	Total
C_1	Marginal $P(C_1)$
C_2	Joint $P(C_2, F_n)$...
...
C_L	Conditional $P(F_1 C_L) = \frac{P(F_1, C_L)}{P(C_L)}$
Total		Marginal $P(F_2)$	N

In the [DSPA Appendix](#), we provide additional technical details, code, and applications of Bayesian simulation, modeling and inference.

8.4 The Laplace Estimator

If at least one $P(F_i | C_L) = 0$, then $P(C_L | F_1, \dots, F_n) = 0$, which means the probability of being in this class is zero. However, $P(F_i | C_L) = 0$ could be result from a random chance in picking the training data.

One of the solutions to this scenario is **Laplace estimation**, also known as Laplace smoothing, which can be accomplished in two ways. One is to add small number to each cell in the frequency table, which allows each class-feature combination to be at least one in the training data. Then $P(F_i | C_L) > 0$ for all i . Another strategy is to add some small value, ϵ , to the numerator and denominator when calculating the posterior probability. Note that these small perturbations of the denominator should be larger than the changes in the numerator to avoid trivial (0) posterior for another class.

8.5 Case Study: Head and Neck Cancer Medication

8.5.1 Step 1: Collecting Data

We utilize the Inpatient Head and Neck Cancer Medication data for this case study, which is the case study 14 in our data archive.

Variables:

- **PID:** coded patient ID.
- **ENC_ID:** coded encounter ID.
- **Seer_stage:** SEER cancer stage (0 = In situ, 1 = Localized, 2 = Regional by direct extension, 3 = Regional to lymph nodes, 4 = Regional (both codes 2 and 3), 5 = Regional, NOS, 7 = Distant metastases/systemic disease, 8 = Not applicable, 9 = Unstaged, unknown, or unspecified). See: <http://seer.cancer.gov/tools/ssm>.
- **Medication_desc:** description of the chemical composition of the medication.
- **Medication_summary:** brief description about medication brand and usage.
- **Dose:** the dosage in the medication summary.
- **Unit:** the unit for dosage in the Medication_summary.
- **Frequency:** the frequency of use in the Medication_summary.
- **Total_dose_count:** total dosage count according to the Medication_summary.

8.5.2 Step 2: Exploring and Preparing the Data

Let's load our data first.

```
hn_med<-read.csv("https://umich.instructure.com/files/1614350/download?downl
oad_frd=1", stringsAsFactors = FALSE)
str(hn_med)

## 'data.frame': 662 obs. of 9 variables:
## $ PID           : int 10000 10008 10029 10063 10071 10103 1012 1013
5 10136 10143 ...
## $ ENC_ID        : int 46836 46886 47034 47240 47276 47511 3138 4773
9 47744 47769 ...
## $ seer_stage    : int 1 1 4 1 9 1 1 1 9 1 ...
## $ MEDICATION_DESC : chr "ranitidine" "heparin injection" "ampicillin/
sulbactam IVPB UH" "fentaNYL injection UH" ...
## $ MEDICATION_SUMMARY: chr "(Zantac) 150 mg tablet oral two times a day"
"5,000 unit subcutaneous three times a day" "(Unasyn) 15 g IV every 6 hours"
"25 - 50 microgram IV every 5 minutes PRN severe pain\nMaximum dose 200 mcg
Per PACU protocol" ...
## $ DOSE          : chr "150" "5000" "1.5" "50" ...
## $ UNIT          : chr "mg" "unit" "g" "microgram" ...
## $ FREQUENCY     : chr "two times a day" "three times a day" "every
6 hours" "every 5 minutes" ...
## $ TOTAL_DOSE_COUNT : int 5 3 11 2 1 2 2 6 15 1 ...
```

Change the `seer_stage` (cancer stage indicator) variable into a factor.

```
hn_med$seer_stage <- factor(hn_med$seer_stage)
str(hn_med$seer_stage)

##  Factor w/ 9 Levels "0","1","2","3",...: 2 2 5 2 9 2 2 2 9 2 ...





```

Data Preparation: Processing Text Data for Analysis

As you can see, the `medication_summary` contains a great amount of text. We should do some text mining to prepare the data for analysis. In R, the `tm` package is a good choice for text mining.

```
# install.packages("tm", repos = "http://cran.us.r-project.org")
# requires R V.3.3.1 +
```

The first step for text mining is to convert text features (text elements) into a `corpus` object, which is a collection of text documents.

```
hn_med_corpus<-Corpus(VectorSource(hn_med$MEDICATION_SUMMARY))
print(hn_med_corpus)
```

After we construct the `corpus` object, we could see that we have 662 documents. Each document represents an encounter (e.g., notes on medical treatment) for a patient.

```
inspect(hn_med_corpus[1:3])
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document Level (indexed): 0
## Content: documents: 3
##
## [1] (Zantac) 150 mg tablet oral two times a day
## [2] 5,000 unit subcutaneous three times a day
## [3] (Unasyn) 15 g IV every 6 hours

hn_med_corpus[[1]]$content
## [1] "(Zantac) 150 mg tablet oral two times a day"
hn_med_corpus[[2]]$content
## [1] "5,000 unit subcutaneous three times a day"
hn_med_corpus[[3]]$content
## [1] "(Unasyn) 15 g IV every 6 hours"
```

There are unwanted punctuation and other symbols in the corpus document that we want to remove. We use the `tm::tm_map()` function for the cleaning.

```
corpus_clean <- tm_map(hn_med_corpus, toLower)
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
corpus_clean <- tm_map(corpus_clean, removeNumbers)
# corpus_clean <- tm_map(corpus_clean, PlainTextDocument)
```

The above lines of code changed all the characters to lower case, removed all punctuations and extra white spaces (typically created by deleting punctuations), and removed numbers (we could also convert the corpus to plain text).

```
inspect(corpus_clean[1:3])
## <<SimpleCorpus>>
## Metadata: corpus specific: 1, document level (indexed): 0
## Content: documents: 3
##
## [1] zantac mg tablet oral two times a day
## [2] unit subcutaneous three times a day
## [3] unasyn g iv every hours
corpus_clean[[1]]$content
## [1] "zantac mg tablet oral two times a day"
corpus_clean[[2]]$content
## [1] "unit subcutaneous three times a day"
corpus_clean[[3]]$content
## [1] "unasyn g iv every hours"
```

The `DocumentTermMatrix()` function can tokenize the medication summary into words. It can count frequent terms in each document in the corpus object.

```
hn_med_dtm<-DocumentTermMatrix(corpus_clean)
```

Data Preparation: Creating Training and Test Datasets

Just like in Chap. 7, we need to separate the dataset into training and test subsets. We have to subset the raw data with other features, the corpus object and the document term matrix.

```

set.seed(12)
# 80% training + 20% testing

subset_int <- sample(nrow(hn_med), floor(nrow(hn_med)*0.8))

hn_med_train<-hn_med[subset_int, ]
hn_med_test<-hn_med[-subset_int, ]
hn_med_dtm_train<-hn_med_dtm[subset_int, ]
hn_med_dtm_test<-hn_med_dtm[-subset_int, ]
corpus_train<-corpus_clean[subset_int]
corpus_test<-corpus_clean[-subset_int]

# hn_med_train<-hn_med[1:562, ]
#hn_med_test<-hn_med[563:662, ]
# hn_med_dtm_train<-hn_med_dtm[1:562, ]
# hn_med_dtm_test<-hn_med_dtm[563:662, ]
#corpus_train<-corpus_clean[1:562]
#corpus_test<-corpus_clean[563:662]

```

Let's examine the distribution of **seer stages** in the training and test datasets.

```

prop.table(table(hn_med_train$seer_stage))

##
##          0          1          2          3          4          5
## 0.03024575 0.38374291 0.08317580 0.14555766 0.06616257 0.03402647
##          7          8          9
## 0.13421550 0.02268431 0.10018904

prop.table(table(hn_med_test$seer_stage))

##
##          0          1          2          3          4          5
## 0.03759398 0.46616541 0.06766917 0.09774436 0.08270677 0.00000000
##          7          8          9
## 0.12030075 0.01503759 0.11278195

```

We can separate (dichotomize) the **seer_stage** into two categories:

- *No stage* or *early stage* cancer, and
- *later stage* cancer.

```

hn_med_train$stage<-hn_med_train$seer_stage %in% c(4, 5, 7)
hn_med_train$stage<-factor(hn_med_train$stage, Levels=c(F, T), labels = c("early_stage", "later_stage"))
hn_med_test$stage<-hn_med_test$seer_stage %in% c(4, 5, 7)
hn_med_test$stage<-factor(hn_med_test$stage, Levels=c(F, T), labels = c("early_stage", "later_stage"))
prop.table(table(hn_med_train$stage))

## early_stage later_stage
## 0.7655955 0.2344045

prop.table(table(hn_med_test$stage))

## early_stage later_stage
## 0.7969925 0.2030075

```

Visualizing Text Data: Word Clouds

A word cloud can help us visualize text data. More frequent words would have larger fonts in the figure, while less common words appear in smaller fonts. There is a `wordcloud` package in R that is commonly used for creating these figures (Figs. 8.1, 8.2, 8.3).

```
# install.packages("wordcloud", repos = "http://cran.us.r-project.org")
library(wordcloud)

wordcloud(corpora.train, min.freq = 40, random.order = FALSE)
```

The `random.order=FALSE` option made more frequent words appear in the middle. `min.freq=40` option sets the cutoff word frequency to be at least 40 times in the corpus object. Therefore, the words must be appear in at least 40 medication summaries to be shown on the graph.

We can also visualize the difference between early stages and later stages using this type of graph (Figs. 8.2 and 8.3).

Fig. 8.1 A wordle diagram representing the common terms (frequency exceeding 40) in the head and neck (H&N) text corpus

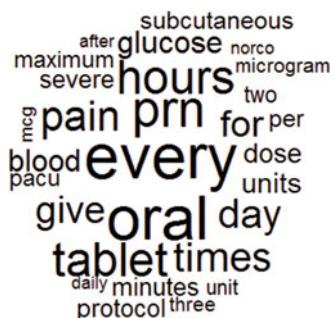
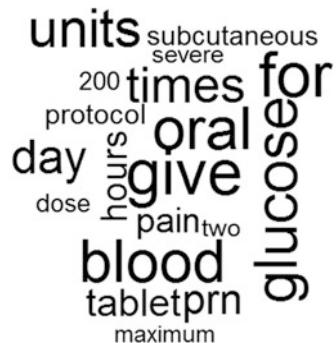


Fig. 8.2 Wordle plot of the common terms included in the medical treatment summary corpus of the **early stage** cancer patients



Fig. 8.3 Wordle plot of the common terms included in the medical treatment summary corpus of the **later stage** cancer patients (compare to Fig. 8.2)



```
early<-subset(hn_med_train, stage=="early_stage")
Later<-subset(hn_med_train, stage=="Later_stage")
wordCloud(early$MEDICATION_SUMMARY, max.words = 20)
wordCloud(Later$MEDICATION_SUMMARY, max.words = 20)
```

We can see that the frequent words are somewhat different in the medication summary between early stage and later stage patients.

Data Preparation: Creating Indicator Features for Frequent Words

For simplicity, we utilize the medication summary as the only feature to classify cancer stages. You may recall that in Chap. 7 we used features for classifications. *In this study, we are going to make the frequencies of words into features.*

```
summary(findFreqTerms(hn_med_dtm_train, 5))
##      Length     Class      Mode
##      103 character character

hn_med_dict <- as.character(findFreqTerms(hn_med_dtm_train, 5))
hn_train <- DocumentTermMatrix(corpus_train, list(dictionary=hn_med_dict))
hn_test <- DocumentTermMatrix(corpus_test, list(dictionary=hn_med_dict))
```

The above code limits the document term matrix with words that have appeared in at least five different documents. This created 103 features for us to use.

The Naive Bayes classifier trains on data with categorical features. Thus, we need to transform our word count features into categorical data. A way to do this is to change the count into an indicator of whether this word appears. We can create a function of our own to deal with this.

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, 1, 0)
  x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
  return(x)
}
```

An important statement is `x<-ifelse(x>0, 1, 0)`. This is saying that if we have an `x` that is greater than 0, we assign value 1 to it, otherwise the value is set to 0.

Now let's apply our own function `convert_counts()` on each column (`MARGIN=2`) of the training and testing datasets.

```
hn_train <- apply(hn_train, MARGIN = 2, convert_counts)
hn_test <- apply(hn_test, MARGIN = 2, convert_counts)
```

So far, we successfully created indicators for words that appeared at least in five different documents in the training data.

8.5.3 Step 3: Training a Model on the Data

The package we will use for Naive Bayes classifier is called `e1071`.

```
# install.packages("e1071", repos = "http://cran.us.r-project.org")
library(e1071)
```

The function `NaiveBayes()` has following components:

```
m<-naiveBayes(train, class, laplace=0)
```

- **train**: data frame containing numeric training data (features)
- **class**: factor vector with the class for each row in the training data.
- **laplace**: positive double controlling Laplace smoothing; default is zero and disables Laplace smoothing.

Let's build our classifier first.

```
hn_classifier <- naiveBayes(hn_train, hn_med_train$stage)
```

Then, we can use the classifier to make predictions using `predict()`. Recall that when we presented the AdaBoost example in Chap. 3, we saw the basic mechanism of machine-learning training, prediction and assessment.

The function `predict()` has the following components:

```
p<-predict(m, test, type="class")
```

- m: classifier trained by `NaiveBayes()`
- test: test data frame or matrix
- type: either "class" or "raw" specifies whether the predictions should be the most likely class value or the raw predicted probabilities.

```
hn_test_pred<-predict(hn_classifier, hn_test)
```

8.5.4 Step 4: Evaluating Model Performance

Similarly to the approach in Chap. 7, we use cross table to compare predicted class and the true class of our test dataset.

```
library(gmodels)
CrossTable(hn_test_pred, hn_med_test$stage)

##   Cell Contents
## |               N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## Total Observations in Table: 133
## | hn_med_test$stage
## hn_test_pred | early_stage | Later_stage |   Row Total |
## -----|-----|-----|-----|
## early_stage |      90 |      24 |      114 |
## |      0.008 |      0.032 |      |
## |      0.789 |      0.211 |      0.857 |
## |      0.849 |      0.889 |      |
## |      0.677 |      0.180 |      |
## -----|-----|-----|-----|
## Later_stage |      16 |       3 |      19 |
## |      0.049 |      0.190 |      |
## |      0.842 |      0.158 |      0.143 |
## |      0.151 |      0.111 |      |
## |      0.120 |      0.023 |      |
## -----|-----|-----|-----|
## Column Total |     106 |      27 |     133 |
## |      0.797 |      0.203 |      |
```

It may be worth skipping forward to Chap. 14, where we present a summary table for the key measures used to evaluate the performance of binary tests, classifiers, or predictions.

The prediction accuracy:

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} = \frac{93}{133} = 0.7.$$

From the cross table we can see that our prediction accuracy is $\frac{93}{133} = 0.70$. However, the *later stage* classification only has three counts. This might be due to the $P(F_l | C_L) \sim 0$ problem that we discussed above.

8.5.5 Step 5: Improving Model Performance

After setting `laplace=15`, the accuracy goes up to 76%. Although this is a small improvement in terms of accuracy, we have a better chance of detecting *later stage* patients.

```
hn_classifier <- naiveBayes(hn_train, hn_med_train$stage, Laplace = 15)
hn_test_pred<-predict(hn_classifier, hn_test)
CrossTable(hn_test_pred, hn_med_test$stage)

##   Cell Contents
## |-----|
## |           N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## Total Observations in Table: 133
##
##          | hn_med_test$stage
## hn_test_pred | early_stage | later_stage |   Row Total |
## -----|-----|-----|-----|
## early_stage |      99 |      25 |      124 |
## |      0.000 |      0.001 |      0.932 |
## |      0.798 |      0.202 |      0.926 |
## |      0.934 |      0.074 |      0.188 |
## |      0.744 |      0.222 |      0.068 |
## |-----|-----|-----|
## later_stage |       7 |       2 |       9 |
## |      0.004 |      0.016 |      0.053 |
## |      0.778 |      0.222 |      0.066 |
## |      0.053 |      0.015 |      0.203 |
## |-----|-----|-----|
## Column Total |     106 |      27 |     133 |
## |      0.797 |      0.203 |      0.926 |
## |-----|-----|-----|
```

8.5.6 Step 6: Compare Naive Bayesian against LDA

As mentioned earlier, Naive Bayes with normality assumption is a special case of Discriminant Analysis. It might be interesting to compare the results against LDA.

```
library(MASS)
df_hn_train = data.frame(lapply(as.data.frame(hn_train), as.numeric), stage = hn_med_train$stage)
df_hn_test = data.frame(lapply(as.data.frame(hn_test), as.numeric), stage = hn_med_test$stage)

hn_Lda <- lda(data=df_hn_train, stage~.)

# hn_pred = predict(hn_Lda, df_hn_test[, -104])
hn_pred = predict(hn_Lda, df_hn_test)
CrossTable(hn_pred$class, df_hn_test$stage)

## Cell Contents
## |-----|
## | N |
## | Chi-square contribution |
## | N / Row Total |
## | N / Col Total |
## | N / Table Total |
## |-----|
## Total Observations in Table: 133
##
## | df_hn_test$stage
## hn_pred$class | early_stage | Later_stage | Row Total |
## -----|-----|-----|-----|
## early_stage | 66 | 22 | 88 |
## | 0.244 | 0.957 | |
## | 0.750 | 0.250 | 0.662 |
## | 0.623 | 0.815 | |
## | 0.496 | 0.165 | |
## -----|-----|-----|-----|
## Later_stage | 40 | 5 | 45 |
## | 0.477 | 1.872 | |
## | 0.889 | 0.111 | 0.338 |
## | 0.377 | 0.185 | |
## | 0.301 | 0.038 | |
## -----|-----|-----|-----|
## Column Total | 106 | 27 | 133 |
## | 0.797 | 0.203 | |
## -----|-----|-----|-----|
```

Here, Naive Bayes outperforms LDA classifier in terms of the overall accuracy. However, LDA has a lower type II error ($\frac{22}{133}$), which is clinically important in order to avoid misdiagnosing later-stage cancer patients as early stage.

In later chapters, we will step deeper into the space of classification problems and see more sophisticated approaches.

8.6 Practice Problem

In the previous case study, we classified the patients with `seer_stage` of “not applicable”(`seer_stage=8`) and “unstaged, unknown or unspecified”(`seer_stage=9`) as no cancer or early cancer stages. Let’s remove these two categories and replicate the Naive Bayes classifier case study again.

```
hn_med1<-hn_med[!hn_med$seer_stage %in% c(8, 9), ]
str(hn_med1); dim(hn_med1)

## 'data.frame': 580 obs. of 9 variables:
## $ PID           : int 10000 10008 10029 10063 10103 1012 10135 10143
10152 10184 ...
## $ ENC_ID        : int 46836 46886 47034 47240 47511 3138 47739 47769
47800 47938 ...
## $ seer_stage    : Factor w/ 9 Levels "0","1","2","3",...: 2 2 5 2 2 2
2 2 7 2 ...
## $ MEDICATION_DESC : chr "ranitidine" "heparin injection" "ampicillin/
sulbactam IVPB UH" "fentaNYL injection UH" ...
## $ MEDICATION_SUMMARY: chr "(Zantac) 150 mg tablet oral two times a day"
"5,000 unit subcutaneous three times a day" "(Unasyn) 15 g IV every 6 hours"
"25 - 50 microgram IV every 5 minutes PRN severe pain\nMaximum dose 200 mcg
Per PACU protocol" ...
## $ DOSE          : chr "150" "5000" "1.5" "50" ...
## $ UNIT          : chr "mg" "unit" "g" "microgram" ...
## $ FREQUENCY     : chr "two times a day" "three times a day" "every
6 hours" "every 5 minutes" ...
## $ TOTAL_DOSE_COUNT : int 5 3 11 2 2 2 6 1 24 2 ...

## [1] 580 9
```

Now we have only 580 observations. We can either use the first 480 of them as the training dataset and the last 100 as the test dataset, or select 80–20 (training–testing) split, and evaluate the prediction accuracy when `laplace=1`?

We can use the same code for creating the classes in training and test dataset. Since the `seer_stage=8` or `9` is not in the data, we classify `seer_stage=0`, `1`, `2` or `3` as “`early_stage`” and `seer_stage=4`, `5` or `7` as “`later_stage`”.

```
hn_med_train1$stage<-hn_med_train1$seer_stage %in% c(4, 5, 7)
hn_med_train1$stage<-factor(hn_med_train1$stage, levels=c(F, T), labels = c(
  "early_stage", "later_stage"))
hn_med_test1$stage<-hn_med_test1$seer_stage %in% c(4, 5, 7)
hn_med_test1$stage<-factor(hn_med_test1$stage, levels=c(F, T), labels = c("e
arly_stage", "later_stage"))
prop.table(table(hn_med_train1$stage))

##
## early_stage later_stage
## 0.7392241 0.2607759

prop.table(table(hn_med_test1$stage))

##
## early_stage later_stage
## 0.7413793 0.2586207
```

Use terms that have appeared in five or more documents in the training dataset to build the document term matrix.

```
##      Length      Class      Mode
##      112 character character

##      Cell Contents
## |-----|
## |           N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
## 
## 
## Total Observations in Table: 116
## 
## 
##      | hn_med_test1$stage
## hn_test_pred1 | early_stage | Later_stage |   Row Total |
## -----|-----|-----|-----|
##   early_stage |      84 |      28 |      112 |
##   |      0.011 |      0.032 |      |
##   |      0.750 |      0.250 |      0.966 |
##   |      0.977 |      0.933 |      |
##   |      0.724 |      0.241 |      |
## -----|-----|-----|-----|
##   Later_stage |      2 |      2 |      4 |
##   |      0.314 |      0.901 |      |
##   |      0.500 |      0.500 |      0.034 |
##   |      0.023 |      0.067 |      |
##   |      0.017 |      0.017 |      |
## -----|-----|-----|-----|
##   Column Total |      86 |      30 |      116 |
##   |      0.741 |      0.259 |      |
## -----|-----|-----|-----|
```

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} = \frac{86}{116} = 0.74.$$

Try to reproduce these results with some new data from the list of our Case-Studies.

8.7 Assignments 8: Probabilistic Learning: Classification Using Naive Bayes

8.7.1 Explain These Two Concepts

- Bayes Theorem
- Laplace Estimation

8.7.2 *Analyzing Textual Data*

Load the SOCR 2011 US Job Satisfaction data. The last column (`Description`) contains free text about each job. Notice that spaces are replaced by underscores, `__`. Mine the text field and suggest some the meta-data analytics.

- Convert the textual meta-data into a corpus object.
- Triage some of the irrelevant punctuation and other symbols in the corpus document, change all text to lower case, etc.
- Tokenize the job descriptions into words. Examine the distributions of `Stress_Category` and `Hiring_Potential`.
- Classify the job stress into two categories.
- Generate a word cloud to visualize the job description text.
- Graphically visualize the difference between low and high `Stress_Category` graph.
- Transform the word count features into categorical data
- Ignore those low frequency words and report the sparsity of your categorical data matrix with or without delete those low frequency words.
- Apply the Naive Bayes classifier to original matrix and lower dimension matrix. What do you observe?
- Apply and compare LDA and Naive Bayes classifiers with respect to the error, specificity and sensitivity.

References

- Kidwell , David A. (2013) *Lazy Learning*, Springer Science & Business Media, ISBN 9401720533, 9789401720533
- Aggarwal, Charu C. (ed.) (2015) *Data Classification: Algorithms and Applications*, Chapman & Hall/CRC, ISBN 1498760589, 9781498760584

Chapter 9

Decision Tree Divide and Conquer Classification



When classification needs to be apparent, kNN or naive Bayes we presented earlier may not be useful as they do not generate explicit classification rules. In some cases, we need to specify well stated rules for our decisions, just like a scoring criterion for driving ability or credit scoring for loan underwriting. The decisions in many situations actually require having a clear and easily understandable decision tree to follow the classification process start to finish.

In this chapter, we will (1) see a simple motivational example of decision trees based on the Iris data; (2) describe decision-tree divide and conquer methods; (3) examine certain measures quantifying classification accuracy; (4) show strategies for pruning decision trees; (5) work through a Quality of Life in Chronic Disease case-study; and (6) review the *One Rule* and *RIPPER* algorithms.

9.1 Motivation

Decision tree learners enable classification via tree structures modeling the relationships among all features and potential outcomes in the data. All decision trees begin with a trunk (all data are part of the same cohort), which is then split into narrower and narrower branches by forking decisions based on the intrinsic data structure. At each step, splitting the data into branches may include binary or multinomial classification. The final decision is obtained when the tree branching process terminates. The terminal (leaf) nodes represent the action to be taken as the result of the series of branching decisions. For predictive models, the leaf nodes provide the expected forecasting results given the series of events in the tree.

There are a number of R packages available for decision tree classification including `rpart`, `C5.0`, `party`, etc.

9.2 Hands-on Example: Iris Data

Let's start by seeing a simple example using the Iris dataset, which we saw in Chap. 3. The data features or attributes include `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`, and classes are represented by the `Species` taxa (setosa; versicolor; and virginica).

```
## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 4.6 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species     : Factor w/ 3 Levels "setosa","versicolor",...: 1 1 1 1 1 1
1 1 1 1 ...

##   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

##
##   setosa versicolor virginica
##      50      50      50
```

The `ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris)` function will build a decision tree (Figs. 9.1 and 9.2).

```
iris_ctree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris)
print(iris_ctree)

## Conditional inference tree with 4 terminal nodes
##
## Response: Species
## Inputs: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
## Number of observations: 150
##
## 1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
## 2)* weights = 50
## 1) Petal.Length > 1.9
##   3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894
##     4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865
##       5)* weights = 46
##       4) Petal.Length > 4.8
##         6)* weights = 8
##         3) Petal.Width > 1.7
##           7)* weights = 46

plot(iris_ctree, cex=2)
```

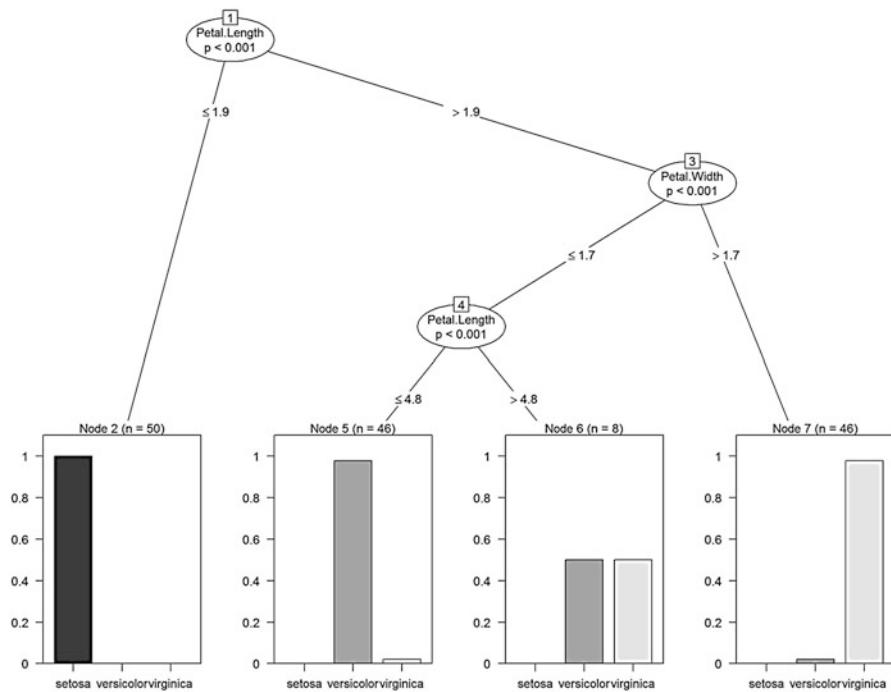


Fig. 9.1 Decision tree classification illustrating four leaf node labels corresponding to the three iris genera

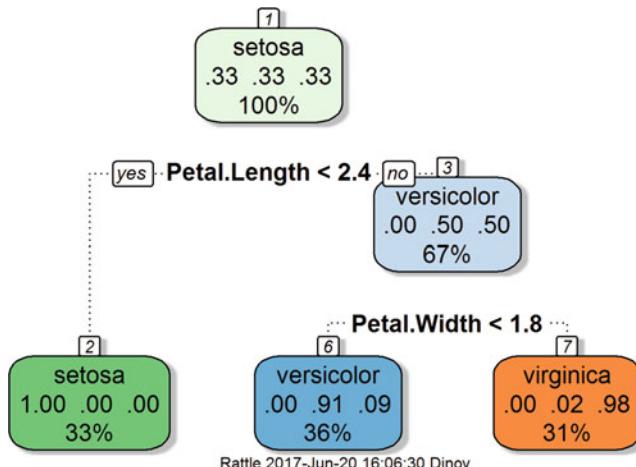


Fig. 9.2 An alternative decision tree classification of the iris flowers dataset

```
head(iris); tail(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

##   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
```

Similarly, we can demonstrate a classification of the *iris taxa* via `rpart`:

```
library(rpart)
iris_rpart = rpart(Species~., data=iris)
print(iris_rpart)

## n= 150
##
## node), split, n, Loss, yval, (yprob)
##           * denotes terminal node
##
## 1) root 150 100 setosa (0.3333333 0.3333333 0.3333333)
##    2) Petal.Length< 2.45 50  0 setosa (1.000 0.0000000 0.0000000) *
##    3) Petal.Length>=2.45 100  50 versicolor (0.000 0.5000000 0.5000000)
##      6) Petal.Width< 1.75 54  5 versicolor (0.000 0.90740741 0.09259259) *
##      7) Petal.Width>=1.75 46  1 virginica (0.000 0.02173913 0.97826087) *

# Use the `rattle::fancyRpartPlot` to generates an elegant plot
library(rattle)

## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

fancyRpartPlot(iris_rpart, cex = 1.5)
```

9.3 Decision Tree Overview

The decision tree algorithm represents an upside down tree with lots of tree branch bifurcations where a series of logical decisions are encoded as tree node splits. The classification begins at the root node and goes through many branches until it gets to the terminal nodes. This iterative process splits the data into different classes by rigid criteria.

9.3.1 Divide and Conquer

Decision trees involve recursive partitioning that uses data features and attributes to split the data into groups (nodes) of similar classes.

To make classification trees using data features, we need to observe the pattern between the data features and potential classes using training data. We can draw scatter plots and separate groups that are clearly clotted together. Each group is considered a segment of the data. After getting the approximate range of each feature value under each group, we can make the decision tree.

$$X = [X_1, X_2, X_3, \dots, X_k] = \underbrace{\begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,k} \\ x_{2,1} & x_{2,2} & \dots & x_{2,k} \\ \dots & \dots & \dots & \dots \\ x_{n,1} & x_{n,2} & \dots & x_{n,k} \end{pmatrix}}_{\text{features/attributes}} \left\{ \text{cases} \right\}$$

The decision tree algorithms use a top-down recursive divide-and-conquer approach (sometimes they may also use bottom up or mixed splitting strategies) to divide and evaluate the splits of a dataset D (input). The best split decision corresponds to the split with the **highest information gain**, reflecting a partition of the data into K subsets (using divide-and-conquer). The iterative algorithm terminates when some stopping criteria are reached. Examples of stopping conditions used to terminate the recursive process include:

- All the samples belong to the same class, that is they have the same label and the sample is already **pure**.
- Stop when majority of the points are already of the same class (relative to some error threshold).
- There are no remaining attributes on which the samples may be further partitioned.

One objective criteria for splitting or clustering data into groups is based on the **information gain measure**, or **impurity reduction**, which can be used to select the test attribute at each node in the decision tree. The attribute with the highest information gain (i.e., greatest entropy reduction) is selected as the test attribute for the current node. This attribute minimizes the information needed to classify the samples in the resulting partitions. There are three main indices to evaluate the impurity reduction: **Misclassification error**, **Gini index** and **Entropy**.

For a given table containing pairs of attributes and their class labels, we can assess the homology of the classes in the table. A table is pure (homogenous) if it only contains a single class. If a data table contains several classes, then we say that the table is impure or heterogeneous. This degree of impurity or heterogeneity can be quantitatively evaluated using impurity measures like entropy, Gini index, and misclassification error.

9.3.2 Entropy

The **Entropy** is an information measure of the amount of disorder or uncertainty in a system. Suppose we have a data set $D = (X_1, X_2, \dots, X_n)$ that includes n features (variables) and suppose each of these features can take on any of k possible values (states). Then the cardinality of the entire system is k^n as each of the features are assumed to have k independent states, thus the total number of different datasets that can be expected is $\underbrace{k \times k \times \dots \times k}_n = k^n$. Suppose p_1, p_2, \dots, p_n represent the

proportions of each class (note: $\sum_i p_i = 1$) present in the child node that results from a split in a decision tree classifier. Then the entropy measure is defined by:

$$\text{Entropy}(D) = -\sum_i p_i \log_2 p_i.$$

If each of the $1 \leq i \leq k$ states for each feature is equally likely to be observed with probability $p_i = \frac{1}{k}$, then the entropy is maximized:

$$\text{Entropy}(D) = -\sum_{i=1}^k \frac{1}{k} \log \frac{1}{k} = \sum_{i=1}^k \frac{1}{k} \log k = \frac{1}{k} \sum_{i=1}^k 1 = 1.$$

In the other extreme, the entropy is minimized. Note that by L'Hopital's Rule $\lim_{x \rightarrow 0} x \times \log(x) = \lim_{x \rightarrow 0} \frac{\frac{1}{x}}{\frac{1}{x^2}} = \lim_{x \rightarrow 0} x = 0$ for a single class classification where the probability of one class is unitary ($p_{i_o} = 1$) and the other ones are trivial ($p_{i \neq i_o} = 0$):

$$\begin{aligned} \text{Entropy}(D) &= -\sum_i \frac{1}{k} \log \left(\frac{1}{k} \right) = p_{i_o} \times \log(p_{i_o}) + \sum_{i \neq i_o} p_i \log(p_i) = \\ &= 1 \times \log(1) + \lim_{x \rightarrow 0} \sum_{i \neq i_o} x \log(x) = 0 + 0 = 0. \end{aligned}$$

In classification settings, higher entropy (i.e., more disorder) corresponds to a sample that has a **mixed collection of labels**. Conversely, lower entropy corresponds to a classification where we have mostly pure partitions. In general, the entropy of a sample $D = \{x_1, x_2, \dots, x_n\}$ is defined by:

$$H(D) = -\sum_{i=1}^k P(c_i|D) \log P(c_i|D),$$

where $P(c_i|D)$ is the probability of a data point in D being labeled with class c_i , and k is the number of classes (clusters). $P(c_i|D)$ can be estimated from the observed data by:

$$P(c_i|D) = \frac{|\{x_j \in D | x_j \text{ has label } y_j = c_i\}|}{|D|}.$$

Observe that if the observations are evenly split amongst all k classes, then $P(c_i|D) = \frac{1}{k}$ and

$$H(D) = - \sum_{i=1}^k \frac{1}{k} \log \frac{1}{k} = 1.$$

At the other extreme, if all the observations are from one class then:

$$H(D) = -1 * \log_k(1) = 0.$$

Also note that the base of the log function is somewhat irrelevant and can be used to normalize (scale) the range of the entropy $\left(\log_b(x) = \frac{\log_2(x)}{\log_2(b)} \right)$.

The **Gain** is the expected reduction in entropy caused by knowing the value of an attribute.

9.3.3 Misclassification Error and Gini Index

Similar to the Entropy measure, the Misclassification error and the Gini index are also applied to evaluate information gain. The Misclassification error is defined by the formula:

$$ME = 1 - \max_k (p_k).$$

And the Gini index is expressed as:

$$GI = \sum_{k=1}^k p_k(1 - p_k) = 1 - \sum_{k=1}^k p_k^2.$$

9.3.4 C5.0 Decision Tree Algorithm

C5.0 algorithm is a popular implementation of decision trees.

To begin with, let's consider the term **purity**. If the segments of data contains a single class, they are considered **pure**. The entropy represents a mathematical formalism measuring purity of data segments.

$$\text{Entropy}(S) = - \sum_{i=1}^c p_i \log_2(p_i),$$

where `entropy` is the measurement, c is the number of total class levels, and p_i refers to the proportion of observations that fall into each class (i.e., probability of a randomly selected data point to belong to the i th class level). For two possible classes, the `entropy` ranges from 0 to 1. For n classes, the `entropy` ranges from 0 to $\log_2(n)$, where the minimum `entropy` corresponds to data that is purely homogeneous (completely deterministic/predictable) and the maximum `entropy` represents completely disordered data (stochastic or extremely noisy). You might wonder what is the benefit of using the `entropy`? Another way to say this is the smaller the `entropy`, the more information is contained in this split method. Systems (data) with high `entropy` indicate significant information content (randomness) and data with low `entropy` indicates highly-compressible data with structure in it.

If we only have one class in the segment, then $\text{Entropy}(S) = (-1) \times \log_2(1) = 0$.

Let's try another example. If we have a segment of data that contains two classes, the first class contains 80% of the data and the second class contains the remaining 20%. Then, we have the following `entropy`:

$$\text{Entropy}(S) = -0.8 \log_2(0.8) - 0.2 \log_2(0.2) = 0.7219281.$$

The relationship for two class proportions and `entropy` are illustrated in Fig. 9.3, where x is the proportion for elements in one of the classes.

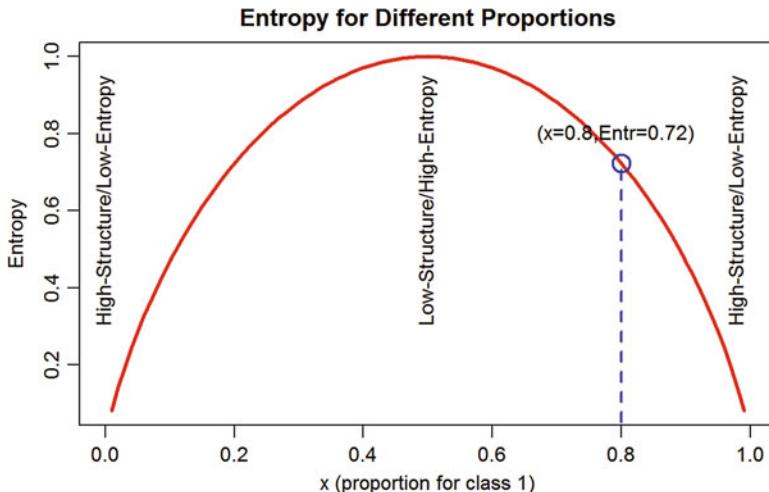


Fig. 9.3 Plot of the `entropy` of a (symmetric) binary process as a function of the proportion of class 1 cases

```
set.seed(1234)
x<-runif(100)
curve(-x*log2(x)-(1-x)*log2(1-x), col="red", main="Entropy for Different
Proportions", xlab = "x (proportion for class 1)", ylab = "Entropy", lwd=3)
```

The closer the binary proportion split is to 0.5, the greater the entropy. The more homogeneous the split (one class becomes the majority) the lower the entropy. **Decision trees** aim to find splits in the data that reduce the entropy, i.e., increasing the homogeneity of the elements within all classes.

This measuring mechanism could be used to measure and compare the information we get using different features as data partitioning characteristics. Let's consider this scenario. Suppose S and S_1 represent the entropy of the system before and after the splitting/partitioning of the data according to a specific data feature attribute (F). Denote the entropies of the original and the derived partition by $Entropy(S)$ and $Entropy(S_1)$, respectively. The **information we gained** from partitioning the data using this specific feature (F) is calculated as a change in the entropy:

$$Gain(F) = Entropy(S) - Entropy(S_1).$$

Note that smaller entropy $Entropy(S_1)$ corresponds with better classification and more information gained. A more complicated case would be that the partitions create multiple segments. Then, the entropy for each partition method is calculated by the following formula:

$$Entropy(S) = \sum_{i=1}^n w_i Entropy(P_i) = \sum_{i=1}^n w_i \left(\sum_{j=1}^c -p_i \log_2(p_i) \right),$$

where w_i is the proportion of examples falling in that segment i . Thus, the total entropy of a partition method is calculated by a weighted sum of entropies for each segment created by this method.

When we get the maximum reduction in entropy with a feature (F), then the $Gain(F) = Entropy(S)$, since $Entropy(S_1) = 0$. On the contrary, if we gain no information with this feature, we have $Gain(F) = 0$.

9.3.5 Pruning the Decision Tree

While making a decision tree, we can classify those observations using as many splits as we want. This eventually might over classify our data. An extreme example of this would be that we make each observation as a class, which is meaningless.

So how do we control the size of the decision tree? One possible solution is to make a cutoff for the number of decisions that a decision tree could make. Similarly, we can control the number of examples in each segment to be not too small. This method is called *early stopping* or *pre-pruning* the decision tree. However, this might make the decision procedure stop prematurely, before some important partition occurs.

Another solution *post-pruning* is that we begin with growing a big decision tree and subsequently reduce the branches based on error rates with penalty at the nodes. This is often more effective than the pre-pruning solution.

The C5.0 algorithm uses the *post-pruning* method to control the size of the decision tree. It first grows an overfitting large tree to contain all the possibilities of partitioning. Then, it cuts out nodes and branches with little effect on classification errors.

9.4 Case Study 1: Quality of Life and Chronic Disease

9.4.1 Step 1: Collecting Data

In this Chapter, we are using the Quality of life and chronic disease dataset, `Case06_QoL_Symptom_ChronicIllness.csv`. This dataset has 41 variables. Detailed description for each variable is provided here (https://umich.instructure.com/files/399150/download?download_frd=1).

Important variables:

- **Charlson Comorbidity Index:** ranging from 0 to 10. A score of 0 indicates no comorbid conditions. Higher scores indicate a greater level of comorbidity.
- **Chronic Disease Score:** A summary score based on the presence and complexity of prescription medications for select chronic conditions. A high score in decades the patient has severe chronic diseases. Entries stored as -9 indicate missing value.

9.4.2 Step 2: Exploring and Preparing the Data

Let's load the data first.

```
qol<-read.csv("https://umich.instructure.com/files/481332/download?download_frd=1")
str(qol)

## 'data.frame': 2356 obs. of 41 variables:
## $ ID : int 171 171 172 179 180 180 180 181 182 183 ...
## $ INTERVIEWDATE : int 0 427 0 0 0 42 0 0 0 0 ...
## $ LANGUAGE : int 1 1 1 1 1 1 1 1 1 2 ...
## $ AGE : int 49 49 62 44 64 64 52 48 49 78 ...
## $ RACE_ETHNICITY : int 3 3 3 7 3 3 3 3 3 4 ...
## $ SEX : int 2 2 2 2 1 1 2 1 1 1 ...
## $ QOL_Q_01 : int 4 4 3 6 3 3 4 2 3 5 ...
## $ QOL_Q_02 : int 4 3 3 6 2 5 4 1 4 6 ...
## $ QOL_Q_03 : int 4 4 4 6 3 6 4 3 4 4 ...
## $ QOL_Q_04 : int 4 4 2 6 3 6 2 2 5 2 ...
## $ QOL_Q_05 : int 1 5 4 6 2 6 4 3 4 3 ...
## $ QOL_Q_06 : int 4 4 2 6 1 2 4 1 2 4 ...
```

```

## $ QOL_Q_07 : int 1 2 5 -1 0 5 8 4 3 7 ...
## $ QOL_Q_08 : int 6 1 3 6 6 6 3 1 2 4 ...
## $ QOL_Q_09 : int 3 4 3 6 2 2 4 2 2 4 ...
## $ QOL_Q_10 : int 3 1 3 6 3 6 3 2 4 3 ...
## $ MSA_Q_01 : int 1 3 2 6 2 3 4 1 1 2 ...
## $ MSA_Q_02 : int 1 1 2 6 1 6 4 3 2 4 ...
## $ MSA_Q_03 : int 2 1 2 6 1 2 3 3 1 2 ...
## $ MSA_Q_04 : int 1 3 2 6 1 2 1 4 1 5 ...
## $ MSA_Q_05 : int 1 1 1 6 1 2 1 6 3 2 ...
## $ MSA_Q_06 : int 1 2 2 6 1 2 1 1 2 2 ...
## $ MSA_Q_07 : int 2 1 3 6 1 1 1 1 1 5 ...
## $ MSA_Q_08 : int 1 1 1 6 1 1 1 2 1 ...
## $ MSA_Q_09 : int 1 1 1 6 2 2 4 6 2 1 ...
## $ MSA_Q_10 : int 1 1 1 6 1 1 1 1 1 3 ...
## $ MSA_Q_11 : int 2 3 2 6 1 1 2 1 1 5 ...
## $ MSA_Q_12 : int 1 1 2 6 1 1 2 6 1 3 ...
## $ MSA_Q_13 : int 1 1 1 6 1 6 2 1 4 2 ...
## $ MSA_Q_14 : int 1 1 1 6 1 2 1 1 3 1 ...
## $ MSA_Q_15 : int 2 1 1 6 1 1 3 2 1 3 ...
## $ MSA_Q_16 : int 2 3 5 6 1 2 1 2 1 2 ...
## $ MSA_Q_17 : int 2 1 1 6 1 1 1 1 1 3 ...
## $ PH2_Q_01 : int 3 2 1 5 1 1 3 1 2 3 ...
## $ PH2_Q_02 : int 4 4 1 5 1 2 1 1 4 2 ...
## $ TOS_Q_01 : int 2 2 2 4 1 1 2 2 1 1 ...
## $ TOS_Q_02 : int 1 1 1 4 4 4 1 2 4 4 ...
## $ TOS_Q_03 : int 4 4 4 4 4 4 4 4 4 4 ...
## $ TOS_Q_04 : int 5 5 5 5 5 5 5 5 5 ...
## $ CHARLSONSCORE : int 2 2 3 1 0 0 2 8 0 1 ...
## $ CHRONICDISEASESCORE: num 1.6 1.6 1.54 2.97 1.28 1.28 1.31 1.67 2.21 2
.51 ...

```

Most of the coded variables like QOL_Q_01 (heath rating) have ordinal values (1 = excellent, 2 = very good, 3 = good, 4 = fair, 5 = poor, 6 = no answer). We can use the `table()` function to see their distributions. We also have some numerical variables in the dataset like CHRONICDISEASESCORE. We can take a look at it by using `summary()`.

Our variable of interest CHRONICDISEASESCORE has some missing data. A simple way to address this is just deleting those observations with missing values. You could also try to impute the missing value using various imputation methods mentioned in Chap. 3.

```



```

Let's create two classes using variable CHRONICDISEASESCORE. We classify the patients with CHRONICDISEASESCORE < mean(CHRONICDISEASESCORE) as having minor disease and the rest as having severe disease. This dichotomous classification (qol\$cd) may not be perfect and we will talk about alternative classification strategies in the practice problem in the end of the chapter.

```
qol$cd<-qol$CHRONICDISEASESCORE>1.497
qol$cd<-factor(qol$cd, levels=c(F, T), labels = c("minor_disease", "severe_disease"))
```

Data Preparation: Creating Random Training and Test Datasets

To make the qol data more organized, we can order the data by the variable ID.

```
qol<-qol[order(qol$ID), ]
# Remove ID (col=1) # the clinical Diagnosis (col=41) will be handled later
qol <- qol[ , -1]
```

Then, we are able to subset *training* and *testing* datasets. Here is an example of a *non-random split* of the entire data into training (2114) and testing (100) sets:

```
qol_train<-qol[1:2114, ]
qol_test<-qol[2115:2214, ]
```

And here is an example of *random assignments* of cases into training and testing sets (80–20% split):

```
set.seed(1234)
train_index <- sample(seq_len(nrow(qol)), size = 0.8*nrow(qol))
qol_train<-qol[train_index, ]
qol_test<-qol[-train_index, ]
```

We can quickly inspect the distributions of the training and testing data to ensure they are not vastly different. We can see that the classes are split fairly equal in training and testing datasets.

```
prop.table(table(qol_train$cd))
## minor_disease severe_disease
##      0.5279503      0.4720497

prop.table(table(qol_test$cd))
## minor_disease severe_disease
##      0.503386      0.496614
```

9.4.3 Step 3: Training a Model On the Data

In this section, we are using the `C5.0()` function from the `C50` package.

The function `C5.0()` has following components:

```
m<-C5.0(train, class, trials=1, costs=NULL)
```

- `train`: data frame containing numeric training data (features).
- `class`: factor vector with the class for each row in the training data.
- `trials`: an optional number to control the boosting iterations (default = 1).
- `costs`: an optional matrix to specify the costs of false positive and false negative.

You could delete the `#` in the following code and run it in R to install and load the `C50` package.

```
# install.packages("C50")
library(C50)
```

In the `qol` dataset (ID column is already removed), column 41 is the class vector (`qol$cd`), and column 40 is the numerical version of vector 41 (`qol$CHRONICDISEASESCORE`). We need to delete these two columns to create our training data that only contains features.

```
summary(qol_train[,-c(40, 41)])
## #> INTERVIEWDATE      LANGUAGE      AGE      RACE_ETHNICITY
## #> Min.   : 0.00  Min.   :1.000  Min.   :20.00  Min.   :1.000
## #> 1st Qu.: 0.00  1st Qu.:1.000  1st Qu.:52.00  1st Qu.:3.000
## #> Median : 0.00  Median :1.000  Median :59.00  Median :3.000
## #> Mean   :21.68  Mean   :1.217  Mean   :58.74  Mean   :3.614
## #> 3rd Qu.: 0.00  3rd Qu.:1.000  3rd Qu.:67.00  3rd Qu.:4.000
## #> Max.   :440.00  Max.   :2.000  Max.   :90.00  Max.   :7.000
## #> SEX      QOL_Q_01      QOL_Q_02      QOL_Q_03
## #> Min.   :1.000  Min.   :1.000  Min.   :1.000  Min.   :1.000
## #> 1st Qu.:1.000  1st Qu.:3.000  1st Qu.:3.000  1st Qu.:3.000
## #> Median :1.000  Median :4.000  Median :3.000  Median :4.000
## #> Mean   :1.422  Mean   :3.661  Mean   :3.408  Mean   :3.714
## #> 3rd Qu.:2.000  3rd Qu.:4.000  3rd Qu.:4.000  3rd Qu.:4.000
## #> Max.   :2.000  Max.   :6.000  Max.   :6.000  Max.   :6.000
...
## #> TOS_Q_03      TOS_Q_04      CHARLSONSCORE
## #> Min.   :1.000  Min.   :1.000  Min.   :-9.0000
## #> 1st Qu.:4.000  1st Qu.:5.000  1st Qu.: 0.0000
## #> Median :4.000  Median :5.000  Median : 1.0000
## #> Mean   :3.787  Mean   :4.686  Mean   : 0.8826
## #> 3rd Qu.:4.000  3rd Qu.:5.000  3rd Qu.: 1.0000
## #> Max.   :5.000  Max.   :6.000  Max.   :10.0000
set.seed(1234)
qol_model<-C5.0(qol_train[,-c(40, 41)], qol_train$cd)
qol_model
```

```

## 
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd)
## 
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
## 
## Tree size: 25
## 
## Non-standard options: attempt to group attributes

summary(qol_model)

## 
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd)
## 
## 
## C5.0 [Release 2.07 GPL Edition]      Tue Jun 20 16:09:16 2017
## -----
## 
## Class specified by attribute `outcome'
## 
## Read 1771 cases (40 attributes) from undefined.data
## 
## Decision tree:
## 
## CHARLSONSCORE <= 0: minor_disease (665/180)
## CHARLSONSCORE > 0:

## ....AGE <= 47:
##     ....MSA_Q_08 > 2: severe_disease (15/4)
##     :   MSA_Q_08 <= 2:
##     :     ....MSA_Q_14 <= 1: minor_disease (86/20)
##     :       MSA_Q_14 > 1:
##     :         ....MSA_Q_10 > 4: minor_disease (6)
##     :           MSA_Q_10 <= 4:
##     :             ....TOS_Q_03 > 4: severe_disease (8)
##     :               TOS_Q_03 <= 4:
##     :                 ....MSA_Q_17 > 2: minor_disease (8/1)
##     :                   MSA_Q_17 <= 2:
##     :                     ....QOL_Q_01 <= 2: minor_disease (4)
##     :                       QOL_Q_01 > 2: severe_disease (38/13)
##     AGE > 47:
##     ....RACE_ETHNICITY > 3:
##         ....QOL_Q_07 > 5: severe_disease (133/26)
##         :   QOL_Q_07 <= 5:
##         :           ....QOL_Q_10 > 5: severe_disease (24/2)
##         :             QOL_Q_10 <= 5:
##         :               ....MSA_Q_14 <= 5: severe_disease (202/72)
##         :                 MSA_Q_14 > 5: minor_disease (11/2)
##         RACE_ETHNICITY <= 3:
##             ....QOL_Q_01 <= 2: minor_disease (50/20)
##             :   QOL_Q_01 > 2:
##                 ....CHARLSONSCORE > 1: severe_disease (184/58)
##                 :   CHARLSONSCORE <= 1:
##                     ....MSA_Q_04 > 5: minor_disease (27/8)

```

```

##           MSA_Q_04 <= 5:
##           ....QOL_Q_07 <= 5:
##           ....QOL_Q_05 <= 2:
##           : ....TOS_Q_04 <= 2: minor_disease (5)
##           : : TOS_Q_04 > 2: severe_disease (52/15)
##           : QOL_Q_05 > 2:
##           : ....MSA_Q_06 <= 5: minor_disease (119/46)
##           : : MSA_Q_06 > 5: severe_disease (10/2)
##           QOL_Q_07 > 5:
##           ....QOL_Q_09 <= 2: severe_disease (18/1)
##           QOL_Q_09 > 2:
##           ....RACE_ETHNICITY <= 2: minor_disease (12/5)
##           RACE_ETHNICITY > 2:
##           ....MSA_Q_17 > 3: severe_disease (19/2)
##           MSA_Q_17 <= 3:
##           ....PH2_Q_01 <= 3: severe_disease (50/14)
##           PH2_Q_01 > 3:
##           ....MSA_Q_14 <= 3: minor_disease (21/6)
##           MSA_Q_14 > 3: severe_disease (4)
##
## Evaluation on training data (1771 cases):
##
##      Decision Tree
## -----
##      Size      Errors
##      25  497(28.1%)  <<
##
##      (a)   (b)   <-classified as
## -----
##      726   209   (a): class minor_disease
##      288   548   (b): class severe_disease

```

```

##
##
## Attribute usage:
##
## 100.00% CHARLSONSCORE
## 62.45% AGE
## 53.13% RACE_ETHNICITY
## 38.40% QOL_Q_07
## 34.61% QOL_Q_01
## 21.91% MSA_Q_14
## 19.03% MSA_Q_04
## 13.38% QOL_Q_10
## 10.50% QOL_Q_05
## 9.32% MSA_Q_08
## 8.13% MSA_Q_17
## 7.28% MSA_Q_06
## 7.00% QOL_Q_09
## 4.23% PH2_Q_01
## 3.61% MSA_Q_10
## 3.27% TOS_Q_03
## 3.22% TOS_Q_04

```

The output of `qol_model` indicates that we have a tree that has 25 terminal nodes. `summary(qol_model)` suggests that the classification error for decision tree is 28% in the training data.

9.4.4 Step 4: Evaluating Model Performance

Now we can make predictions using the decision tree that we just built. The `predict()` function we will use is the same as the one we showed in earlier chapters, e.g., Chaps. 3 and 8. In general, `predict()` is extended by each specific type of regression, classification, clustering, or forecasting machine learning technique. For example, `randomForest::predict.randomForest()` is invoked by:

```
predict(RF_model, newdata, type="response", norm.votes=TRUE,
predict.all=FALSE, proximity=FALSE, nodes=FALSE, cutoff, ...),
```

where `type` represents type of prediction output to be generated - "response" (equivalent to "class"), "prob" or "votes". Thus, the predicted values are either predicted "response" class labels, matrix of class probabilities, or vote counts.

This time we are going to introduce the `confusionMatrix()` function under package `caret` as the evaluation method. When we combine it with a `table()` function, the output of the evaluation is very straight forward.

```
qol_pred<-predict(qol_model, qol_test[, -c(40, 41)]) # removing the last 2
columns CHRONICDISEASESCORE and cd, which represent the clinical outcomes we
are predicting!
# install.packages("caret")
library(caret)

confusionMatrix(table(qol_pred, qol_test$cd))

## Confusion Matrix and Statistics
##
## qol_pred      minor_disease  severe_disease
##   minor_disease          149          89
##   severe_disease          74         131
##
##                               Accuracy : 0.6321
##                               95% CI : (0.5853, 0.6771)
##   No Information Rate : 0.5034
##   P-Value [Acc > NIR] : 3.317e-08
##
##                               Kappa : 0.2637
##   Mcnemar's Test P-Value : 0.2728
##
##                               Sensitivity : 0.6682
##                               Specificity : 0.5955
##   Pos Pred Value : 0.6261
##   Neg Pred Value : 0.6390
##   Prevalence : 0.5034
##   Detection Rate : 0.3363
##   Detection Prevalence : 0.5372
##   Balanced Accuracy : 0.6318
##
##   'Positive' Class : minor_disease
```

The Confusion Matrix shows that the testing data prediction accuracy is about 63%. However, this may vary (see the corresponding confidence interval).

9.4.5 Step 5: Trial Option

The `C5.0` function includes, an option `trials=`, which is an integer specifying the number of boosting iterations. The default value of one indicates that a single model is used, and we can specify a larger number of iterations, for instance `trials=6`.

```
set.seed(1234)
qol_boost6<-C5.0(qol_train[, -c(40, 41)], qol_train$cd, trials=6) # try alternative values for the trials option
qol_boost6

## 
## Call:
## C5.0.default(x = qol_train[, -c(40, 41)], y = qol_train$cd, trials = 6)
## 
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
## 
## Number of boosting iterations: 6
## Average tree size: 11.7
## 
## Non-standard options: attempt to group attributes
```

We can see that the size of the tree reduced to about 12 (this may vary at each run).

Since this is a fairly small tree, we can visualize it by the function `plot()`. We also use the option `type="simple"` to make the tree look more condensed (Fig. 9.4).

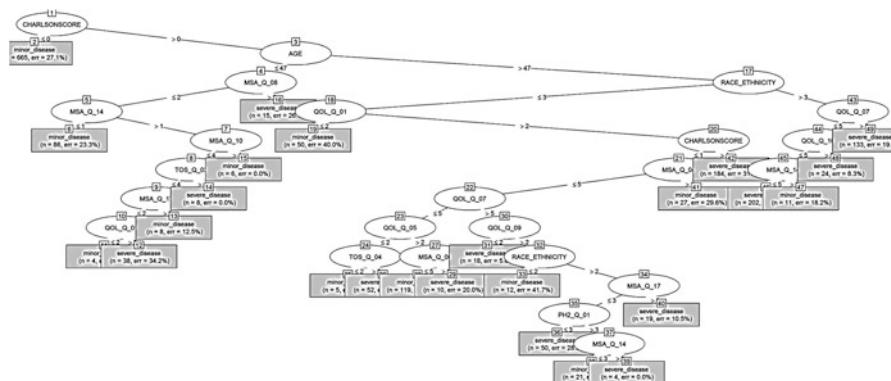


Fig. 9.4 Classification tree plot of the quality of life (QoL) data

```
plot(qol_boost6, type="simple")
```

Caution The plotting of decision trees will fail if you have columns that start with numbers or special characters (e.g., "5variable", "!variable"). In general, avoid spaces, special characters, and other non-terminal symbols in column/row names.

The next step would be making predictions and testing the corresponding accuracy.

```
qol_boost_pred6 <- predict(qol_boost6, qol_test[, -c(40, 41)])
confusionMatrix(table(qol_boost_pred6, qol_test$cd))

## Confusion Matrix and Statistics
## qol_boost_pred6  minor_disease  severe_disease
##   minor_disease           140            75
##   severe_disease           83            145
##
##                               Accuracy : 0.6433
##                               95% CI : (0.5968, 0.688)
##   No Information Rate : 0.5034
##   P-Value [Acc > NIR] : 1.987e-09
##   Kappa : 0.2868
##   Mcnemar's Test P-Value : 0.5776
##   Sensitivity : 0.6278
##   Specificity : 0.6591
##   Pos Pred Value : 0.6512
##   Neg Pred Value : 0.6360
##   Prevalence : 0.5034
##   Detection Rate : 0.3160
##   Detection Prevalence : 0.4853
##   Balanced Accuracy : 0.6434
##
##   'Positive' Class : minor_disease
```

The accuracy is about 64%. However, this may vary each time we run the experiment (mind the confidence interval). In some studies, the *trials* option provides significant improvement to the overall accuracy. A good choice for this option is *trials* = 10.

9.4.6 Loading the Misclassification Error Matrix

Suppose we want to reduce the *false negative rate*, in this case, misclassifying a severe case as minor. False negative (failure to detect a severe disease case) may be more costly than false positive (misclassifying a minor disease case as severe). Misclassification errors can be expressed as a matrix:

```
error_cost<-matrix(c(0, 1, 4, 0), nrow = 2)
error_cost
##      [,1] [,2]
## [1,]     0     4
## [2,]     1     0
```

Let's build a decision tree with the option `cpsts=error_cost`.

```
set.seed(1234)
qol_cost<-C5.0(qol_train[-c(40, 41)], qol_train$cd, costs=error_cost)
qol_cost_pred<-predict(qol_cost, qol_test)
confusionMatrix(table(qol_cost_pred, qol_test$cd))

## Confusion Matrix and Statistics
##
##
## qol_cost_pred    minor_disease  severe_disease
##   minor_disease           60           17
##   severe_disease          163          203
##
##                               Accuracy : 0.5937
##                               95% CI : (0.5463, 0.6398)
##   No Information Rate : 0.5034
##   P-Value [Acc > NIR] : 8.352e-05
##
##                               Kappa : 0.1909
##   Mcnemar's Test P-Value : < 2.2e-16
##
##                               Sensitivity : 0.2691
##                               Specificity : 0.9227
##   Pos Pred Value : 0.7792
##   Neg Pred Value : 0.5546
##   Prevalence : 0.5034
##   Detection Rate : 0.1354
##   Detection Prevalence : 0.1738
##   Balanced Accuracy : 0.5959
##
##   'Positive' Class : minor_disease
```

Although the overall accuracy decreased, the false negative cell labels were reduced from 75 (without specifying a cost matrix) to 17 (when specifying a non-trivial (loaded) cost matrix). This comes at the cost of increasing the rate of false-positive labeling (minor disease cases misclassified as severe).

9.4.7 Parameter Tuning

There are multiple choices to plot trees fitted by `rpart`, `C50`.

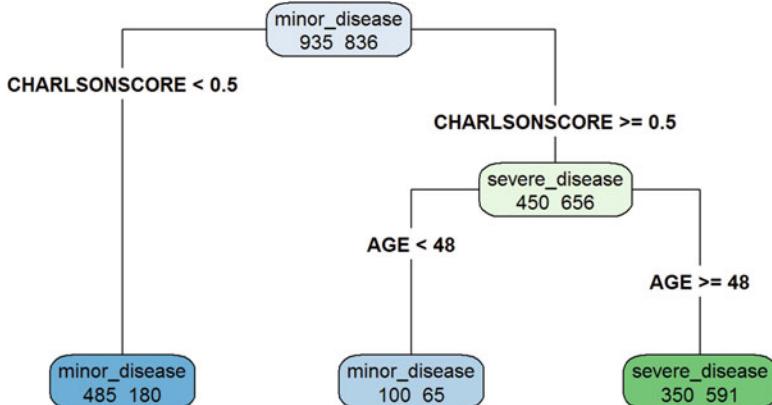


Fig. 9.5 Decision tree classification of the QoL data

```

library("rpart")
# remove CHRONICDISEASESCORE, but keep *cd* label
set.seed(1234)
qol_model<-rpart(cd~, data=qol_train[, -40], cp=0.01)
# here we use rpart::cp = *complexity parameter* = 0.01
qol_model

## n= 1771
##
## node), split, n, Loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 1771 836 minor_disease (0.5279503 0.4720497)
##    2) CHARLSONSCORE< 0.5 665 180 minor_disease (0.7293233 0.2706767) *
##    3) CHARLSONSCORE>=0.5 1106 450 severe_disease (0.4068716 0.5931284)
##      6) AGE< 47.5 165 65 minor_disease (0.6060606 0.3939394) *
##      7) AGE>=47.5 941 350 severe_disease (0.3719447 0.6280553) *
  
```

You can also plot directly using `rpart.plot` (Fig. 9.5).

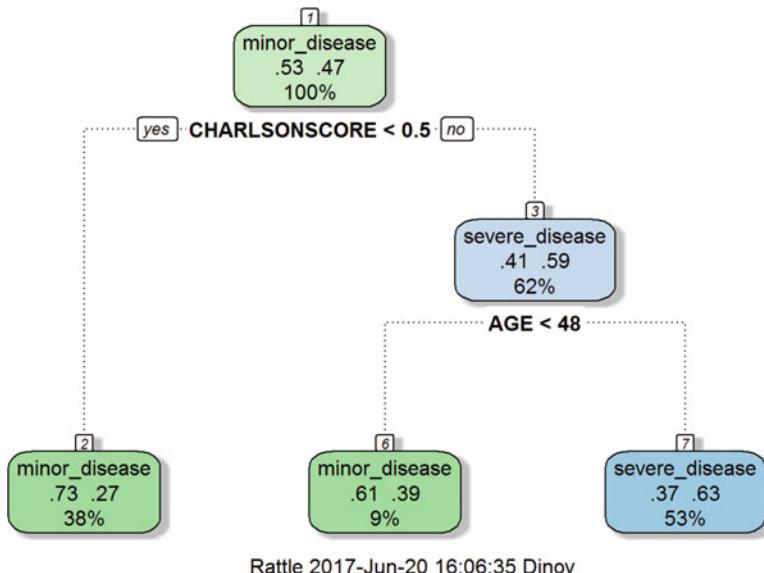
```

library(rpart.plot)
rpart.plot(qol_model, type = 4, extra = 1, clip.right.labs = F)
  
```

We can use `fancyRpartPlot` (Figs. 9.6).

```

library("rattle")
fancyRpartPlot(qol_model, cex = 1)
  
```



Rattle 2017-Jun-20 16:06:35 Dinov

Fig. 9.6 Another decision tree classification of the QoL data, compare to Fig. 9.5

```

qol_pred<-predict(qol_model, qol_test,type = 'class')
confusionMatrix(table(qol_pred, qol_test$cd))
## Confusion Matrix and Statistics
## qol_pred      minor_disease severe_disease
##   minor_disease          133           64
##   severe_disease          90          156
##
##               Accuracy : 0.6524
##               95% CI : (0.606, 0.6967)
##   No Information Rate : 0.5034
##   P-Value [Acc > NIR] : 1.759e-10
##   Kappa : 0.3053
##   Mcnemar's Test P-Value : 0.04395
##   Sensitivity : 0.5964
##   Specificity : 0.7091
##   Pos Pred Value : 0.6751
##   Neg Pred Value : 0.6341
##   Prevalence : 0.5034
##   Detection Rate : 0.3002
##   Detection Prevalence : 0.4447
##   Balanced Accuracy : 0.6528
##   'Positive' Class : minor_disease
  
```

These results are consistent with their counterparts reported using C5.0. How can we tune the parameters to further improve the results? (Fig. 9.7).

```

set.seed(1234)
control = rpart.control(cp = 0.000, xval = 100, minsplit = 2)
qol_model= rpart(cd ~ ., data = qol_train[, -40], control = control)
plotcp(qol_model)
  
```

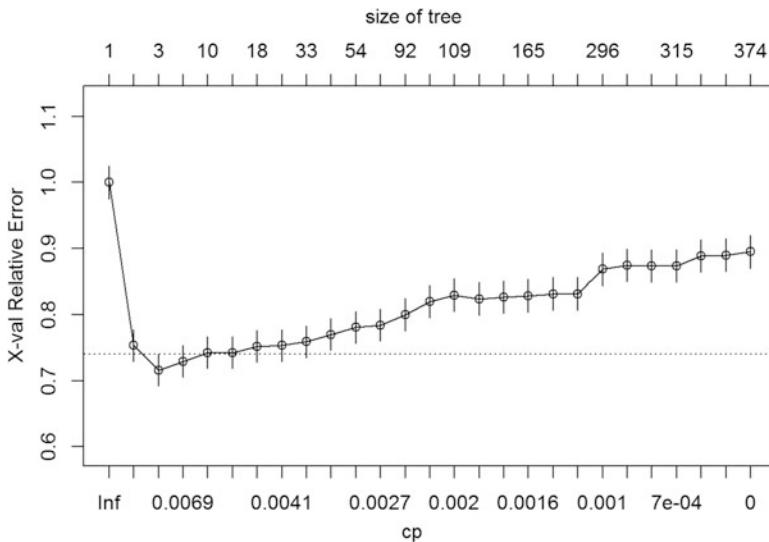


Fig. 9.7 Tuning the decision tree classification by reducing the error across the spectrum of cost-complexity pruning parameter (cp) and tree size

```
printcp(qol_model)

## Classification tree:
## rpart(formula = cd ~ ., data = qol_train[, -40], control = control)
##
## Variables actually used in tree construction:
## [1] AGE           CHARLSONSCORE  INTERVIEWDATE  LANGUAGE
## [5] MSA_Q_01      MSA_Q_02      MSA_Q_03      MSA_Q_04
## [9] MSA_Q_05      MSA_Q_06      MSA_Q_07      MSA_Q_08
## [13] MSA_Q_09     MSA_Q_10      MSA_Q_11      MSA_Q_12
## [17] MSA_Q_13     MSA_Q_14      MSA_Q_15      MSA_Q_16
## [21] MSA_Q_17     PH2_Q_01      PH2_Q_02      QOL_Q_01
## [25] QOL_Q_02     QOL_Q_03      QOL_Q_04      QOL_Q_05
## [29] QOL_Q_06     QOL_Q_07      QOL_Q_08      QOL_Q_09
## [33] QOL_Q_10     RACE_ETHNICITY SEX          TOS_Q_01
## [37] TOS_Q_02     TOS_Q_03      TOS_Q_04
##
## Root node error: 836/1771 = 0.47205
##
## n= 1771
##
##          CP nsplit rel error xerror      xstd
## 1 0.24641148      0 1.000000 1.00000 0.025130
## 2 0.04186603      1 0.7535885 0.75359 0.024099
## 3 0.00717703      2 0.7117225 0.71651 0.023816
## 4 0.00657895      3 0.7045455 0.72967 0.023920
## 5 0.00598086      9 0.6543062 0.74282 0.024020
## 6 0.00478469     14 0.6244019 0.74282 0.024020
## 7 0.00418660     17 0.6100478 0.75239 0.024090
## 8 0.00398724     21 0.5933014 0.75359 0.024099
## 9 0.00358852     32 0.5466507 0.75957 0.024141
```

```

## 10 0.00318979      41 0.5143541 0.77033 0.024215
## 11 0.00299043      53 0.4665072 0.78110 0.024286
## 12 0.00239234      59 0.4485646 0.78469 0.024309
## 13 0.00209330      91 0.3708134 0.80024 0.024406
## 14 0.00199362      95 0.3624402 0.82057 0.024522
## 15 0.00191388     108 0.3349282 0.83014 0.024574
## 16 0.00179426     122 0.2978469 0.82416 0.024542
## 17 0.00159490     151 0.2416268 0.82656 0.024555
## 18 0.00153794     164 0.2177033 0.82895 0.024567
## 19 0.00149522     171 0.2069378 0.83134 0.024580
## 20 0.00119617     182 0.1866029 0.83134 0.024580
## 21 0.00089713     295 0.0514354 0.86842 0.024758
## 22 0.00079745     306 0.0406699 0.87440 0.024783
## 23 0.00071770     309 0.0382775 0.87321 0.024778
## 24 0.00068353     314 0.0346890 0.87321 0.024778
## 25 0.00059809     321 0.0299043 0.88876 0.024841
## 26 0.00039872     367 0.0023923 0.88995 0.024846
## 27 0.00000000     373 0.0000000 0.89474 0.024864

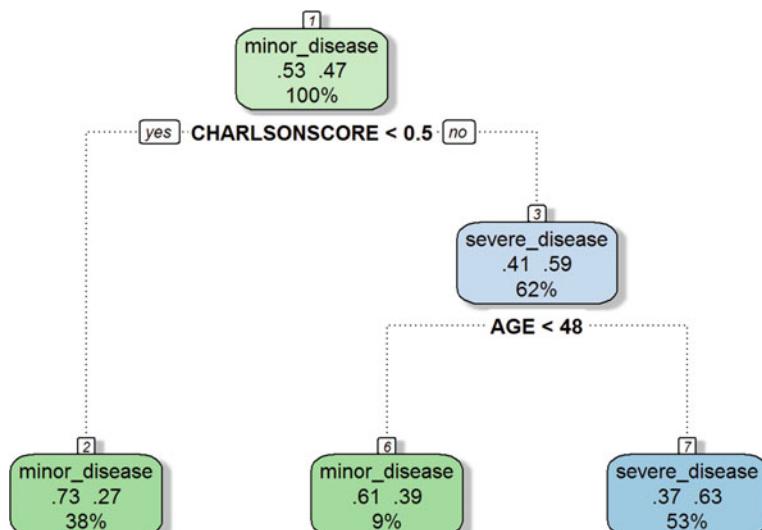
```

Now, we can prune the tree according to the optimal cp , *complexity parameter* to which the `rpart` object will be trimmed. Instead of using the *real error* (e.g., $1 - R^2$, RMSE) to capture the discrepancy between the observed labels and the model-predicted labels, we will use the `xerror`, which averages the discrepancy between observed and predicted classifications using *cross-validation*, see Chap. 21. Figs. 9.8, 9.9, and 9.10 show some alternative decision tree pruning results.

```

set.seed(1234)
selected_tr <- prune(qol_model, cp= qol_model$cptable[which.min(qol_model$cptable[,"xerror"]),"CP"])
fancyRpartPlot(selected_tr, cex = 1)

```



Rattle 2017-Jun-20 16:06:36 Dinov

Fig. 9.8 Pruned decision tree classification for the QoL data, compare to Figs. 9.5 and 9.6

```

qol_pred_tune<-predict(selected_tr, qol_test, type = 'class')
confusionMatrix(table(qol_pred_tune, qol_test$cd))

## Confusion Matrix and Statistics
## qol_pred_tune      minor_disease  severe_disease
##   minor_disease          133          64
##   severe_disease          90         156
##
##                               Accuracy : 0.6524
##                               95% CI : (0.606, 0.6967)
##   No Information Rate : 0.5034
##   P-Value [Acc > NIR] : 1.759e-10
##   Kappa : 0.3053
##   Mcnemar's Test P-Value : 0.04395
##   Sensitivity : 0.5964
##   Specificity : 0.7091
##   Pos Pred Value : 0.6751
##   Neg Pred Value : 0.6341
##   Prevalence : 0.5034
##   Detection Rate : 0.3002
##   Detection Prevalence : 0.4447
##   Balanced Accuracy : 0.6528
##   'Positive' Class : minor_disease

```

The result is roughly same as that of C5.0. Despite the fact that there is no substantial classification improvement, the tree-pruning process generates a graphical representation of the decision making protocol (`selected_tr`) that is much simpler and intuitive compared to the original (un-pruned) tree (`qol_model`):

```
fancyRpartPlot(qol_model, cex = 0.1)
```

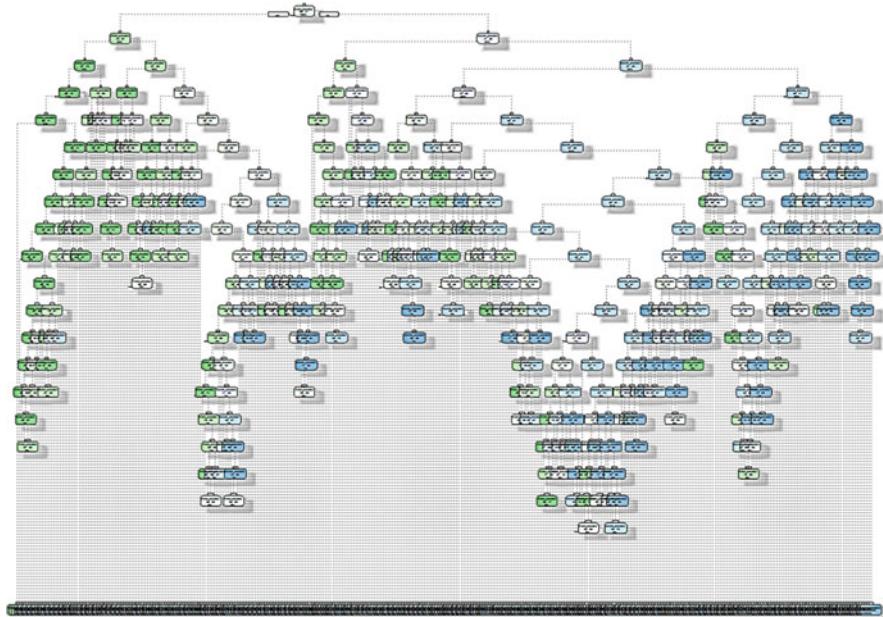


Fig. 9.9 Testing data (QoL dataset) decision tree prediction results (for chronic disease, CD)

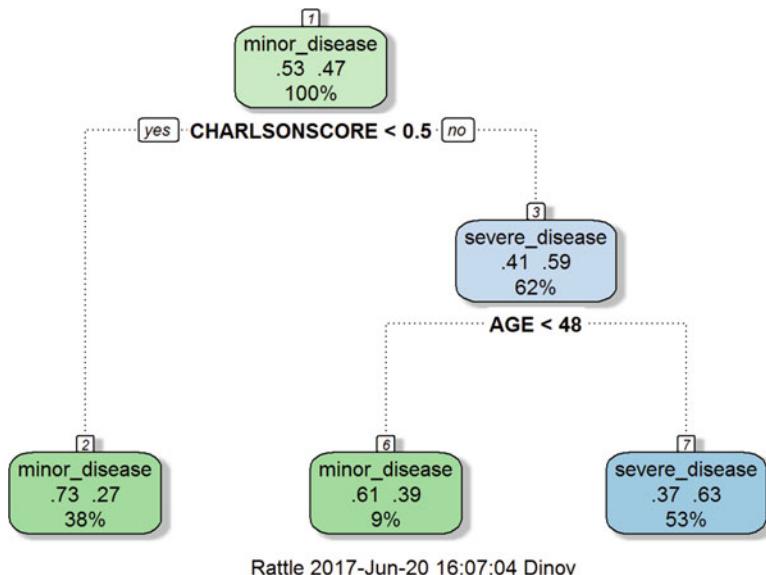


Fig. 9.10 Training data QoL decision tree plot

9.5 Compare Different Impurity Indices

We can change `split = "entropy"` to `"error"` or `"gini"` to apply an alternative information gain index. Experiment with these setting and compare the results.

```

set.seed(1234)
qol_model = rpart(cd ~ ., data=qol_train[ , -40],
parms = List(split = "entropy"))
fancyRpartPlot(qol_model, cex = 1)

# Modify and test using "error" and "gini"
# qol_pred<-predict(qol_model, qol_test,type = 'class')
# confusionMatrix(table(qol_pred, qol_test$cd))
  
```

9.6 Classification Rules

In addition to the classification trees we just saw, we can explore *classification rules* that utilize `if-else` logical statements to assign classes to unlabeled data. Below we review three classification rule strategies.

9.6.1 Separate and Conquer

Separate and conquer repeatedly splits the data (and subsets of the data) by rules that cover a subset of examples. This procedure is very similar to the *Divide and conquer*

approach. However, a notable difference is that each rule can be independent, and yet, each decision node in a tree has to be linked to past decisions.

9.6.2 *The One Rule Algorithm*

To understand the One Rule (OneR) algorithm, we need to know about its "sibling" - ZeroR rule. ZeroR rule means that we assign the mode class to unlabeled test observations regardless of its feature value. The One rule algorithm is an improved version of ZeroR that uses a single rule for classification. In other words, OneR splits the training dataset into several segments based on feature values. Then, it assigns the modes of the classes within each segment to related observations in the unlabeled test data. In practice, we first test multiple rules and pick the rule with the smallest error rate to be our One Rule. Remember, these rules may be subjective.

9.6.3 *The RIPPER Algorithm*

The *Repeated Incremental Pruning to Produce Error Reduction* algorithm is a combination of the ideas behind decision tree and classification rules. It consists of a three-step process:

- **Grow:** add conditions to a rule until it cannot split the data into more segments.
- **Prune:** delete some of the conditions that have large error rates.
- **Optimize:** repeat the above two steps until we cannot add or delete any of the conditions.

9.7 Case Study 2: QoL in Chronic Disease (Take 2)

Let's take another look at the same dataset as Case Study 1 - this time applying *classification rules*. Naturally, we will skip over the first two data handling steps and go directly to step three.

9.7.1 *Step 3: Training a Model on the Data*

Let's start by using the `OneR()` function in the `RWeka` package. Before installing the package you might want to check that the Java program in your computer is up to date. Also, its version has to match the version of R (i.e., 64bit R needs 64bit Java).

The function `OneR()` has the following invocation protocol:

```
m<-OneR(class~predictors, data=mydata)
```

- class: factor vector with the class for each row in **mydata**.
- predictors: feature variables in **mydata**. If we want to include x_1, x_2 as predictors and y as the class label variable, we do $y \sim x_1 + x_2$. To specify a full model, we use this notation: $y \sim .$, which includes all of the column variables as predictors.
- mydata: the dataset where the features and labels can be found.

```
# install.packages("RWeka")
library(RWeka)
# just remove the CHRONICDISEASESCORE but keep cd
set.seed(1234)
qol_1R<-OneR(cd~, data=qol[ , -40])
qol_1R

## CHARLSONSCORE:
## < -4.5 -> severe_disease
## < 0.5 -> minor_disease
## < 5.5 -> severe_disease
## < 8.5 -> minor_disease
## >= 8.5 -> severe_disease
## (1453/2214 instances correct)
```

Note that 1,453 out of 2,214 cases are correctly classified, 66%, by the “one rule”.

9.7.2 Step 4: Evaluating Model Performance

```
summary(qol_1R)

##
## === Summary ===
##
## Correctly Classified Instances      1453           65.6278 %
## Incorrectly Classified Instances    761            34.3722 %
## Kappa statistic                   0.3206
## Mean absolute error               0.3437
## Root mean squared error          0.5863
## Relative absolute error          68.8904 %
## Root relative squared error      117.3802 %
## Total Number of Instances        2214
##
## === Confusion Matrix ===
##
##      a   b  <- classified as
## 609 549 |   a = minor_disease
## 212 844 |   b = severe_disease
```

We obtained a single rule that correctly specifies 66% of the patients, which is in line with the prior decision tree classification results. Due to algorithmic stochasticity, it's normal that these results may vary each time you run the algorithm, albeit we used `seed(1234)` to ensure some result reproducibility.

9.7.3 Step 5: Alternative Model1

Another possible option for the classification rules would be the RIPPER rule algorithm that we discussed earlier in the chapter. In R we use the Java based function `JRip()` to invoke this algorithm.

`JRip()` function has the same components as the `OneR()` function:

```
m<-JRip(class~predictors, data=mydata)
set.seed(1234)
qol_jrip1<-JRip(cd~, data=qol[, -40])
qol_jrip1

## JRIP rules:
## =====
## (CHARLSONSCORE >= 1) and (RACE_ETHNICITY >= 4) and (AGE >= 49) => cd=severe_disease (448.0/132.0)
## (CHARLSONSCORE >= 1) and (AGE >= 53) => cd=severe_disease (645.0/265.0)
## => cd=minor_disease (1121.0/360.0)
##
## Number of Rules : 3
summary(qol_jrip1)

## Correctly Classified Instances      1457      65.8085 %
## Incorrectly Classified Instances    757      34.1915 %
## Kappa statistic                   0.3158
## Mean absolute error              0.4459
## Root mean squared error          0.4722
## Relative absolute error          89.3711 %
## Root relative squared error      94.5364 %
## Total Number of Instances        2214
## === Confusion Matrix ===
##      a   b  <- classified as
## 761 397 |   a = minor_disease
## 360 696 |   b = severe_disease
```

This `JRip()` classifier uses only three rules and has a relatively similar accuracy 66%. As each individual has unique characteristics, classification in real world data is rarely perfect (close to 100% accuracy).

9.7.4 Step 5: Alternative Model2

Another idea is to repeat the generation of trees multiple times, predict according to each tree's performance, and finally ensemble those weighted votes into a combined classification result. This is precisely the idea behind `random forest` classification, see Chap. 15 (Figs. 9.11 and 9.12).

```
require(randomForest)
set.seed(12)
# rf.fit <- tuneRF(qol_train[, -40], qol_train[, 40], stepFactor=1.5)
rf.fit <- randomForest(cd~, data=qol_train[, -40], importance=TRUE, ntree=2000, mtry=26)
varImpPlot(rf.fit); print(rf.fit)
```

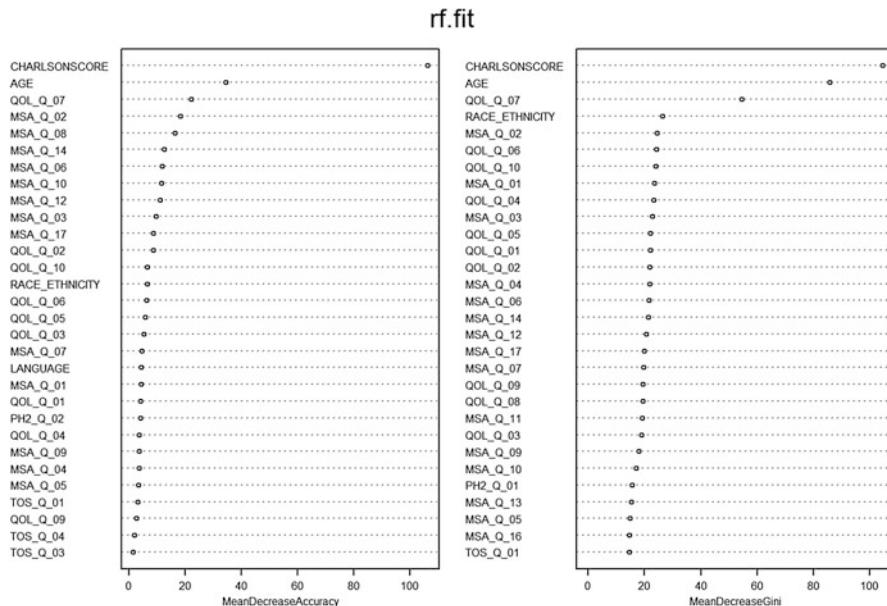


Fig. 9.11 Variable importance plots of random forest classification of the QoL CD variable using accuracy (left) and Gini index (right) as evaluation metrics

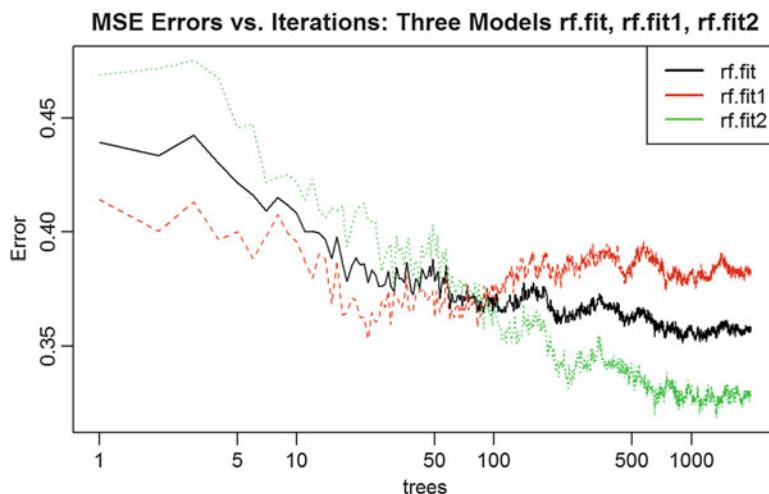


Fig. 9.12 Error plots of the random forest prediction of CD (QoL chronic disease) using three different trees models

```

## Call:
## randomForest(formula = cd ~ ., data = qol_train[, -40],
importance = TRUE, ntree = 2000, mtry = 26)
##             Type of random forest: classification
##                         Number of trees: 2000
## No. of variables tried at each split: 26
##
##                 OOB estimate of error rate: 35.86%
## Confusion matrix:
##                         minor_disease severe_disease class.error
## minor_disease              576           359  0.3839572
## severe_disease              276           560  0.3301435

rf.fit1 <- randomForest(cd~. , data=qol_train[ , -40], importance=TRUE, ntree=2000, mtry=26)
rf.fit2 <- randomForest(cd~. , data=qol_train[ , -40], importance=TRUE, node.size=5, ntree=5000, mtry=26)

plot(rf.fit, log="x", main="rf.fit (Black), rf.fit1 (Red), rf.fit2 (Green)")
points(1:5000, rf.fit1$mse, col="red", type="l")
points(1:5000, rf.fit2$mse, col="green", type="l")
qol_pred<-predict(rf.fit2, qol_test, type = 'class')
confusionMatrix(table(qol_pred, qol_test$cd))

## Confusion Matrix and Statistics
## qol_pred      minor_disease severe_disease
## minor_disease          138           69
## severe_disease          85          151
##
##                 Accuracy : 0.6524
##                 95% CI : (0.606, 0.6967)
## No Information Rate : 0.5034
## P-Value [Acc > NIR] : 1.759e-10
## Kappa : 0.305
## Mcnemar's Test P-Value : 0.2268
## Sensitivity : 0.6188
## Specificity : 0.6864
## Pos Pred Value : 0.6667
## Neg Pred Value : 0.6398
## Prevalence : 0.5034
## Detection Rate : 0.3115
## Detection Prevalence : 0.4673
## Balanced Accuracy : 0.6526
## 'Positive' Class : minor_disease

```

These variable importance plots (varplot) show the rank order of importance of the features according to the specific index (Accuracy, left, and Gini, right). More information about random forests is available in Chap. 15: Improving Model Performance.

In random forest (RF) classification, the node size (nodesize) refers to the smallest node that can be split, i.e., nodes with fewer cases than the nodesize are never subdivided. Increasing the node size leads to smaller trees, which may compromise previous predictive power. On the flip side, increasing the tree size

(maxnodes) and the number of trees (ntree) tends to increase the predictive accuracy. However, there are tradeoffs between increasing node-size and tree-size simultaneously. To optimize the RF predictive accuracy, try smaller node sizes and more trees. Ensembling (forest) results from a larger number of trees will likely generate better results.

9.8 Practice Problem

In the previous case study, we classified the CHRONICDISEASESCORE into two groups. What will happen if we use three groups? Let's separate CHRONICDISEASESCORE evenly into three groups. Recall the `quantile()` function that we talked about in Chap. 3. We can use it to get the cut-points for classification. Then, a `for` loop will help us split the variable CHRONICDISEASESCORE into three categories.

```
quantile(qol$CHRONICDISEASESCORE, probs = c(1/3, 2/3))
## 33.33333% 66.66667%
##      1.06      1.80

for(i in 1:2214){
  if(qol$CHRONICDISEASESCORE[i]>0.7&qol$CHRONICDISEASESCORE[i]<2.2){
    qol$cdthree[i]=2
  }
  else if(qol$CHRONICDISEASESCORE[i]>=2.2){
    qol$cdthree[i]=3
  }
  else{
    qol$cdthree[i]=1
  }
}

qol$cdthree<-factor(qol$cdthree, Levels=c(1, 2, 3), labels =
c("minor_disease", "mild_disease", "severe_disease"))
```

After labeling the three categories in the new variable `cdthree`, our job of preparing the class variable is done. Let's follow along the earlier sections in the chapter to determine how well the tree classifiers and the rule classifiers perform in the three-category case. First, try to build a tree classifier using `C5.0()` with 10 boost trials. One small tip is that in the training dataset, we cannot have column 40 (CHRONICDISEASESCORE), 41 (`cd`), and now 42 (`cdthree`) because they all contain class outcome related variables.

```

# qol_train1<-qol[1:2114, ]
# qol_test1<-qol[2115:2214, ]
train_index <- sample(seq_len(nrow(qol)), size = 0.8*nrow(qol))
qol_train1<-qol[train_index, ]
qol_test1<-qol[-train_index, ]

prop.table(table(qol_train1$cdthree))

##
##  minor_disease  mild_disease severe_disease
## 0.1699605      0.6459627      0.1840768

prop.table(table(qol_test1$cdthree))

##
##  minor_disease  mild_disease severe_disease
## 0.1760722      0.6478555      0.1760722

set.seed(1234)
qol_model1<-C5.0(qol_train1[ , -c(40, 41, 42)], qol_train1$cdthree,
trials=10)
qol_model1

##
## Call:
## C5.0.default(x = qol_train1[, -c(40, 41, 42)], y =
## qol_train1$cdthree, trials = 10)
##
## Classification Tree
## Number of samples: 1771
## Number of predictors: 39
##
## Number of boosting iterations: 10
## Average tree size: 230.5
##
## Non-standard options: attempt to group attributes

qol_pred1<-predict(qol_model1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
##          qol_pred1
##          minor_disease  mild_disease severe_disease
##  minor_disease      12          58            8
##  mild_disease       23         239           25
##  severe_disease      3          61           14
##
## Overall Statistics
##
##          Accuracy : 0.5982
## 95% CI : (0.5509, 0.6442)
##  No Information Rate : 0.8081
##  P-Value [Acc > NIR] : 1
##
##          Kappa : 0.0923
##  Mcnemar's Test P-Value : 4.174e-07
##
## Statistics by Class:
##

```

```

##          Class: minor_disease Class: mild_disease
## Sensitivity          0.31579          0.6676
## Specificity          0.83704          0.4353
## Pos Pred Value       0.15385          0.8328
## Neg Pred Value       0.92877          0.2372
## Prevalence           0.08578          0.8081
## Detection Rate       0.02709          0.5395
## Detection Prevalence 0.17607          0.6479
## Balanced Accuracy    0.57641          0.5514

##          Class: severe_disease
## Sensitivity          0.2979
## Specificity          0.8384
## Pos Pred Value       0.1795
## Neg Pred Value       0.9096
## Prevalence           0.1061
## Detection Rate       0.0316
## Detection Prevalence 0.1761
## Balanced Accuracy    0.5681

```

We can see that the prediction accuracy with three categories is way lower than the one we did with two categories.

Next, try to build a rule classifier with `OneR()`.

```

set.seed(1234)
qol_1R1<-OneR(cdthree~, data=qol[, -c(40, 41)])
qol_1R1

## INTERVIEWDATE:
## < 3.5    -> mild_disease
## < 28.5   -> severe_disease
## < 282.0  -> mild_disease
## < 311.5  -> severe_disease
## >= 311.5 -> mild_disease
## (1436/2214 instances correct)
summary(qol_1R1)

##
## === Summary ===
##
## Correctly Classified Instances      1436          64.86  %
## Incorrectly Classified Instances   778          35.14  %
## Kappa statistic                   0.022
## Mean absolute error              0.2343
## Root mean squared error          0.484
## Relative absolute error          67.5977 %
## Root relative squared error     116.2958 %
## Total Number of Instances        2214

##
## === Confusion Matrix ===
##
##      a     b     c  <- classified as
##      0    375    4 |   a = minor_disease
##      0   1422    9 |   b = mild_disease
##      0    390   14 |   c = severe_disease

qol_pred1<-predict(qol_1R1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
```

```

##          gol_pred1
##          minor_disease mild_disease severe_disease
##  minor_disease          0         78          0
##  mild_disease          0        285          2
##  severe_disease         0         76          2
##
## Overall Statistics
##
##          Accuracy : 0.6479
##  95% CI : (0.6014, 0.6923)
##  No Information Rate : 0.991
##  P-Value [Acc > NIR] : 1
##
##          Kappa : 0.012
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##          Class: minor_disease Class: mild_disease
## Sensitivity          NA          0.64920
## Specificity          0.8239      0.50000
## Pos Pred Value        NA          0.99303
## Neg Pred Value        NA          0.01282
## Prevalence            0.0000      0.99097
## Detection Rate        0.0000      0.64334
## Detection Prevalence  0.1761      0.64786
## Balanced Accuracy     NA          0.57460
##
##          Class: severe_disease
## Sensitivity          0.500000
## Specificity          0.826879
## Pos Pred Value        0.025641
## Neg Pred Value        0.994521
## Prevalence            0.009029
## Detection Rate        0.004515
## Detection Prevalence  0.176072
## Balanced Accuracy     0.663440

```

The OneRule classifier that is purely based on the value of the INTERVIEWDATE has 65% internal classification accuracy, and also 65% external (validation data) prediction accuracy. Although, the latter assessment is a bit misleading, as the vast majority of external validation data are classified in only one class - *mild_disease*.

Finally, let's revisit the JRip() classifier with the same three class labels according to cdthree.

```

set.seed(1234)
qol_jrip1<-JRip(cdthree~, data=qol[, -c(40, 41)])
qol_jrip1

## JRIP rules:
## =====
## (CHARLSONSCORE <= 0) and (AGE <= 50) and (MSA_Q_06 <= 1) and
## (QOL_Q_07 >= 1) and (MSA_Q_09 <= 1) => cdthree=minor_disease (35.0/11.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_10 >= 4) and (QOL_Q_07 >= 9) =>
## cdthree=severe_disease (54.0/20.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_02 >= 5) and (MSA_Q_09 <= 4) and
## (MSA_Q_04 >= 3) => cdthree=severe_disease (64.0/30.0)
## (CHARLSONSCORE >= 1) and (QOL_Q_02 >= 4) and (PH2_Q_01 >= 3) and
## (QOL_Q_10 >= 4) and (RACE_ETHNICITY >= 4) => cdthree=severe_disease
## (43.0/19.0)
## => cdthree=mild_disease (2018.0/653.0)
##
## Number of Rules : 5

summary(qol_jrip1)

## === Summary ===
##
## Correctly Classified Instances      1481      66.8925 %
## Incorrectly Classified Instances    733      33.1075 %
## Kappa statistic                   0.1616
## Mean absolute error               0.3288
## Root mean squared error          0.4055
## Relative absolute error          94.8702 %
## Root relative squared error     97.42   %
## Total Number of Instances        2214

## === Confusion Matrix ===
##
##      a     b     c  <- classified as
##  24   342   13 |   a = minor_disease
##  10  1365   56 |   b = mild_disease
##   1   311   92 |   c = severe_disease

qol_pred1<-predict(qol_jrip1, qol_test1)

confusionMatrix(table(qol_test1$cdthree, qol_pred1))

## Confusion Matrix and Statistics
##
##                qol_pred1
##                minor_disease  mild_disease  severe_disease
## minor_disease           5          70            3
## mild_disease            2          275           10
## severe_disease          0          61            17

##
## Overall Statistics
##   Accuracy : 0.6704
##   95% CI : (0.6245, 0.7141)
##   No Information Rate : 0.9165
##   P-Value [Acc > NIR] : 1
##
##   Kappa : 0.1583
##   Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##   Class: minor_disease  Class: mild_disease

```

```

## Sensitivity          0.71429      0.6773
## Specificity         0.83257      0.6757
## Pos Pred Value     0.06410      0.9582
## Neg Pred Value     0.99452      0.1603
## Prevalence          0.01580      0.9165
## Detection Rate     0.01129      0.6208
## Detection Prevalence 0.17607      0.6479
## Balanced Accuracy   0.77343      0.6765
##                               Class: severe_disease
## Sensitivity          0.56667
## Specificity         0.85230
## Pos Pred Value     0.21795
## Neg Pred Value     0.96438
## Prevalence          0.06772
## Detection Rate     0.03837
## Detection Prevalence 0.17607
## Balanced Accuracy   0.70948

```

In terms of the predictive accuracy on the testing data (`qol_test1$cdthree`), we can see from these outputs that the RIPPER algorithm performed better (67%) than the C5.0 decision tree (60%) and similarly to the OneR algorithm (65%), which suggests that simple algorithms might outperform complex methods for certain real world case-studies. Later, in Chap. 15, we will provide more details about optimizing and improving classification and prediction performance.

Try to replicate these results with other data from the list of our Case-Studies.

9.9 Assignments 9: Decision Tree Divide and Conquer Classification

9.9.1 Explain These Concepts

- Information Gain Measure
- Impurity
- Entropy
- Gini

9.9.2 Decision Tree Partitioning

Use the SOCR Neonatal Pain data to build and display a decision tree recursively partitioning the data using the provided features and attributes to split the data into similar classes.

- Collect and preprocess the data, e.g., data conversion and variable selection.
- Randomly split the data into training and testing sets.
- Train decision tree models on the data using C5.0 and rpart.

- Evaluate and compare the two models.
- Tune the rpart parameter and repeat the evaluation and comparison again.
- Assess the prediction accuracy and report the confusion matrix.
- Comment on different aspects of the prediction performance.
- Use various *impurity* measures and re-estimate the models.
- Try to use the RWeka package to train decision models and compare the results.
- Try to apply *Random Forest* and obtain variables importance plot.

References

- Fischetti, T, Lantz, B, Abedin, J, Mittal, HV, Makhabel, B, Berlinger, E, Illes, F, Badics, M, Banai, A, Daroczi, G (2016) *R: Data Analysis and Visualization*, Packt Publishing Ltd, ISBN 1786460483, 9781786460486.
- Liu, H, Gegov, A, Cocea, M. (2015) *Rule Based Systems for Big Data: A Machine Learning Approach*, Springer, Volume 13 (Studies in Big Data), ISBN 3319236962, 9783319236964.
- Witten, IH, Frank, E, Hall, MA, Pal, CJ. (2016) *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, Series in Data Management Systems, ISBN 0128043571, 9780128043578.

Chapter 10

Forecasting Numeric Data Using Regression Models



In the previous Chaps. 7, 8, and 9, we covered classification methods that use mathematical formalism to address everyday life prediction problems. In this Chapter, we will focus on specific model-based statistical methods providing forecasting and classification functionality. Specifically, we will (1) demonstrate the predictive power of multiple linear regression; (2) show the foundation of regression trees and model trees; and (3) examine two complementary case-studies (Baseball Players and Heart Attack).

It may be helpful to first review Chap. 5 (Linear Algebra/Matrix Manipulations) and Chap. 7 (Introduction to Machine Learning).

10.1 Understanding Regression

Regression is a measurement of relationship between a *dependent variable* (value to be predicted) and a group of *independent variables* (predictors similar to features, discussed in Chap. 7). We assume the relationship between our dependent variable and independent variables follows a predefined model, e.g., an affine or hyper-linear model.

10.1.1 Simple Linear Regression

First recall the material presented in Chap. 5 (*Linear Algebra & Matrix Computing*). The simplest case of regression modeling involves a single predictor.

$$y = a + bx.$$

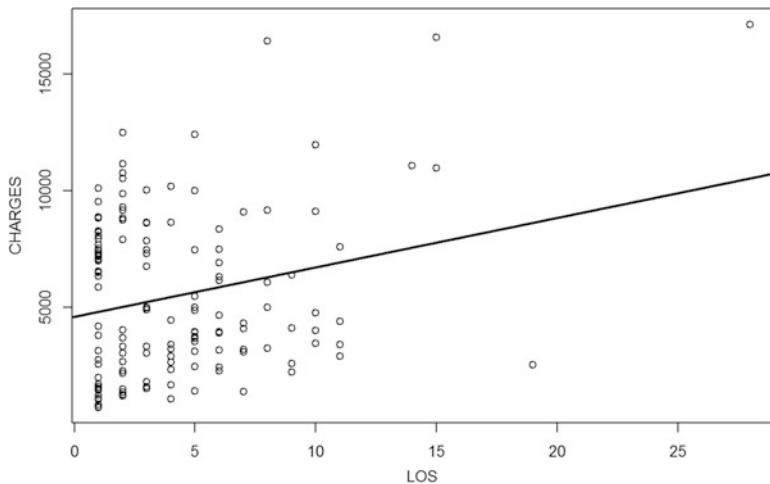


Fig. 10.1 Scatterplot and a linear model of length of stay (LOS) vs. hospital charges for the heart attack data

This formula should appear familiar by now. In this slope-intercept analytical expression, a is our intercept while b is the slope. That is an equation form of the simple linear regression model. If we know a and b , for any given x (input) we can *predict* y (output) via the above formula. If we plot x and y in a 2D coordinate system, the model is graphically represented as a straight line.

However, this is the ideal case. When we plot using real world data, the pattern may be harder to recognize. Let's look at the scatter plot (see Chap. 3) and simple linear regression line of two variables "hospital charges" or CHARGES (independent variable) and length of stay in the hospital or LOS (predictor). The data is available online, CaseStudy12_AdultsHeartAttack_Data. We removed two observations that have missing data using the command `heart_attack<-heart_attack[complete.cases(heart_attack),]`.

```
heart_attack<-
read.csv("https://umich.instructure.com/files/1644953/download?download_frd
=1", stringsAsFactors = F) heart_attack$CHARGES<-as.numeric
(heart_attack$CHARGES)

## Warning: NAs introduced by coercion

heart_attack<-heart_attack[complete.cases(heart_attack), ]

fit1<-lm(CHARGES~LOS, data=heart_attack)
par(cex=.8)
plot(heart_attack$LOS, heart_attack$CHARGES, xLab="LOS", yLab = "CHARGES")
abline(fit1, lwd=2)
```

It seems to be common sense that the longer you stay in the hospital, the higher the medical costs will be. However, on the scatter plot, we have only a bunch of dots showing some sign of an increasing pattern (Fig. 10.1).

The estimated expression for this regression line is:

$$\hat{y} = 4582.70 + 212.29 \times x,$$

or equivalently

$$CHARGES = 4582.70 + 212.29 \times LOS.$$

It is simple to make predictions with this regression line. Assume we have a patient that spent 10 days in hospital, then we have $LOS=10$. The predicted charge is likely to be $\$4582.70 + \$212.29 \times 10 = \$6705.6$. Plugging x into the expression equation automatically gives us an estimated value of the outcome y . This Chapter of the Probability and statistics EBook provides an introduction to linear modeling (http://wiki.socr.umich.edu/index.php/EBook#Chapter_X:_Correlation_and_Regession).

10.2 Ordinary Least Squares Estimation

How did we get the estimated expression? The most common estimating method in statistics is *ordinary least squares* (OLS). OLS estimators are obtained by minimizing the sum of the squared errors – that is the sum of squared vertical distances between each point on the scatter plot and its predicted value on the regression line (Fig. 10.2).

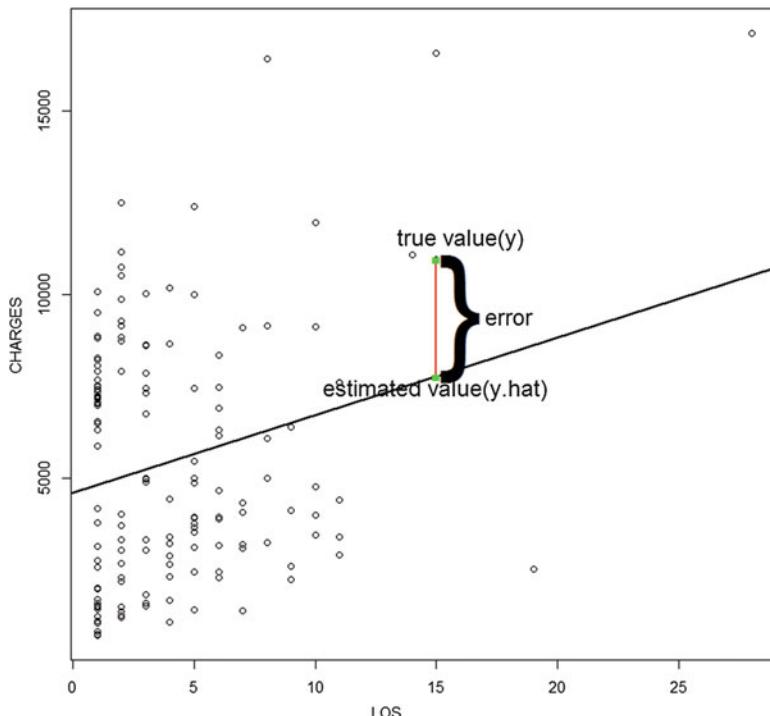


Fig. 10.2 Graphical representation of the residuals representing the difference between observed and predicted values

OLS is minimizing the following formula:

$$\sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - (a + b \times x_i))^2 = \sum_{i=1}^n e_i^2.$$

Some simple mathematical operations to minimize the sum square error yield the following solution for the slope parameter b :

$$b = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}.$$

While the intercept a is given by:

$$a = \bar{y} - b\bar{x}.$$

Recall what we learned in Chap. 3, where the variance was obtained by averaging the sum of squares $(var(x) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2)$. When we use \bar{x} to estimate the mean of x , we have the following formula for the sample variance: $var(x) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$. We can see that this is $\frac{1}{n-1}$ times the denominator of b . Similar to the variance, the covariance of x and y measures the average sum of the x deviance times the y deviance.

$$Cov(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x)(y_i - \mu_y).$$

If we use sample averages (\bar{x}, \bar{y}) , we have: $Cov(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$. This is $\frac{1}{n-1}$ times the numerator of b .

Combining the above, we get an estimate of the slope coefficient (effect-size of LOS on Charge):

$$b = \frac{Cov(x, y)}{var(x)}.$$

Let's examine these closed-form analytical expressions using the heart attack data.

```
b<-cov(heart_attack$LOS, heart_attack$CHARGES)/var(heart_attack$LOS); b
## [1] 212.2869
a<-mean(heart_attack$CHARGES)-b*mean(heart_attack$LOS); a
## [1] 4582.7
```

We can see that these estimates are exactly the same result as the previously reported.

10.2.1 Model Assumptions

Regression modeling has the following five key assumptions:

- Linear relationship,
- Multivariate normality,
- No or little multicollinearity,
- No auto-correlation, independence,
- Homoscedasticity.

10.2.2 Correlations

The SOCR Interactive Scatterplot Game (requires Java enabled browser) provides a dynamic interface demonstrating linear models, trends, correlations, slopes, and residuals.

Using the covariance, we can calculate the correlation, which indicates how closely the relationship between two variables follows a straight line.

$$\rho_{x,y} = \text{Corr}(x, y) = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y} = \frac{\text{Cov}(x, y)}{\sqrt{\text{Var}(x) \text{Var}(y)}}.$$

In R, correlation is given by `cor()` while square root of variance, or standard deviation, is given by `sd()`.

```
r<-cov(heart_attack$LOS, heart_attack$CHARGES)/(sd(heart_attack$LOS)*
sd(heart_attack$CHARGES))
r
## [1] 0.2449743
cor(heart_attack$LOS, heart_attack$CHARGES)
## [1] 0.2449743
```

The same outputs are obtained by the manual and the automated correlation calculations. This correlation is a positive number that is relatively small. We can say there is a weak positive linear association between these two variables. If we have a negative correlation estimate, it suggests a negative linear association. We have a weak association when $0.1 \leq \text{Cor} < 0.3$, a moderate association for $0.3 \leq \text{Cor} < 0.5$, and a strong association for $0.5 \leq \text{Cor} \leq 1.0$. If the correlation is below 0.1 then it suggests little to no linear relation between the variables.

10.2.3 Multiple Linear Regression

In practice, most interesting problems involve multiple predictors and one dependent variable, which requires estimating a multiple linear model. That is:

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon,$$

or equivalently

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon.$$

We usually use the second notation method in statistics. This equation shows the linear relationship between k predictors and a dependent variable. In total we have $k + 1$ coefficients to estimate.

The matrix notation for corresponding to the above equation is:

$$Y = X\beta + \epsilon,$$

where

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

$$X = \begin{pmatrix} 1 & x_{11} & x_{21} & \dots & x_{k1} \\ 1 & x_{12} & x_{22} & \dots & x_{k2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & x_{2n} & \dots & x_{kn} \end{pmatrix},$$

$$\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix},$$

and

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

is the error term.

Similar to simple linear regression, our goal is to minimize sum of squared errors. Solving for β , we get:

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

This is the matrix form solution, where X^{-1} is the inverse matrix of X and X^T is the transpose matrix.

Let's write a simple R function `reg` (`x,y`), that implements this matrix formula.

```
reg<-function(y, x){
  x<-as.matrix(x)
  x<-cbind(Intercept=1, x)
  solve(t(x) %*% x) %*% t(x) %*% y
}
```

The method `solve()` is used to compute the matrix inverse and `%*%` is matrix multiplication.

Next, we will apply this function to our heart attack dataset. To begin, let's check if the simple linear regression output is the same as we calculated earlier.

```
reg(y=heart_attack$CHARGES, x=heart_attack$LOS)
## [,1]
## Intercept 4582.6997
##           212.2869
```

As the slope and intercept are consistent with our previous estimates, we can continue and include additional variables as predictors. For instance, we can just add `age` into the model.

```
str(heart_attack)
## 'data.frame': 148 obs. of 8 variables:
## $ Patient : int 1 2 3 4 5 6 7 8 9 10 ...
## $ DIAGNOSIS: int 41041 41041 41091 41081 41091 41091 41091 41091 41041 ...
41041 ...
## $ SEX      : chr "F" "F" "F" "F" ...
## $ DRG      : int 122 122 122 122 122 121 121 121 121 123 ...
## $ DIED     : int 0 0 0 0 0 0 0 0 1 ...
## $ CHARGES  : num 4752 3941 3657 1481 1681 ...
## $ LOS      : int 10 6 5 2 1 9 15 15 2 1 ...
## $ AGE      : int 79 34 76 80 55 84 84 70 76 65 ...
reg(y=heart_attack$CHARGES, x=heart_attack[, c(7, 8)])
## [,1]
## Intercept 7280.55493
## LOS        259.67361
## AGE        -43.67677
```

10.3 Case Study 1: Baseball Players

10.3.1 Step 1: Collecting Data

We utilize the MLB data "01a_data.txt". The dataset contains 1034 records of heights and weights for some current and recent Major League Baseball (MLB) Players. These data were obtained from different resources (e.g., IBM Many Eyes).

This dataset includes the following variables:

- **Name:** MLB Player Name,
- **Team:** The Baseball team the player was a member of at the time the data was acquired,
- **Position:** Player field position,
- **Height:** Player height in inch,
- **Weight:** Player weight in pounds, and
- **Age:** Player age at time of record.

10.3.2 Step 2: Exploring and Preparing the Data

Let's load this dataset first. We use `as.is=T` to make non-numerical vectors into characters. Also, we delete the Name variable because we don't need players' names in this case study.

```
mlb<- read.table('https://umich.instructure.com/files/330381/download?download_id_frd=1', as.is=T, header=T)
str(mlb)

## 'data.frame': 1034 obs. of 6 variables:
## $ Name   : chr "Adam_Donachie" "Paul_Bako" "Ramon_Hernandez"
##   "Kevin_Millar" ...
## $ Team   : chr "BAL" "BAL" "BAL" "BAL" ...
## $ Position: chr "Catcher" "Catcher" "Catcher" "First_Baseman" ...
## $ Height  : int 74 74 72 72 73 69 69 71 76 71 ...
## $ Weight  : int 180 215 210 210 188 176 209 200 231 180 ...
## $ Age    : num 23 34.7 30.8 35.4 35.7 ...
```

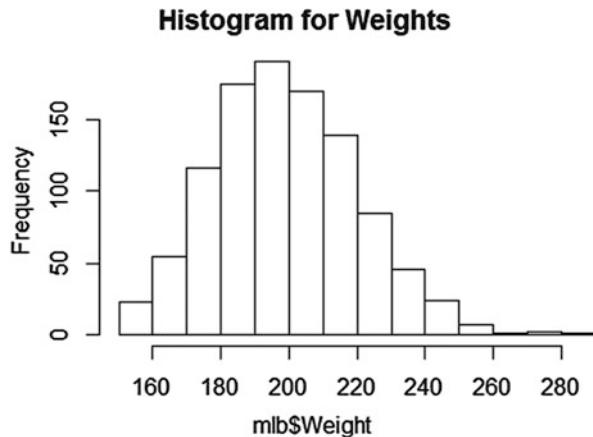
`mlb<-mlb[, -1]`

By looking at the `str()` output, we notice that the variable `TEAM` and `Position` are misspecified as characters. To fix this, we can use the function `as.factor()` to convert numerical or character vectors to factors.

```
mlb$Team<-as.factor(mlb$Team)
mlb$Position<-as.factor(mlb$Position)
```

The data is good to go. Let's explore it using some summary statistics and plots (Fig. 10.3).

Fig. 10.3 Frequency histogram of the MLB player's weights



```
summary(mlb$Weight)
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 150.0 187.0 200.0 201.7 215.0 290.0
hist(mlb$Weight, main = "Histogram for Weights")
```

The above plot illustrates our dependent variable `Weight`. As we learned in Chap. 3, this distribution appears somewhat right-skewed (Fig. 10.3).

Applying `GGpairs` to obtain a compact dataset summary we can mark heavy weight and light weight players (according to *light < median < heavy*) by different colors in the plot on Fig. 10.4

```
require(GGally)
mlb_binary = mlb

mlb_binary$bi_weight =
as.factor(ifelse(mlb_binary$Weight > median(mlb_binary$Weight), 1, 0))
g_weight <- ggpairs(data=mlb_binary[-1], title="MLB Light/Heavy Weights",
mapping=ggplot2::aes(colour = bi_weight),
lower=List(combo=wrap("facethist", binwidth=1)))
g_weight
```

Next, we may also mark player positions by different colors in the plot (Fig. 10.5).

```
g_position <- ggpairs(data=mlb[-1], title="MLB by Position",
mapping=ggplot2::aes(colour = Position),
lower=List(combo=wrap("facethist", binwidth=1)))
g_position
```

What about potential predictors?



Fig. 10.4 Pair plots of the MLB data by player's light (red) or heavy (blue) weights

```



```

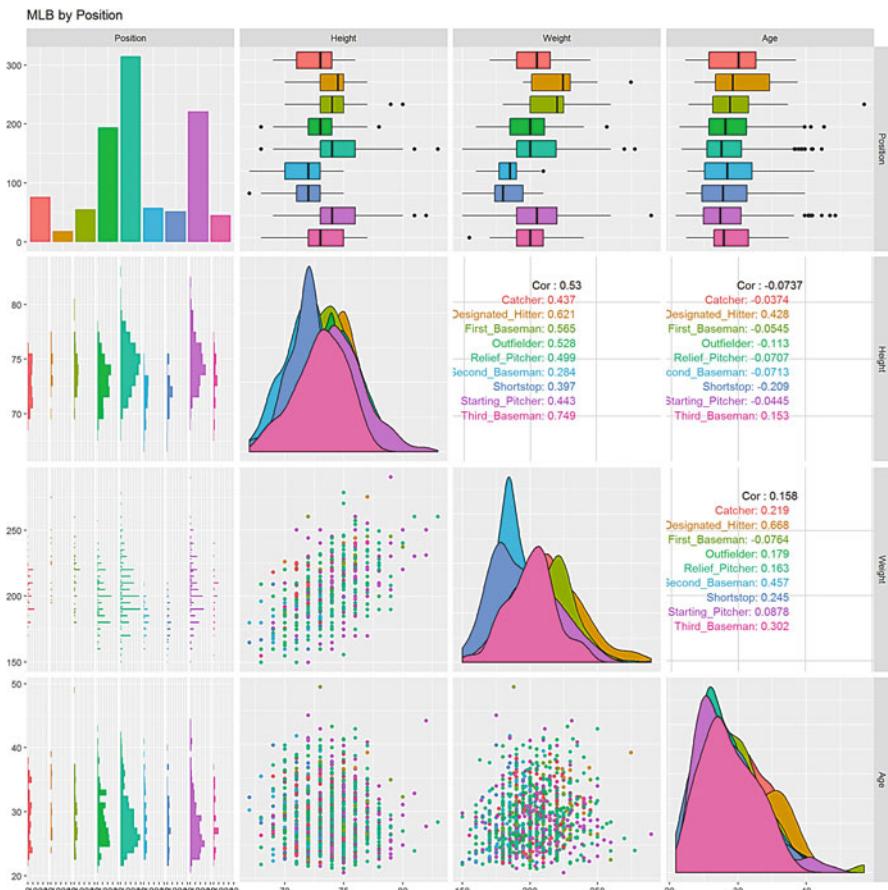


Fig. 10.5 Pair plots of the MLB data by position type

```
##      Min. 1st Qu. Median  Mean 3rd Qu.  Max.
## 67.0    72.0   74.0  73.7   75.0  83.0
summary(mLb$Age)
##      Min. 1st Qu. Median  Mean 3rd Qu.  Max.
## 20.90  25.44  27.92  28.74  31.23  48.52
```

In this case, we have two numerical predictors, two categorical predictors and 1,034 observations. Let's see how R treats different classes of variables.

10.3.3 Exploring Relationships Among Features: The Correlation Matrix

Before fitting a model, let's examine the independence of our potential predictors and the dependent variable. Multiple linear regression assumes that predictors are all independent of each other. Is this assumption valid? As we mentioned earlier, the `cor()` function can answer this question in pairwise manner. Note that we only look at numerical variables.

```
cor(mlb[c("Weight", "Height", "Age")])
##           Weight      Height      Age
## Weight  1.0000000  0.5303180  0.15784706
## Height  0.5303180  1.0000000 -0.07367013
## Age     0.1578471 -0.07367013  1.00000000
```

Observe that $cor(y, x) = cor(x, y)$ and $cov(x, x) = 1$. Also, our `Height` variable is weakly related to the players' age in a negative manner. This looks very good and wouldn't cause any multicollinearity problem. If two of our predictors are highly correlated, they both provide almost the same information, which could imply multicollinearity. A common practice is to delete one of them in the model or use dimensionality reduction methods.

10.3.4 Visualizing Relationships Among Features: The Scatterplot Matrix

To visualize pairwise correlations, we could use `scatterplot` or `pairs()` plot (Fig. 10.6).

```
pairs(mlb[c("Weight", "Height", "Age")])
```

You might get a sense of the data, but it is difficult to see any linear pattern. We can make a more sophisticated graph using `pairs.panels()` in the `psych` package (Fig. 10.7).

```
# install.packages("psych")
library(psych)
pairs.panels(mlb[, c("Weight", "Height", "Age")])
```

This plot provides much more information about the three variables. Above the diagonal, we have our correlation coefficients in numerical form. On the diagonal, there are histograms of variables. Below the diagonal, visual information is presented to help us understand the trend. This specific graph shows that height and weight are positively and strongly correlated. Also, the relationships between age and height, as well as, age and weight are very weak, see the horizontal red line in the panel below the main diagonal graphs, which indicates weak relationships (Fig. 10.7).

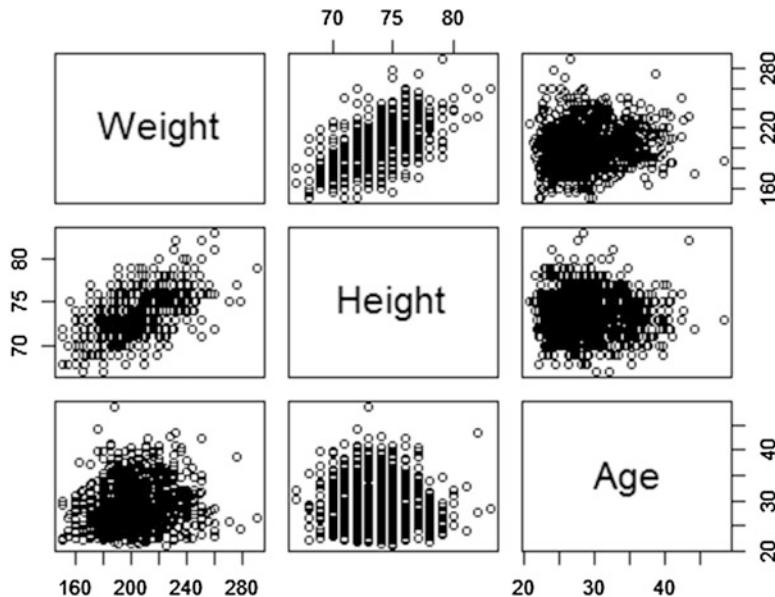


Fig. 10.6 MLB players weights, heights and ages

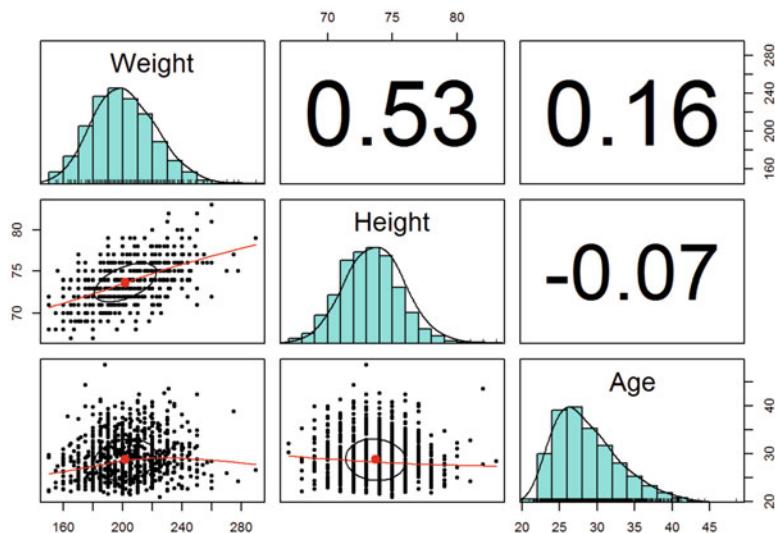


Fig. 10.7 A more detailed pairs plot of MLB players weights, heights and ages

10.3.5 Step 3: Training a Model on the Data

The function we are going to use now is `lm()`. No additional package is needed when using this function.

The `lm()` function has the following components:

`m<-lm(dv ~ iv, data=mydata)`

- dv: dependent variable
- iv: independent variables. Just like `OneR()` in Chap. 9, if we use `.` as `iv`, then all of the variables, except the dependent variable (`dv`), are included as predictors.
- data: specifies the data containing both dependent variable and independent variables.

```
fit<-Lm(Weight~, data=mlb)
fit

##
## Call:
## Lm(formula = Weight ~ ., data = mlb)
##
## Coefficients:
##              (Intercept)              TeamARZ
##                -164.9995             7.1881
##                TeamATL              TeamBAL
##                -1.5631             -5.3128
##                TeamBOS              TeamCHC
##                -0.2838             0.4026
##                TeamCIN              TeamCLE
##                2.1051             -1.3160
##                TeamCOL              TeamCWS
##                -3.7836             4.2944
##                TeamDET              TeamFLA
##                2.3024             2.6985
##                TeamHOU              TeamKC
##                -0.6808             -4.7664
##                TeamLA              TeamMIN
##                2.8598             2.1269
##                TeamMLW              TeamNYM
##                4.2897             -1.9736
##                TeamNYY              TeamOAK
##                1.7483             -0.5464
##                TeamPHI              TeamPIT
##                -6.8486             4.3023
##                TeamSD              TeamSEA
##                2.6133             -0.9147
##                TeamSF              TeamSTL
##                0.8411             -1.1341
##                TeamTB              TeamTEX
##                -2.6616             -0.7695
##                TeamTOR              TeamWAS
##                1.3943             -1.7555
## PositionDesignated_Hitter PositionFirst_Baseman
##                8.9037             2.4237
## PositionOutfielder PositionRelief_Pitcher
##                -6.2636             -7.7695
```

```

##   PositionSecond_Baseman      PositionShortstop
##                   -13.0843           -16.9562
##   PositionStarting_Pitcher    PositionThird_Baseman
##                   -7.3599            -4.6035
##   Height                      Age
##                   4.7175            0.8906

```

As we can see from the output, factors are included in the model by creating several indicators, one for each factor level. For each numerical variable, a corresponding model coefficient is estimated.

10.3.6 Step 4: Evaluating Model Performance

As we did in previous case-studies, let's examine the model performance (Figs. 10.8 and 10.9).

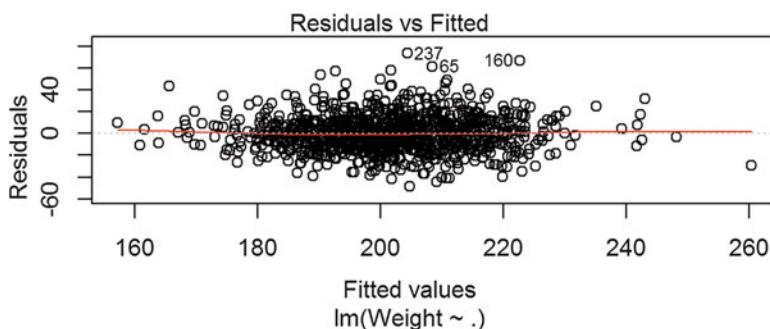


Fig. 10.8 Scatterplot of the residuals vs. model fitted values

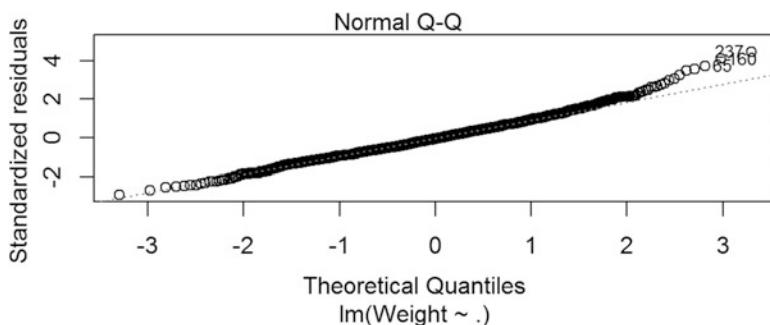


Fig. 10.9 QQ-normal plot of the residuals suggesting a linear model may explain the players' weight

```

summary(fit)

## 
## Call:
## Lm(formula = Weight ~ ., data = mlb)
## 
## Residuals:
##   Min     1Q Median     3Q    Max 
## -48.692 -10.909 -0.778  9.858 73.649 
## 
## Coefficients:
## (Intercept)          Estimate Std. Error t value Pr(>|t|)    
## TeamARZ                7.1881   4.2590   1.688  0.091777 .  
## TeamATL               -1.5631   3.9757  -0.393  0.694278  
## TeamBAL               -5.3128   4.0193  -1.322  0.186533  
## TeamBOS               -0.2838   4.0034  -0.071  0.943492  
## TeamCHC                0.4026   3.9949   0.101  0.919749  
## TeamCIN                2.1051   3.9934   0.527  0.598211  
## TeamCLE               -1.3160   4.0356  -0.326  0.744423  
## TeamCOL               -3.7836   4.0287  -0.939  0.347881  
## TeamCWS                4.2944   4.1022   1.047  0.295413  
## TeamDET                2.3024   3.9725   0.580  0.562326  
## TeamFLA                2.6985   4.1336   0.653  0.514028  
## TeamHOU               -0.6808   4.0634  -0.168  0.866976  
## TeamKC                 -4.7664   4.0242  -1.184  0.236525  
## TeamLA                 2.8598   4.0817   0.701  0.483686  
## TeamMIN                2.1269   4.0947   0.519  0.603579  
## TeamMLW                4.2897   4.0243   1.066  0.286706  
## TeamNYM                -1.9736   3.9493  -0.500  0.617370  
## TeamNYY                1.7483   4.1234   0.424  0.671655  
## TeamOAK                -0.5464   3.9672  -0.138  0.890474  
## TeamPHI                -6.8486   3.9949  -1.714  0.086778 .  
## TeamPIT                4.3023   4.0210   1.070  0.284890  
## TeamSD                 2.6133   4.0915   0.639  0.523148  
## TeamSEA                -0.9147   4.0516  -0.226  0.821436  
## TeamSF                 0.8411   4.0520   0.208  0.835593  
## TeamSTL                -1.1341   4.1193  -0.275  0.783132  
## TeamTB                 -2.6616   4.0944  -0.650  0.515798  
## TeamTEX                -0.7695   4.0283  -0.191  0.848556  
## TeamTOR                1.3943   4.0681   0.343  0.731871  
## TeamWAS                -1.7555   4.0038  -0.438  0.661142  
## PositionDesignated_Hitter 8.9037   4.4533   1.999  0.045842 *  
## PositionFirst_Baseman  2.4237   3.0058   0.806  0.420236  
## PositionOutfielder  -6.2636   2.2784  -2.749  0.006084 ** 
## PositionRelief_Pitcher -7.7695   2.1959  -3.538  0.000421 ***  
## PositionSecond_Baseman -13.0843   2.9638  -4.415  1.12e-05 ***  
## PositionShortstop   -16.9562   3.0406  -5.577  3.16e-08 ***  
## PositionStarting_Pitcher -7.3599   2.2976  -3.203  0.001402 ** 
## PositionThird_Baseman -4.6035   3.1689  -1.453  0.146613  
## Height                  4.7175   0.2563  18.405 < 2e-16 *** 
## Age                     0.8906   0.1259   7.075  2.82e-12 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 16.78 on 994 degrees of freedom 
## Multiple R-squared:  0.3858, Adjusted R-squared:  0.3617 
## F-statistic: 16.01 on 39 and 994 DF,  p-value: < 2.2e-16

plot(fit, which = 1:2)

```

The model summary shows us how well the model fits the data.

Residuals This tells us about the residuals. If we have extremely large or extremely small residuals for some observations compared to the rest of residuals, either they are outliers due to reporting error or the model fits data poorly. We have 73.649 as our maximum and -48.692 as our minimum. The residuals could be characterized by examining their range and by viewing the residual diagnostic plots.

Coefficients In this section of the output, we look at the very right column that has symbols like stars or dots showing if that variable is significant and should be included in the model. However, if no symbol is included next to a variable, then it means this estimated covariate coefficient in the linear model covariance could be trivial. Another thing we can look at is the $\text{Pr}(>|t|)$ column. A number close to zero in this column indicates the row variable is significant, otherwise it could be removed from the model.

In this example, some of the teams and positions are significant and some are not. Both *Age* and *Height* are significant.

R-squared What percent in y is explained by the included predictors? Here, we have 38.58%, which indicates the model is not bad but could be improved. Usually a well-fitted linear regression would have R-squared over 70%.

The **diagnostic plots** also help us understand the model quality.

Residual vs. Fitted This is the main residual diagnostic plot, Fig. 10.8. We can see that the residuals of observations indexed 65, 160 and 237 are relatively far apart from the rest. They may represent potential influential points or outliers.

Normal Q-Q This plot examines the normality assumption of the model, Fig. 10.9. The scattered dots represent the matched quantiles of the data and the normal distribution. If the Q-Q plot closely resembles a line bisecting the first quadrant in the plane, the normality assumption is valid. In our case, it is relatively close to the line. So, we can say that our model is valid in terms of normality.

10.4 Step 5: Improving Model Performance

We can employ the `step` function to perform forward or backward selection of important features/predictors. It works for both `lm` and `glm` models. In most cases, backward-selection is preferable because it tends to retain much larger models. On the other hand, there are various criteria to evaluate a model. The common model evaluation metrics include *AIC*, *BIC*, Adjusted R^2 , etc. In Chap. 14, we will present more details about prediction evaluation and assessment of classification. Let's compare the backward and forward model selection approaches. The `step` function argument `direction` allows this control (default is `both`, which will select the better result from either backward or forward selection).

```

step(fit,direction = "backward")
## Start:  AIC=5871.04
## Weight ~ Team + Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## - Team    29    9468 289262 5847.4
## <none>          279793 5871.0
## - Age     1    14090 293883 5919.8
## - Position 8    20301 300095 5927.5
## - Height   1    95356 375149 6172.3
##
## Step:  AIC=5847.45
## Weight ~ Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## <none>          289262 5847.4
## - Age     1    14616 303877 5896.4
## - Position 8    20406 309668 5901.9
## - Height   1    100435 389697 6153.6
##
## Call:
## lm(formula = Weight ~ Position + Height + Age, data = mlb)
##
## Coefficients:
## (Intercept)  PositionDesignated_Hitter
##             -168.0474                  8.6968
## PositionFirst_Baseman      PositionOutfielder
##                  2.7780                  -6.0457
## PositionRelief_Pitcher     PositionSecond_Baseman
##                  -7.7782                  -13.0267
## PositionShortstop      PositionStarting_Pitcher
##                  -16.4821                  -7.3961
## PositionThird_Baseman      Height
##                  -4.1361                  4.7639
## Age
##             0.8771
##
step(fit,direction = "forward")
## Start:  AIC=5871.04
## Weight ~ Team + Position + Height + Age
##
## Call:
## lm(formula = Weight ~ Team + Position + Height + Age, data = mlb)
##
## Coefficients:
## (Intercept)      TeamARZ
##             -164.9995      7.1881
## TeamATL        TeamBAL
##                  -1.5631     -5.3128
## TeamBOS        TeamCHC
##                  -0.2838      0.4026
## TeamCIN        TeamCLE
##                  2.1051     -1.3160
## TeamCOL        TeamCWS

```

```

##          -3.7836      4.2944
##          TeamDET      TeamFLA
##          2.3024      2.6985
##          TeamHOU      TeamKC
##          -0.6808     -4.7664
##          TeamLA       TeamMIN
##          2.8598      2.1269
##          TeamMLW      TeamNYM
##          4.2897     -1.9736
##          TeamNYY      TeamOAK
##          1.7483     -0.5464
##          TeamPHI      TeamPIT
##          -6.8486      4.3023
##          TeamSD       TeamSEA
##          2.6133     -0.9147
##          TeamSF       TeamSTL
##          0.8411     -1.1341
##          TeamTB       TeamTEX
##          -2.6616     -0.7695
##          TeamTOR      TeamWAS
##          1.3943     -1.7555
## PositionDesignated_Hitter PositionFirst_Baseman
##          8.9037      2.4237
## PositionOutfielder      PositionRelief_Pitcher
##          -6.2636     -7.7695
## PositionSecond_Baseman  PositionShortstop
##          -13.0843     -16.9562
## PositionStarting_Pitcher PositionThird_Baseman
##          -7.3599     -4.6035
##          Height        Age
##          4.7175      0.8906

step(fit,direction = "both")

## Start:  AIC=5871.04
## Weight ~ Team + Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## - Team    29      9468 289262 5847.4
## <none>          279793 5871.0
## - Age     1      14090 293883 5919.8
## - Position 8      20301 300095 5927.5
## - Height   1      95356 375149 6172.3
##
## Step:  AIC=5847.45
## Weight ~ Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## <none>          289262 5847.4
## + Team    29      9468 279793 5871.0
## - Age     1      14616 303877 5896.4
## - Position 8      20406 309668 5901.9
## - Height   1      100435 389697 6153.6
##
## Call:
## lm(formula = Weight ~ Position + Height + Age, data = mlb)

```

```

## 
## Coefficients:
## (Intercept) PositionDesignated_Hitter
## -168.0474 8.6968
## PositionFirst_Baseman PositionOutfielder
## 2.7780 -6.0457
## PositionRelief_Pitcher PositionSecond_Baseman
## -7.7782 -13.0267
## PositionShortstop PositionStarting_Pitcher
## -16.4821 -7.3961
## PositionThird_Baseman Height
## -4.1361 4.7639
## Age
## 0.8771

```

We can observe that forward retains the whole model. The better feature selection model uses backward step-wise selection.

Both backward and forward are greedy algorithms and neither guarantees an optimal model result. The optimal feature selection requires exploring every possible combination of the predictors, which is practically not feasible, due to computational complexity, $\binom{n}{k}$ combinations.

Alternatively, we can choose models based on various **information criteria**.

```

step(fit, k=2)

## Start: AIC=5871.04
## Weight ~ Team + Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## - Team    29     9468 289262 5847.4
## <none>          279793 5871.0
## - Age     1     14090 293883 5919.8
## - Position 8     20301 300095 5927.5
## - Height   1     95356 375149 6172.3
##
## Step: AIC=5847.45
## Weight ~ Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## <none>          289262 5847.4
## - Age     1     14616 303877 5896.4
## - Position 8     20406 309668 5901.9
## - Height   1     100435 389697 6153.6
##
## 
## Call:
## lm(formula = Weight ~ Position + Height + Age, data = mlb)
## 
## Coefficients:
## (Intercept) PositionDesignated_Hitter
## -168.0474 8.6968
## PositionFirst_Baseman PositionOutfielder

```

```

##                  2.7780          -6.0457
##  PositionRelief_Pitcher    PositionSecond_Baseman
##          -7.7782          -13.0267
##  PositionShortstop  PositionStarting_Pitcher
##          -16.4821          -7.3961
##  PositionThird_Baseman          Height
##          -4.1361          4.7639
##          Age
##          0.8771

step(fit,k=log(nrow(mlb)))

## Start:  AIC=6068.69
## Weight ~ Team + Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## - Team    29    9468 289262 5901.8
## <none>          279793 6068.7
## - Position  8    20301 300095 6085.6
## - Age      1    14090 293883 6112.5
## - Height    1    95356 375149 6365.0
##
## Step:  AIC=5901.8
## Weight ~ Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## <none>          289262 5901.8
## - Position  8    20406 309668 5916.8
## - Age      1    14616 303877 5945.8
## - Height    1    100435 389697 6203.0

##
## Call:
## lm(formula = Weight ~ Position + Height + Age, data = mlb)
##
## Coefficients:
## (Intercept)  PositionDesignated_Hitter
##          -168.0474          8.6968
##  PositionFirst_Baseman    PositionOutfielder
##          2.7780          -6.0457
##  PositionRelief_Pitcher    PositionSecond_Baseman
##          -7.7782          -13.0267
##  PositionShortstop  PositionStarting_Pitcher
##          -16.4821          -7.3961
##  PositionThird_Baseman          Height
##          -4.1361          4.7639
##          Age
##          0.8771

```

$k = 2$ yields the AIC criterion, and $k = \log(n)$ refers to BIC. Let's try to evaluate the model performance again (Figs. 10.10 and 10.11).

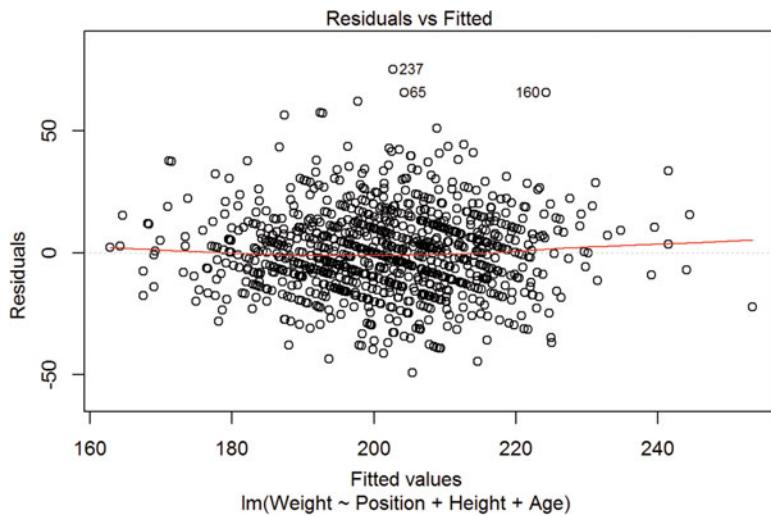


Fig. 10.10 Residuals vs. fitted values scatterplot

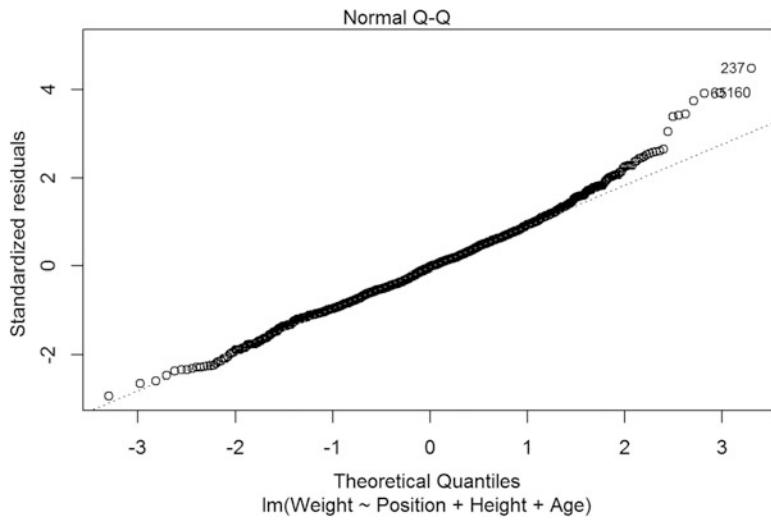


Fig. 10.11 QQ normal probability plot of the model residuals

```

fit2 = step(fit,k=2,direction = "backward")
## Start:  AIC=5871.04
## Weight ~ Team + Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## - Team    29    9468 289262 5847.4
## <none>          279793 5871.0
## - Age     1    14090 293883 5919.8
## - Position 8    20301 300095 5927.5
## - Height   1    95356 375149 6172.3
##
## Step:  AIC=5847.45
## Weight ~ Position + Height + Age
##
##          Df Sum of Sq    RSS    AIC
## <none>          289262 5847.4
## - Age     1    14616 303877 5896.4
## - Position 8    20406 309668 5901.9
## - Height   1    100435 389697 6153.6

summary(fit2)
##
## Call:
## lm(formula = Weight ~ Position + Height + Age, data = mlb)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -49.427 -10.855 - 0.344 10.110 75.301
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           -168.0474   19.0351 -8.828 < 2e-16 ***
## PositionDesignated_Hitter 8.6968    4.4258  1.965 0.049679 *  
## PositionFirst_Baseman   2.7780    2.9942  0.928 0.353741    
## PositionOutfielder     -6.0457    2.2778 -2.654 0.008072 **  
## PositionRelief_Pitcher -7.7782    2.1913 -3.550 0.000403 ***  
## PositionSecond_Baseman -13.0267   2.9531 -4.411 1.14e-05 ***
## PositionShortstop      -16.4821   3.0372 -5.427 7.16e-08 ***
## PositionStarting_Pitcher -7.3961   2.2959 -3.221 0.001316 **  
## PositionThird_Baseman  -4.1361   3.1656 -1.307 0.191647    
## Height                  4.7639    0.2528 18.847 < 2e-16 ***
## Age                     0.8771    0.1220  7.190 1.25e-12 ***
## ---                     
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.82 on 1023 degrees of freedom
## Multiple R-squared:  0.365,  Adjusted R-squared:  0.3588
## F-statistic: 58.81 on 10 and 1023 DF,  p-value: < 2.2e-16
plot(fit2, which = 1:2)

```

Sometimes, we prefer a simpler model even if there is slight loss in performance. In this case, we have a simpler model and $R^2 = 0.365$. The whole model is still very significant. Some potential influential points or outliers that are relatively far from other residuals observations are shown on Fig. 10.12.

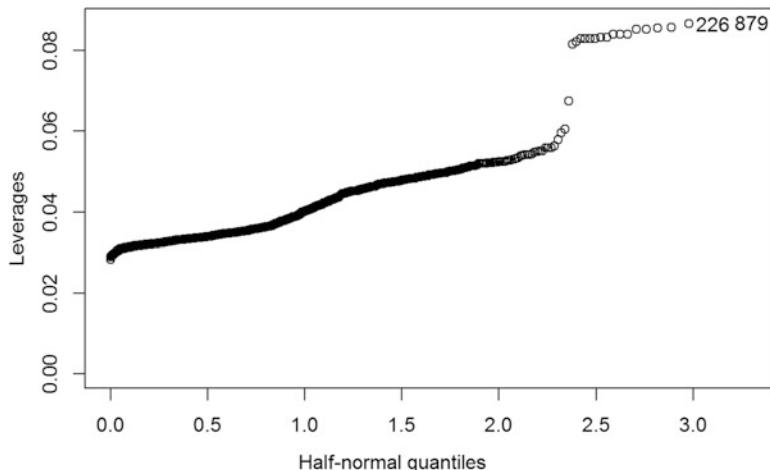


Fig. 10.12 A half-normal probability plot suggesting important factors or interactions by estimating the impact of a given main effect, or interaction, and its rank relative to other main effects and interactions computed via least squares estimation. The horizontal and vertical axes represent the $(n-1)$ theoretical order statistic medians from a half-normal distribution and the ordered absolute value of the estimated effects for the main factors and available interactions, respectively

```
# Half-normal plot for leverages
# install.packages("faraway")
library(faraway)

halfnorm(lm.influence(fit)$hat, nLab = 2, yLab="Leverages")
```

```
mlb[c(226,879),]
##      Team          Position Height Weight   Age
## 226  NYY Designated_Hitter    75    230 36.14
## 879  SD Designated_Hitter    73    200 25.60
summary(mlb)
##      Team          Position      Height      Weight
##  NYM   : 38  Relief_Pitcher :315  Min.   :67.0  Min.   :150.0
##  ATL   : 37  Starting_Pitcher:221  1st Qu.:72.0  1st Qu.:187.0
##  DET   : 37  Outfielder      :194  Median  :74.0  Median  :200.0
##  OAK   : 37  Catcher        : 76  Mean    :73.7  Mean    :201.7
##  BOS   : 36  Second_Baseman : 58  3rd Qu.:75.0  3rd Qu.:215.0
##  CHC   : 36  First_Baseman  : 55  Max.    :83.0  Max.    :290.0
## (Other):813 (Other)          :115
##      Age
##  Min.  :20.90
##  1st Qu.:25.44
##  Median :27.93
##  Mean   :28.74
##  3rd Qu.:31.23
##  Max.   :48.52
```

A deeper discussion of variable selection, controlling the false discovery rate, is provided in Chaps. 17 and 18.

10.4.1 Model Specification: Adding Non-linear Relationships

In linear regression, the relationship between independent and dependent variables is assumed to be linear. However, this might not be the case. The relationship between age and weight could be quadratic, since middle-aged people might gain weight dramatically.

```

mлb$age2<-mлb$Age)^2
fit2<-Lm(Weight~., data=mлb)
summary(fit2)

## Call:
## Lm(formula = Weight ~ ., data = mлb)
##
## Residuals:
##      Min      1Q  Median      3Q      Max
## -49.068 -10.775 -1.021  9.922  74.693
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)            -209.07068  27.49529 -7.604 6.65e-14 ***
## TeamARZ                  7.41943  4.25154  1.745 0.081274 .  
## TeamATL                 -1.43167  3.96793 -0.361 0.718318  
## TeamBAL                 -5.38735  4.01119 -1.343 0.179552  
## TeamBOS                 -0.06614  3.99633 -0.017 0.986799  
## TeamCHC                  0.14541  3.98833  0.036 0.970923  
## TeamCIN                  2.24022  3.98571  0.562 0.574201  
## TeamCLE                 -1.07546  4.02870 -0.267 0.789563  
## TeamCOL                 -3.87254  4.02069 -0.963 0.335705  
## TeamCWS                  4.20933  4.09393  1.028 0.304111  
## TeamDET                  2.66990  3.96769  0.673 0.501160  
## TeamFLA                  3.14627  4.12989  0.762 0.446343  
## TeamHOU                 -0.77230  4.05526 -0.190 0.849000  
## TeamKC                  -4.90984  4.01648 -1.222 0.221837  
## TeamLA                  3.13554  4.07514  0.769 0.441820  
## TeamMIN                  2.09951  4.08631  0.514 0.607512  
## TeamMLW                  4.16183  4.01646  1.036 0.300363  
## TeamNYM                 -1.25057  3.95424 -0.316 0.751870  
## TeamNYY                  1.67825  4.11502  0.408 0.683482  
## TeamOAK                  -0.68235  3.95951 -0.172 0.863212  
## TeamPHI                 -6.85071  3.98672 -1.718 0.086039 .  
## TeamPIT                  4.12683  4.01348  1.028 0.304086  
## TeamSD                  2.59525  4.08310  0.636 0.525179  
## TeamSEA                 -0.67316  4.04471 -0.166 0.867853  
## TeamSF                  1.06038  4.04481  0.262 0.793255  
## TeamSTL                 -1.38669  4.11234 -0.337 0.736037  
## TeamTB                  -2.44396  4.08716 -0.598 0.550003  
## TeamTEX                 -0.68740  4.02023 -0.171 0.864270  
## TeamTOR                  1.24439  4.06029  0.306 0.759306  
## TeamWAS                 -1.87599  3.99594 -0.469 0.638835  
## PositionDesignated_Hitter 8.94440  4.44417  2.013 0.044425 *  
## PositionFirst_Baseman    2.55100  3.00014  0.850 0.395368  
## PositionOutfielder     -6.25702  2.27372 -2.752 0.006033 **  
## PositionRelief_Pitcher   -7.68904  2.19166 -3.508 0.000471 ***  
## PositionSecond_Baseman   -13.01400  2.95787 -4.400 1.20e-05 ***  
## PositionShortstop      -16.82243  3.03494 -5.543 3.81e-08 ***  
## PositionStarting_Pitcher -7.08215  2.29615 -3.084 0.002096 **
```

```
## PositionThird_Baseman    -4.66452   3.16249  -1.475  0.140542
## Height                  4.71888   0.25578  18.449  < 2e-16 ***
## Age                     3.82295   1.30621   2.927  0.003503 **
## age2                    -0.04791   0.02124  -2.255  0.024327 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.74 on 993 degrees of freedom
## Multiple R-squared:  0.3889, Adjusted R-squared:  0.3643
## F-statistic: 15.8 on 40 and 993 DF,  p-value: < 2.2e-16
```

This actually brought up the overall R^2 up to 0.3889.

10.4.2 Transformation: Converting a Numeric Variable to a Binary Indicator

As discussed earlier, middle-aged people might have a different pattern in weight increase compared to younger people. The overall pattern could be not cumulative, but could rather represent alternative lines for the young and the middle-aged people. We assume 30 is the age threshold separating young from middle-age players. Players over 30 may have a steeper line for weight increase than those under 30. Here, we use the `ifelse()` function that we mentioned in Chap. 8 to create the indicator of this Age threshold.

```

## TeamHOU      -0.4964  4.0659 -0.122 0.902846
## TeamKC      -4.7138  4.0238 -1.171 0.241692
## TeamLA      2.9194  4.0814  0.715 0.474586
## TeamMIN     2.2885  4.0965  0.559 0.576528
## TeamMLW      4.4749  4.0269  1.111 0.266731
## TeamNYM     -1.8173  3.9510 -0.460 0.645659
## TeamNYY      1.7074  4.1229  0.414 0.678867
## TeamOAK     -0.3388  3.9707 -0.085 0.932012
## TeamPHI     -6.6192  3.9993 -1.655 0.098220 .
## TeamPIT      4.6716  4.0332  1.158 0.247029
## TeamSD      2.8600  4.0965  0.698 0.485243
## TeamSEA     -1.0121  4.0518 -0.250 0.802809
## TeamSF       1.0244  4.0545  0.253 0.800587
## TeamSTL     -1.1894  4.1187 -0.269 0.787703
## TeamTB      -2.4485  4.0980 -0.597 0.550312
## TeamTEX     -0.6112  4.0300 -0.152 0.879485
## TeamTOR      1.3959  4.0674  0.343 0.731532
## TeamWAS     -1.4189  4.0139 -0.354 0.723784
## PositionDesignated_Hitter  9.2378  4.4621  2.070 0.038683 *
## PositionFirst_Baseman    2.6074  3.0096  0.866 0.386501
## PositionOutfielder     -6.0408  2.2863 -2.642 0.008367 **
## PositionRelief_Pitcher   -7.5100  2.2072 -3.403 0.000694 ***
## PositionSecond_Baseman   -12.8870 2.9683 -4.342 1.56e-05 ***
## PositionShortstop      -16.8912 3.0406 -5.555 3.56e-08 ***
## PositionStarting_Pitcher -7.0825  2.3099 -3.066 0.002227 **
## PositionThird_Baseman   -4.4307  3.1719 -1.397 0.162773
## Age          0.6904  0.2153  3.207 0.001386 **
## age30        2.2636  1.9749  1.146 0.251992
## Height       4.7113  0.2563  18.380 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.77 on 993 degrees of freedom
## Multiple R-squared:  0.3866, Adjusted R-squared:  0.3619
## F-statistic: 15.65 on 40 and 993 DF,  p-value: < 2.2e-16

```

This model performs worse than the quadratic model in terms of R^2 . Moreover, age30 is not significant. So, we will stick with the earlier quadratic model.

10.4.3 Model Specification: Adding Interaction Effects

So far, each feature's individual effect was considered in our models. It is possible that features act in pairs to affect the independent variable. Let's examine that deeper.

Interactions are combined effects of two or more features. If we are not sure whether two features interact term we could test by adding an interaction term into the model. If the interaction term is significant, it confirms that there may be non-trivial interaction between the features.

```

fit4<-Lm(Weight~Team+Height+Age*Position+age2, data=mlb)
summary(fit4)

## Call:
## Lm(formula = Weight ~ Team + Height + Age * Position + age2,
##      data = mlb)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -48.761 -11.049  -0.761   9.911  75.533
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           -199.15403  29.87269 -6.667 4.35e-11 ***
## TeamARZ                  8.10376  4.26339  1.901  0.0576 .  
## TeamATL                 -0.81743  3.97899 -0.205  0.8373  
## TeamBAL                 -4.64820  4.03972 -1.151  0.2502  
## TeamBOS                  0.37698  4.00743  0.094  0.9251  
## TeamCHC                  0.33104  3.99507  0.083  0.9340  
## TeamCIN                  2.56023  3.99603  0.641  0.5219  
## TeamCLE                 -0.66254  4.03154 -0.164  0.8695  
## TeamCOL                  -3.72098  4.03759 -0.922  0.3570  
## TeamCWS                  4.63266  4.10884  1.127  0.2598  
## TeamDET                  3.21380  3.98231  0.807  0.4199  
## TeamFLA                  3.56432  4.14902  0.859  0.3905  
## TeamHOU                 -0.38733  4.07249 -0.095  0.9242  
## TeamKC                   -4.66678  4.02384 -1.160  0.2464  
## TeamLA                   3.51766  4.09400  0.859  0.3904  
## TeamMIN                  2.31585  4.10502  0.564  0.5728  
## TeamMLW                  4.34793  4.02501  1.080  0.2803  
## TeamNYM                  -0.28505  3.98537 -0.072  0.9430  
## TeamNYY                  1.87847  4.12774  0.455  0.6491  
## TeamOAK                  -0.23791  3.97729 -0.060  0.9523  
## TeamPHI                  -6.25671  3.99545 -1.566  0.1177  
## TeamPIT                  4.18719  4.01944  1.042  0.2978  
## TeamSD                   2.97028  4.08838  0.727  0.4677  
## TeamSEA                 -0.07220  4.05922 -0.018  0.9858  
## TeamSF                   1.35981  4.07771  0.333  0.7388  
## TeamSTL                 -1.23460  4.11960 -0.300  0.7645  
## TeamTB                   -1.90885  4.09592 -0.466  0.6413  
## TeamTEX                 -0.31570  4.03146 -0.078  0.9376  
## TeamTOR                  1.73976  4.08565  0.426  0.6703  
## TeamWAS                 -1.43933  4.00274 -0.360  0.7192  
## Height                  4.70632  0.25646 18.351 < 2e-16 ***
## Age                     3.32733  1.37088  2.427  0.0154 *  
## PositionDesignated_Hitter -44.82216 30.68202 -1.461  0.1444  
## PositionFirst_Baseman    23.51389 20.23553  1.162  0.2455  
## PositionOutfielder      -13.33140 15.92500 -0.837  0.4027  
## PositionRelief_Pitcher   -16.51308 15.01240 -1.100  0.2716  
## PositionSecond_Baseman   -26.56932 20.18773 -1.316  0.1884  
## PositionShortstop        -27.89454 20.72123 -1.346  0.1786  
## PositionStarting_Pitcher -2.44578 15.36376 -0.159  0.8736  
## PositionThird_Baseman    -10.20102 23.26121 -0.439  0.6611  
## age2                     -0.04201  0.02170 -1.936  0.0531 .  
## Age:PositionDesignated_Hitter 1.77289  1.00506  1.764  0.0780 .
```

```

## Age:PositionFirst_Baseman      -0.71111  0.67848 -1.048  0.2949
## Age:PositionOutfielder        0.24147  0.53650  0.450  0.6527
## Age:PositionRelief_Pitcher   0.30374  0.50564  0.601  0.5482
## Age:PositionSecond_Baseman   0.46281  0.68281  0.678  0.4981
## Age:PositionShortstop        0.38257  0.70998  0.539  0.5901
## Age:PositionStarting_Pitcher -0.17104  0.51976 -0.329  0.7422
## Age:PositionThird_Baseman    0.18968  0.79561  0.238  0.8116
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.73 on 985 degrees of freedom
## Multiple R-squared:  0.3945, Adjusted R-squared:  0.365
## F-statistic: 13.37 on 48 and 985 DF,  p-value: < 2.2e-16

```

The results indicates that the overall R^2 improved and some of the interactions are significant at the under 0.1 level.

10.5 Understanding Regression Trees and Model Trees

As we saw in Chap. 9, a decision tree builds by multiple `if-else` logical decisions and can classify observations. We could add regression into decision trees so that a decision tree can make numerical predictions.

10.5.1 Adding Regression to Trees

Numeric prediction trees are built in the same way as classification trees. Data are partitioned first via a divide-and-conquer strategy based on features. Recall that, homogeneity in classification trees may be assessed by measures like the entropy. In prediction, tree homogeneity is measured by statistics such as variance, standard deviation or absolute deviation, from the mean.

A common splitting criterion for regression trees is the **standard deviation reduction (SDR)**.

$$SDR = sd(T) - \sum_{i=1}^n \left| \frac{T_i}{T} \right| \times sd(T_i),$$

where $sd(T)$ is the standard deviation for the original data. After the summation of all segments, $\left| \frac{T_i}{T} \right|$ is the proportion of observations in the i th segment compared to the total number of observations and $sd(T_i)$ is the standard deviation for the i th segment. Let's look at one simple example.

*Original data : {1, 2, 3, 3, 4, 5, 6, 6, 7, 8},
 Split method 1 : {1, 2, 3|3, 4, 5, 6, 6, 7, 8}, and
 Split method 2 : {1, 2, 3, 3, 4, 5|6, 6, 7, 8}.*

In split method 1, $T_1 = \{1, 2, 3\}$, $T_2 = \{3, 4, 5, 6, 6, 7, 8\}$. In split method 2, $T_1 = \{1, 2, 3, 3, 4, 5\}$, $T_2 = \{6, 6, 7, 8\}$.

```
ori<-c(1, 2, 3, 3, 4, 5, 6, 6, 7, 8)
at1<-c(1, 2, 3)
at2<-c(3, 4, 5, 6, 6, 7, 8)
bt1<-c(1, 2, 3, 3, 4, 5)
bt2<-c(6, 6, 7, 8)
sdr_a <- sd(ori)-(Length(at1)/Length(ori)*sd(at1)+Length(at2)/Length(ori) *
sd(at2))
sdr_b <- sd(ori)-(Length(bt1)/Length(ori)*sd(bt1)+Length(bt2)/Length(ori) *
sd(bt2))
sdr_a
## [1] 0.7702557
sdr_b
## [1] 1.041531
```

`length()` is used in the above R codes to get the number of elements in a specific vector.

Larger SDR indicates greater reduction in standard deviation after splitting. Here, split method 2 yields greater SDR, so the regression tree split will use the second method, which results in more homogeneous sets than the first method.

Now, the tree will be split under `bt1` and `bt2` following the same rules (greater SDR wins). When we cannot split further `bt1` and `bt2` are terminal nodes. The observations classified into `bt1` will be predicted with $mean(bt1) = 3$, and those classified as `bt2` with $mean(bt2) = 6.75$.

10.6 Case Study 2: Baseball Players (Take 2)

10.6.1 Step 2: Exploring and Preparing the Data

We will continue with the MLB dataset, which includes 1,034 observations. Let's try to randomly separate them into training and testing datasets first.

```
set.seed(1234)
train_index <- sample(seq_len(nrow(mlb)), size = 0.75*nrow(mlb))
mlb_train<-mlb[train_index, ]
mlb_test<-mlb[-train_index, ]
```

We used a random 75–25% split to divide the data into training and testing sets.

10.6.2 Step 3: Training a Model On the Data

In R, the function `rpart()`, under the `rpart` package, provides regression tree modeling:

```
m<-rpart(dv~iv, data=mydata)
```

- dv: dependent variable
- iv: independent variable
- mydata: training data containing dv and iv.

We use two numerical features in the MLB data "01a_data.txt" `Age` and `Height` as features.

```
#install.packages("rpart")
library(rpart)

mlb.rpart<-rpart(Weight~Height+Age, data=mlb_train)
mlb.rpart

## n= 775
##
## node), split, n, deviance, yval
##      * denotes terminal node
##
## 1) root 775 323502.600 201.4361
##    2) Height< 73.5 366 112465.500 192.5000
##      4) Height< 70.5 55  9865.382 178.7818 *
##      5) Height>=70.5 311  90419.300 194.9260
##        10) Age< 31.585 234  71123.060 192.8547 *
##        11) Age>=31.585 77  15241.250 201.2208 *
##    3) Height>=73.5 409 155656.400 209.4328
##      6) Height< 76.5 335 118511.700 206.8627
##        12) Age< 28.6 194  75010.250 202.2938
##          24) Height< 74.5 76  20688.040 196.8026 *
##          25) Height>=74.5 118  50554.610 205.8305 *
##        13) Age>=28.6 141  33879.870 213.1489 *
##    7) Height>=76.5 74  24914.660 221.0676
##      14) Age< 25.37 12  3018.000 206.0000 *
##      15) Age>=25.37 62  18644.980 223.9839 *
```

The output contains rich information. `split` indicates the decision criterion; `n` is the number of observations that fall in this segment; `yval` is the predicted value if the test data falls into a segment.

10.6.3 Visualizing Decision Trees

A fancy way of drawing the `rpart` decision tree is by the `rpart.plot()` function under `rpart.plot` package (Fig. 10.13).

```
# install.packages("rpart.plot")
library(rpart.plot)
rpart.plot(mlb.rpart, digits=3)
```

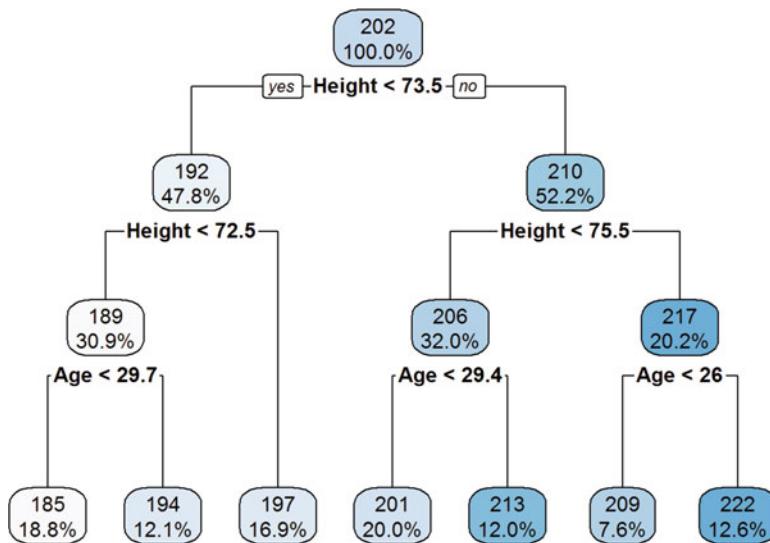


Fig. 10.13 MLB decision tree partitioning

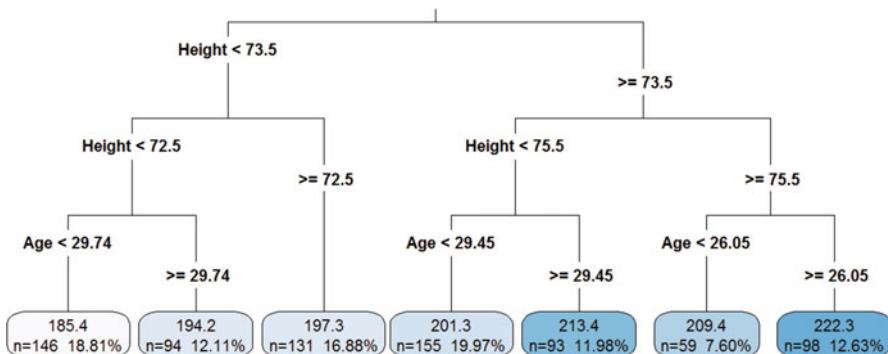


Fig. 10.14 Expanding the decision tree by specifying significant digits, drawing separate split labels for the left and right directions, displaying the number and percentage of observations in the node, and positioning the leaf nodes at the bottom of the graph

A more detailed graph can be obtained by specifying more options in the function call (Fig. 10.14).

```
rpart.plot(mlb.rpart, digits = 4, fallen.leaves = T, type=3, extra=101)
```

We may also use a more elaborate tree plot from package `rattle` to observe the order and rules of splits (Fig. 10.15).

```
library(rattle)
fancyRpartPlot(mlb.rpart, cex = 0.8)
```

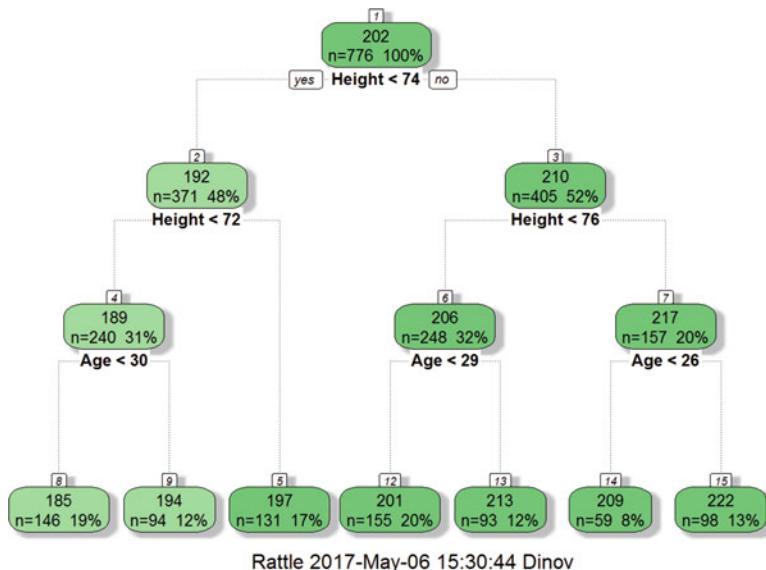


Fig. 10.15 An alternative plot of the MLB decision tree

10.6.4 Step 4: Evaluating Model Performance

Let's make predictions with the regression tree model using the `predict()` command.

```
mlb.p<-predict(mlb.rpart, mlb_test)
summary(mlb.p)

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 185.4   194.2   201.3   201.8   213.4   222.3

summary(mlb_test$Weight)

##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## 150.0   190.0   200.0   202.4   215.0   260.0
```

Comparing the five-number statistics for the predicted and true `Weight`, we can see that the model cannot precisely identify extreme cases such as the maximum. However, within the IQR, the predictions are relatively accurate. Correlation could be used to measure the correspondence of two equal length numeric variables. Let's use `cor()` to examine the prediction accuracy.

```
cor(mlb.p, mlb_test$Weight)
## [1] 0.4940257
```

The predicted values are moderately correlated with the true values.

10.6.5 Measuring Performance with Mean Absolute Error

To measure the distance between predicted value and the true value, we can use a measurement called mean absolute error (MAE) defined by the formula:

$$MAE = \frac{1}{n} \sum_{i=1}^n |pred_i - obs_i|,$$

where the $pred_i$ is the i th predicted value and obs_i is the i th observed value. Let's make a corresponding MAE function in R and evaluate our model performance.

```
MAE<-function(obs, pred){
  mean(abs(obs-pred))
}
MAE(mlb_test$Weight, mlb.p)
## [1] 14.97519
```

This implies that on average, the difference between the predicted value and the observed value is 14.975. Considering that the `Weight` variable in our test dataset ranges from 150 to 260, the model is reasonable.

What if we used a more the most primitive method for prediction – the test data **mean**?

```
mean(mlb_test$Weight)
## [1] 202.3643
MAE(mlb_test$Weight, 202.3643)
## [1] 17.11207
```

This shows that the regression decision tree is better than using the mean to predict every observation in the test dataset. However, it is not dramatically better. There might be room for improvement.

10.6.6 Step 5: Improving Model Performance

To improve the performance of our decision tree, we are going to use a model tree instead of a regression tree. We can use the `M5P()` function, under the package `RWeka`, which implements the M5 algorithm. This function uses similar syntax as `rpart()`.

```
m<-M5P(dv~iv, data=mydata)
```

```
#install.packages("RWeka")

# Sometimes RWeka installations may be off a bit, see:
# http://stackoverflow.com/questions/41878226/using-rweka-m5p-in-rstudio-yields-java-lang-noclassdeffounderror-no-uib-cipr-ma

Sys.getenv("WEKA_HOME") # where does it point to? Maybe some obscure path?

## [1] ""
## if yes, correct the variable:
Sys.setenv(WEKA_HOME="C:\\MY\\PATH\\WEKA_WPM")
Library(RWeka)
# WPM("list-packages", "installed")

mlb.m5<-M5P(Weight~Height+Age, data=mlb_train)
mlb.m5

## M5 pruned model tree:
## (using smoothed Linear models)
## LM1 (776/82.097%)
##
## LM num: 1
## Weight =
## 4.9957 * Height
## + 1.0629 * Age
## - 197.0898
##
## Number of Rules : 1
```

Instead of using segment averages to predict the player's weight, this model uses a linear regression (LM1) as the terminal node. In some datasets with more variables, M5P could yield different linear models at each terminal node.

```
summary(mlb.m5)

## === Summary ===
##
## Correlation coefficient          0.571
## Mean absolute error            13.3503
## Root mean squared error        17.1622
## Relative absolute error        80.3144 %
## Root relative squared error   82.0969 %
## Total Number of Instances      776

mlb.p.m5<-predict(mlb.m5, mlb_test)
summary(mlb.p.m5)

##      Min. 1st Qu. Median  Mean 3rd Qu.  Max.
## 166.1   193.5  201.7  202.0  209.7  247.8

cor(mlb.p.m5, mlb_test$Weight)
## [1] 0.5500171

MAE(mlb_test$Weight, mlb.p.m5)
## [1] 14.07716
```

`summary(mlb.m5)` reports some rough diagnostic statistics. We can see that the correlation and MAE for this model are better than the previous `rpart()` model.

10.7 Practice Problem: Heart Attack Data

Let's go back to the heart attack dataset (CaseStudy12_AdultsHeartAttack_Data.csv) and practice this approach.

```
heart_attack<-read.csv("https://umich.instructure.com/files/1644953/download?download_frd=1", stringsAsFactors = F)
str(heart_attack)

## 'data.frame': 150 obs. of 8 variables:
## $ Patient : int 1 2 3 4 5 6 7 8 9 10 ...
## $ DIAGNOSIS: int 41041 41041 41091 41081 41091 41091 41091 41091 41041
41041 ...
## $ SEX      : chr "F" "F" "F" "F" ...
## $ DRG      : int 122 122 122 122 122 121 121 121 121 123 ...
## $ DIED     : int 0 0 0 0 0 0 0 0 1 ...
## $ CHARGES  : chr "4752" "3941" "3657" "1481" ...
## $ LOS      : int 10 6 5 2 1 9 15 15 2 1 ...
## $ AGE      : int 79 34 76 80 55 84 84 70 76 65 ...
```

First, we need to convert the CHARGES (independent variable) to numerical form. NA's are created, so let's keep only the complete cases as mentioned in the beginning of this Chapter. Also, let's create a gender variable as an indicator for female patients using `ifelse()` and delete the previous SEX column.

```
heart_attack$CHARGES<-as.numeric(heart_attack$CHARGES)
heart_attack<-heart_attack[complete.cases(heart_attack), ]
heart_attack$gender<-ifelse(heart_attack$SEX=="F", 1, 0)
heart_attack<-heart_attack[, -3]
```

Next, we can build a model tree using `M5P()` with all the features in the model. As usual, we need to separate the `heart_attack` data into training and test datasets (e.g., use the 75–25% random split).

Using the model to predict CHARGES in the test dataset, we can obtain the following correlation and MAE.

```
## [1] 0.5616003
## [1] 3193.502
```

We can see that the predicted values and observed values are strongly correlated. In terms of MAE, it may seem very large at first glance.

```
range(ha_test$CHARGES)
## [1] 701 17137
# 17137-701
# 3193.502/16436
```

However, the test data itself has a wide range, and the MAE is within 20% of the range. With only 148 observations, the model represents a fairly good prediction of the expected hospital stay charges. Try to reproduce these results and also test the same techniques to other data from the list of our Case-Studies.

10.8 Assignments: 10. Forecasting Numeric Data Using Regression Models

Use the Quality of Life data (Case06_QoL_Symptom_ChronicIllness) to fit several different Multiple Linear Regression models predicting clinically relevant outcomes, e.g., Chronic Disease Score.

- Summarize and visualize data using `summary`, `str`, `pairs.panels`, `ggplot`.
- Report correlation for numeric data and try to visualize it (e.g., heatmap, `pairs` plot, etc.)
- Examine potential dependences of the predictors and the dependent response variables.
- Fit a couple of Multiple Linear Regression models, report the results, and explain the summary, residuals, effect-size coefficients, and the coefficient of determination, R^2 .
- Draw model diagnostic plots, at least QQ plot, residuals plot and leverage plot (half norm plot).
- Report results in terms of the model.
- Predict outcomes for new data.
- Try to improve the model performance using `step` function based on AIC and BIC.
- Fit a regression tree model and compare with OLS model.
- Try to use RWeka : :M5P to improve the model.

References

- Fahrmeir, L, Kneib, T, Lang, S, Marx, B. (2013) *Regression: Models, Methods and Applications*, Springer Science & Business Media, ISBN 3642343333, 9783642343339.
- Hyndman, RJ, Athanasopoulos, G. (2014) *Forecasting: principles and practice*, OTexts, ISBN 0987507109, 9780987507105.
- Zhao, Y. (2012) *R and Data Mining: Examples and Case Studies*, Academic Press, ISBN 012397271X, 9780123972712.
- http://wiki.socr.umich.edu/index.php/EBook#Chapter_X:_Correlation_and_Regression.

Chapter 11

Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines



In this Chapter, we are going to cover two very powerful machine-learning algorithms. These techniques have complex mathematical formulations; however, efficient algorithms and reliable software packages have been developed to utilize them for various practical applications. We will (1) describe Neural Networks as analogues of biological neurons; (2) develop hands-on a neural net that can be trained to compute the square-root function; (3) describe support vector machine (SVM) classification; and (4) complete several case-studies, including optical character recognition (OCR), the Iris flowers, Google Trends and the Stock Market, and Quality of Life in chronic disease.

Later, in Chap. 23, we will provide more details and additional examples of deep neural network learning. For now, let's start by exploring the *magic* inside the machine learning black box.

11.1 Understanding Neural Networks

11.1.1 *From Biological to Artificial Neurons*

An Artificial Neural Network (ANN) model mimics the biological brain response to multisource inputs, e.g., sensory-motor stimuli. ANNs simulate the brain using networks of interconnected neuron cells to create massively parallel processors. Of course, ANNs use networks of artificial nodes, not brain cells, to train data.

When we have three signals (or inputs) x_1 , x_2 and x_3 , the first step is weighting the features (w 's) according to their importance. Then, the weighted signals are summed by the “neuron cell” and this sum is passed on according to an **activation function** denoted by f . The last step is generating an output y at the end of the process. A typical output will have the following mathematical relationship to the inputs.

$$y(x) = f\left(\sum_{i=1}^n w_i x_i\right).$$

There are three important components for building a neural network:

- **Activation function:** transforms weighted and summed inputs to an output.
- **Network topology:** describes the number of “neuron cells”, the number of layers and manner in which the cells are connected.
- **Training algorithm:** how to determine weights w_i .

Let's look at each of these components one by one.

11.1.2 Activation Functions

One of the functions, known as threshold activation function, triggers an output signal once a specified input threshold has been attained (Fig. 11.1).

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}.$$

This is the simplest form for activation functions. It is rarely used in real world situations. The most commonly used alternative is the sigmoid activation function, where $f(x) = \frac{1}{1+e^{-x}}$. Here, e is Euler's natural number, which is also the base of the natural logarithm function. The output signal is no longer binary but can be any real number ranged from 0 to 1 (Fig. 11.2).

Fig. 11.1 An example of a hard threshold activation function, $f(x)$

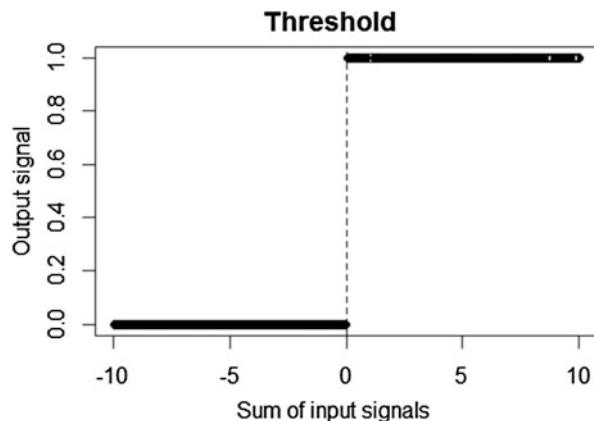


Fig. 11.2 The S-shaped sigmoid activation function

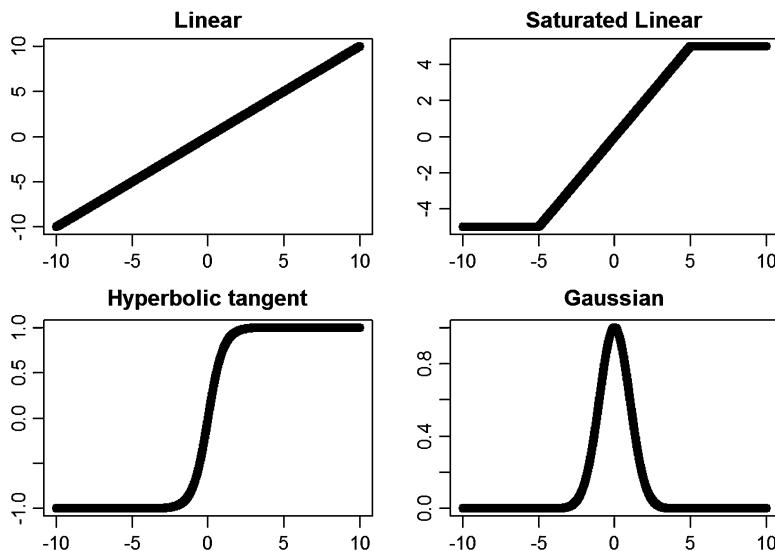
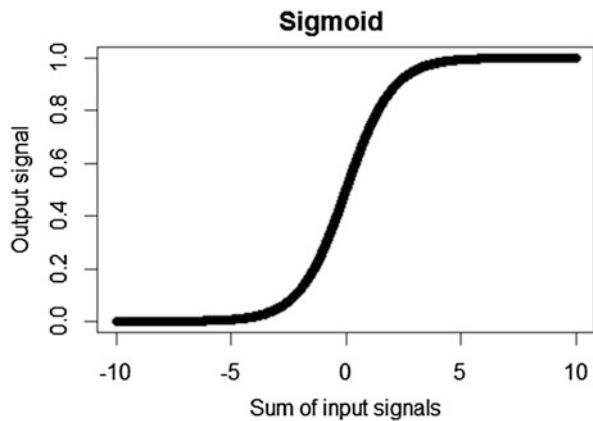


Fig. 11.3 Alternative types of activation functions

Other activation functions might also be useful, Fig. 11.3.

Basically, we can chose a proper activation function based on the corresponding codomain of the function. For example, with hyperbolic tangent activation function, we can only have outputs ranging from -1 to 1 regardless of the input. With linear function we can go from $-\infty$ to $+\infty$. Our Gaussian activation function will give us a model called *Radial Basis Function* network (Fig. 11.3).

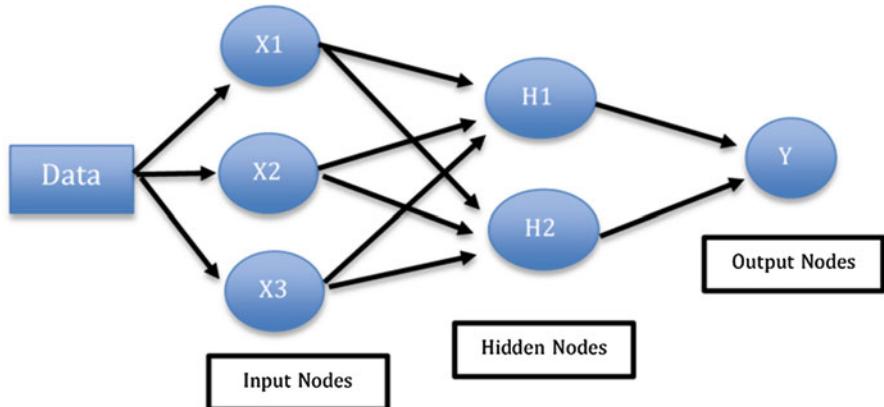


Fig. 11.4 A schematic of a two-layer neural network

11.1.3 Network Topology

Number of layers: The x 's or features in the dataset are called **input nodes** while the predicted values are called the **output nodes**. Multilayer networks include multiple hidden layers. Figure 11.4 shows a two layer neural network.

When we have multiple layers, the information flow could be complicated.

11.1.4 The Direction of Information Travel

The arrows in the last graph (with multiple layers) suggest a feed forward network. In such a network, we can also have multiple outcomes modeled simultaneously (Fig. 11.5).

Alternatively, in a recurrent network (feedback network), information can also travel backwards in loops (or delay). This is illustrated in Fig. 11.6, where the short-term memory increases the power of recurrent networks dramatically. However, in practice, recurrent networks are rarely used.

11.1.5 The Number of Nodes in Each Layer

The number of input nodes and output nodes are predetermined by the dataset and the predictive variables. The number we can specify determines the hidden nodes in the model. To simplify the model, our goal is to add the least number of hidden nodes possible when the model performance remains reasonable.

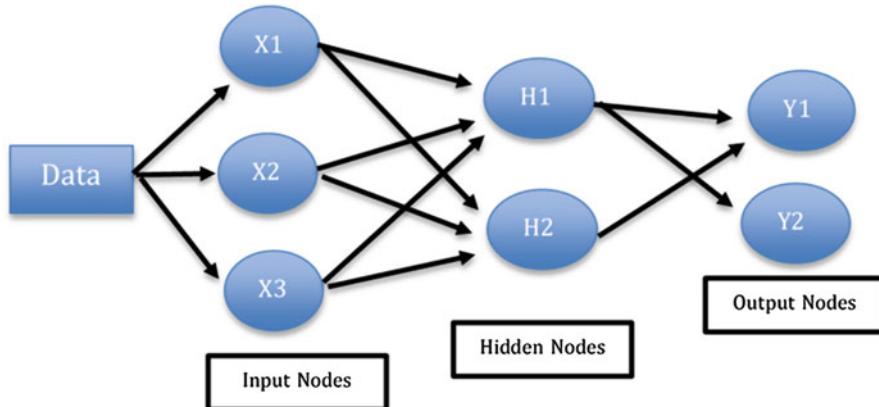


Fig. 11.5 A multi-output neural network example

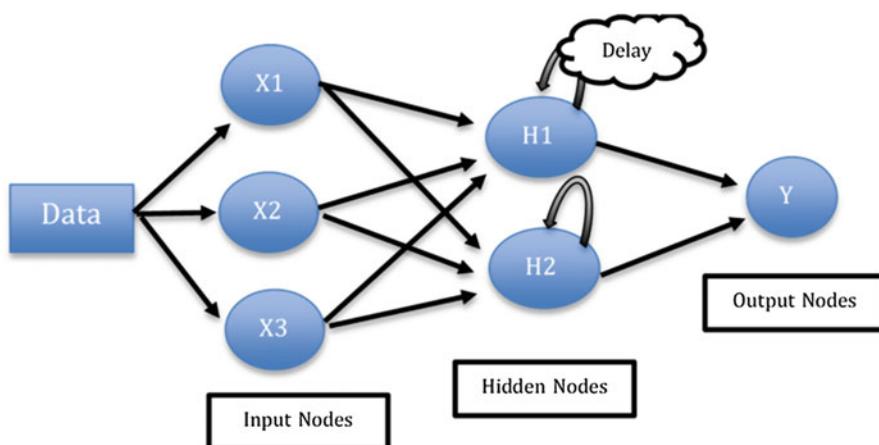


Fig. 11.6 A schematic of a delay (feedback) neural network

11.1.6 Training Neural Networks with Backpropagation

This algorithm could determine the weights in the model using the strategy of back-propagating errors. First, we assign random weights (but all weights must be non-trivial). For example, we can use normal distribution, or any other random process, to assign initial weights. Then we adjust the weights iteratively by repeating the process until certain convergence or stopping criterion is met. Each iteration contains two phases.

- Forward phase: from input layer to output layer using current weights. Outputs are produced at the end of this phase, and
- Backward phase: compare the outputs and true target values. If the difference is significant, we change the weights and go through the forward phase, again.

In the end, we pick a set of weights, which correspond to the least total error, to be the final weights in our network.

11.2 Case Study 1: Google Trends and the Stock Market: Regression

11.2.1 Step 1: Collecting Data

In this case study, we are going to use the Google trends and stock market dataset. A doc file with the meta-data and the CSV data are available on the Case-Studies Canvas Site. These daily data (between 2008 and 2009) can be used to examine the associations between Google search trends and the daily marker index - Dow Jones Industrial Average.

Variables

- **Index:** Time Index of the Observation
- **Date:** Date of the observation (Format: YYYY-MM-DD)
- **Unemployment:** The Google Unemployment Index tracks queries related to "unemployment, social, social security, unemployment benefits" and so on.
- **Rental:** The Google Rental Index tracks queries related to "rent, apartments, for rent, rentals," etc. **RealEstate:** The Google Real Estate Index tracks queries related to "real estate, mortgage, rent, apartments" and so on.
- **Mortgage:** The Google Mortgage Index tracks queries related to "mortgage, calculator, mortgage calculator, mortgage rates".
- **Jobs:** The Google Jobs Index tracks queries related to "jobs, city, job, resume, career, monster" and so forth.
- **Investing:** The Google Investing Index tracks queries related to "stock, finance, capital, yahoo finance, stocks", etc.
- **DJI_Index:** The Dow Jones Industrial (DJI) index. These data are interpolated from 5 records per week (Dow Jones stocks are traded on week-days only) to 7 days per week to match the constant 7-day records of the Google-Trends data.
- **StdDJI:** The standardized-DJI Index computed by: $\text{StdDJI} = 3 + (\text{DJI} - 11,091) / 1,501$, where $m = 11,091$ and $s = 1,501$ are the approximate mean and standard-deviation of the DJI for the period (2005–2011).

- **30-Day Moving Average Data Columns:** The 8 variables below are the 30-day moving averages of the 8 corresponding (raw) variables above.
 - *Unemployment30MA, Rental30MA, RealEstate30MA, Mortgage30MA, Jobs30MA, Investing30MA, DJI_Index30MA, StdDJI_30MA.*
- **180-Day Moving Average Data Columns:** The 8 variables below are the 180-day moving averages of the 8 corresponding (raw) variables.
 - *Unemployment180MA, Rental180MA, RealEstate180MA, Mortgage180MA, Jobs180MA, Investing180MA, DJI_Index180MA, StdDJI_180MA.*

Here we use the `RealEstate` as our dependent variable. Let's see if the Google Real Estate Index could be predicted by other variables in the dataset.

11.2.2 Step 2: Exploring and Preparing the Data

First, we need to load the dataset into R.

```
google<-read.csv("https://umich.instructure.com/files/416274/download?downLo
ad_frd=1", stringsAsFactors = F)
```

Let's delete the first two columns, since the only goal is to predict Google Real Estate Index with other indexes and DJI.

```
google<-google[, -c(1, 2)]
str(google)

## 'data.frame':    731 obs. of  24 variables:
## $ Unemployment      : num  1.54 1.56 1.59 1.62 1.64 1.64 1.71 1.85 1.82
1.78 ...
## $ Rental            : num  0.88 0.9 0.92 0.92 0.94 0.96 0.99 1.02 1.02 1
.01 ...
## $ RealEstate        : num  0.79 0.81 0.82 0.82 0.83 0.84 0.86 0.89 0.89
0.89 ...
## $ Mortgage          : num  1 1.05 1.07 1.08 1.1 1.11 1.15 1.22 1.23 1.24
...
## $ Jobs              : num  0.99 1.05 1.1 1.14 1.17 1.2 1.3 1.41 1.43 1.4
4 ...
## $ Investing         : num  0.92 0.94 0.96 0.98 0.99 0.99 1.02 1.09 1.1 1
.1 ...
## $ DJI_Index         : num  13044 13044 13057 12800 12827 ...
## $ StdDJI            : num  4.3 4.3 4.31 4.14 4.16 4.16 4.16 4 4.1 4.17 .
..
## $ Unemployment_30MA : num  1.37 1.37 1.38 1.38 1.39 1.4 1.4 1.42 1.43 1.
44 ...
## $ Rental_30MA       : num  0.72 0.72 0.73 0.73 0.74 0.75 0.76 0.77 0.78
0.79 ...
## $ RealEstate_30MA   : num  0.67 0.67 0.68 0.68 0.68 0.69 0.7 0.7 0.71 0.
72 ...
## $ Mortgage_30MA     : num  0.98 0.97 0.97 0.97 0.98 0.98 0.98 0.99 0.99
```

```

1 ...
## $ Jobs_30MA      : num  1.06 1.06 1.05 1.05 1.05 1.05 1.05 1.06 1.07
1.08 ...
## $ Investing_30MA : num  0.99 0.98 0.98 0.98 0.98 0.97 0.97 0.97 0.98
0.99 ...
## $ DJI_Index_30MA  : num  13405 13396 13390 13368 13342 ...
## $ StdDJI_30MA    : num  4.54 4.54 4.53 4.52 4.5 4.48 4.46 4.44 4.41 4
4 ...
## $ Unemployment_180MA: num  1.44 1.44 1.44 1.44 1.44 1.44 1.44 1.44 1.44
1.44 ...
## $ Rental_180MA    : num  0.87 0.87 0.87 0.87 0.87 0.87 0.86 0.86 0.86
0.86 ...
## $ RealEstate_180MA : num  0.89 0.89 0.88 0.88 0.88 0.88 0.88 0.88 0.88
0.87 ...
## $ Mortgage_180MA   : num  1.18 1.18 1.18 1.18 1.17 1.17 1.17 1.17 1.17
1.17 ...
## $ Jobs_180MA      : num  1.24 1.24 1.24 1.24 1.24 1.24 1.24 1.24 1.24
1.24 ...
## $ Investing_180MA  : num  1.04 1.04 1.04 1.04 1.04 1.04 1.04 1.04 1.04
1.04 ...
## $ DJI_Index_180MA  : num  13493 13492 13489 13486 13482 ...
## $ StdDJI_180MA    : num  4.6 4.6 4.6 4.6 4.59 4.59 4.59 4.58 4.58 4.58
...

```

As we can see from the structure of the data, these indices and DJI have different ranges. We should rescale the data. In Chap. 6, we learned that normalizing these features using our own `normalize()` function provides one solution. We can use `lapply()` to apply the `normalize()` function to each column.

```

normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
google_norm<-as.data.frame(lapply(google, normalize))
summary(google_norm$RealEstate)

##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 0.0000 0.4615 0.6731 0.6292 0.8077 1.0000

```

Looks like all the vectors are normalized into the $[0, 1]$ range.

The next step would be to split the `google` dataset into training and testing subsets. This time we will use the `sample()` and `floor()` function to separate the training and testing sets. `sample()` is a function to create a set of indicators for row numbers. We can subset the original dataset with random rows using these indicators. `floor()` takes a number x and returns the closest integer to x

```
sample(row, size)
```

- `row`: rows in the dataset that you want to select from. If you want to select all of the rows, you can use `nrow(data)` or `1 : nrow(data)`(single number or vector).
- `size`: how many rows you want for your subset.

```
sub<-sample(nrow(google_norm), floor(nrow(google_norm)*0.75))
google_train<-google_norm[sub, ]
google_test<-google_norm[-sub, ]
```

We are good to go and can move forward to the model training phase.

11.2.3 Step 3: Training a Model on the Data

Here, we use the function `neuralnet()` in package `neuralnet`. `neuralnet` returns a NN object containing:

- call: the matched call.
- response: extracted from the data argument.
- covariate: the variables extracted from the data argument.
- model.list: a list containing the covariates and the response variables extracted from the formula argument.
- err.fct and act.fct: the error and activation functions.
- net.result: a list containing the overall result of the neural network for every repetition.
- weights: a list containing the fitted weights of the neural network for every repetition.
- result.matrix: a matrix containing the reached threshold, needed steps, error, AIC, BIC, and weights for every repetition. Each column represents one repetition.

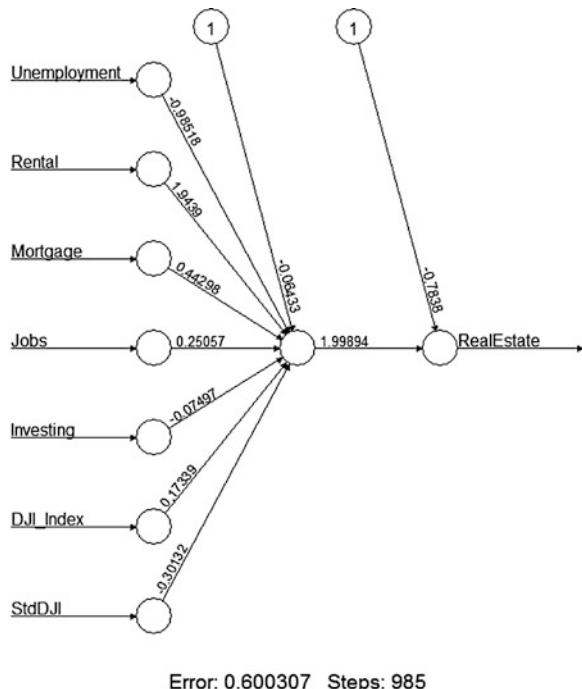
```
m<-neuralnet(target~predictors, data=mydata, hidden=1),
where:
```

- target: variable we want to predict.
- predictors: predictors we want to use. Note that we cannot use `"."` to denote all the variables in this function. We have to add all predictors one by one to the model.
- data: training dataset.
- hidden: number of hidden nodes that we want to use in the model. By default, it is set to one.

```
# install.packages("neuralnet")
library(neuralnet)
google_model<-neuralnet(RealEstate~Unemployment+Rental+Mortgage+Jobs+Investing+DJI_Index+StdDJI, data=google_train)
plot(google_model)
```

Figure 11.7 shows that we have only one hidden node. Error stands for the sum of squared errors and Steps is how many iterations the model had to go through. These outputs could be different when you run the exact same code because the weights are stochastically generated.

Fig. 11.7 A simple neural network predicting the real estate prices using Google market data



11.2.4 Step 4: Evaluating Model Performance

Similar to the `predict()` function that we have mentioned in previous Chapters, `compute()` is an alternative method that helps with ANN model predictions.

```
p<-compute(m, test)
```

- `m`: a trained neural networks model.
- `test`: the test dataset. This dataset should only contain the same type of predictors in the neural network model.

In our model, we picked `Unemployment`, `Rental`, `Mortgage`, `Jobs`, `Investing`, `DJI_Index`, `StdDJI` as our predictors. So, we need to find these corresponding column numbers in the test dataset (1, 2, 4, 5, 6, 7, 8, respectively).

```
google_pred<-compute(google_model, google_test[, c(1:2, 4:8)])
pred_results<-google_pred$net.result
cor(pred_results, google_test$RealEstate)
## [,1]
## [1,] 0.9653369986
```

As mentioned in Chap. 9, we can still use the correlation between predicted results and observed Real Estate Index to evaluate the data. A correlation over 0.96 is very good for real world datasets. Could this still be improved?

11.2.5 Step 5: Improving Model Performance

This time we will include four hidden nodes in the model. Let's see what results we can get from this more elaborate ANN model (Fig. 11.8).

```
google_model2<-neuralnet(RealEstate~Unemployment+Rental+Mortgage+Jobs+Investing+DJL_Index+StdDJL, data=google_train, hidden = 4)
plot(google_model2)
```

Although the graph looks complicated, we have smaller Error, or sum of squared errors. Neural net models may be used for *classification* and *regression*, which we will see in the next part. Let's first try regression tree modeling.

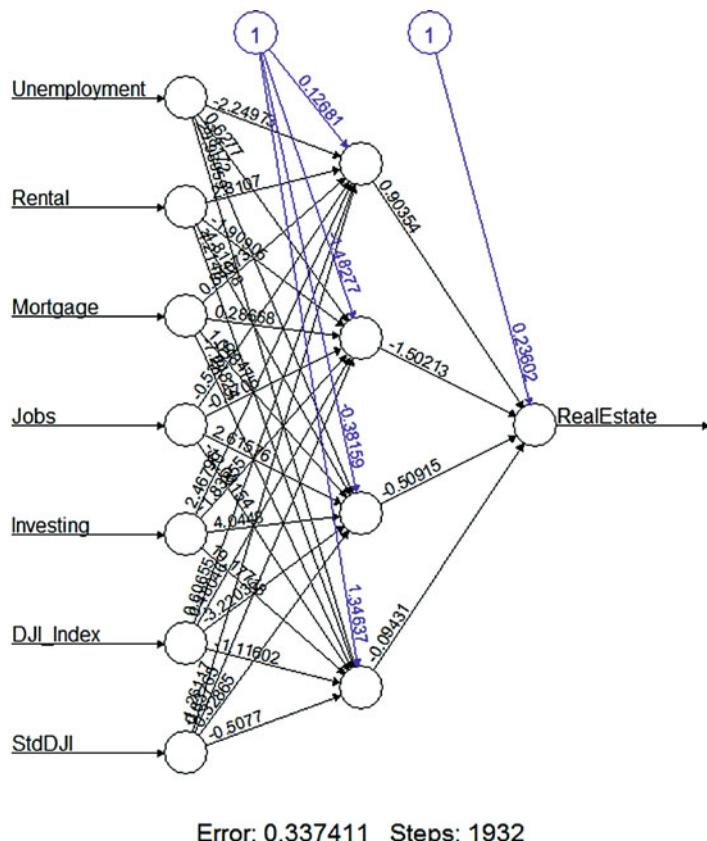


Fig. 11.8 A more elaborate neural network showing decreased prediction error, compare to Fig. 11.7

```
google_pred2<-compute(google_model2, google_test[, c(1:2, 4:8)])
pred_results2<-google_pred2$net.result
cor(pred_results2, google_test$RealEstate)

## [,1]
## [1,] 0.9869109731
```

We get an even higher correlation. This is almost an ideal result! The predicted and observed indices have a very strong linear relationship. Nevertheless, too many hidden nodes might even decrease the correlation between predicted and true values, which will be examined in the practice problems later in this Chapter.

11.2.6 Step 6: Adding Additional Layers

We observe an even lower Error by using three hidden layers with numbers of nodes 4,3,3 within each, respectively.

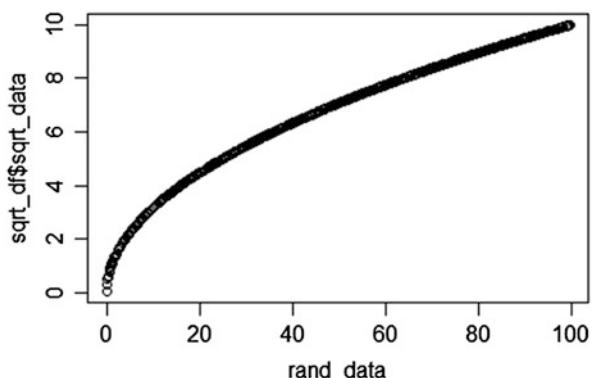
```
google_model2<-neuralnet(RealEstate~Unemployment+Rental+Mortgage+Jobs+Investing+DJI_Index+StdDJI, data=google_train, hidden = c(4,3,3))
google_pred2<-compute(google_model2, google_test[, c(1:2, 4:8)])
pred_results2<-google_pred2$net.result
cor(pred_results2, google_test$RealEstate)

## [,1]
## [1,] 0.9853727545
```

11.3 Simple NN Demo: Learning to Compute $\sqrt{\cdot}$

This simple example demonstrates the foundation of the neural network prediction of a basic mathematical function (square-root): $\sqrt{\cdot} : \mathfrak{R}^+ \rightarrow \mathfrak{R}^+$ (Fig. 11.9).

Fig. 11.9 The square-root function evaluated at random Uniform(0,100) values



```

# generate random training data: 1,000 |X_i|, where X_i ~ Uniform (0,10) or
# perhaps ~ N(0,1)
rand_data <- abs(runif(1000, 0, 100))

# create a 2 column data-frame (and_data, sqrt_data)
sqrt_df <- data.frame(rand_data, sqrt_data=sqrt(rand_data))
plot(rand_data, sqrt_df$sqrt_data)

# Train the neural net
net.sqrt <- neuralnet(sqrt_data ~ rand_data, sqrt_df, hidden=10,
threshold=0.01)

# report the NN
# print(net.sqrt)

# generate testing data seq(from=0.1, to=N, step=0.1)
N <- 200.0 # out of range [100: 200] is also included in the testing!
test_data <- seq(0, N, 0.1); test_data_sqrt <- sqrt(test_data)

# try to predict the square-root values using 10 hidden nodes
# Compute or predict for test data, test_data
pred_sqrt <- compute(net.sqrt, test_data)$net.result

# compute uses the trained neural net (net.sqrt),
# to estimate the square-roots of the testing data

# compare real (test_data_sqrt) and NN-predicted (pred_sqrt) square
# roots of test_data
plot(pred_sqrt, test_data_sqrt, xlim=c(0, 12), ylim=c(0, 12));
abline(0,1, col="red", lty=2)
legend("bottomright", c("Pred vs. Actual SQRT", "Pred=Actual Line"),
cex=0.8, lty=c(1,2), lwd=c(2,2), col=c("black", "red"))

compare_df <- data.frame(pred_sqrt, test_data_sqrt); # compare_df

plot(test_data, test_data_sqrt)
lines(test_data, pred_sqrt, pch=22, col="red", lty=2)
legend("bottomright", c("Actual SQRT", "Predicted SQRT"), lty=c(1,2), lwd=c(2,2),
col=c("black", "red"))

```

We observe that the NN, `net.sqrt` actually learns and predicts the complex square root function with good accuracy, Figs 11.10 and 11.11. Of course, individual results may vary, as we randomly generate the training data (`rand_data`) and due to the stochastic construction of the ANN.

Fig. 11.10 Test data validation of the neural network predicting the behavior of the square-root function

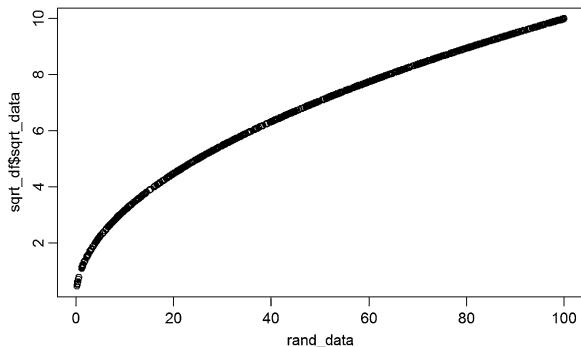
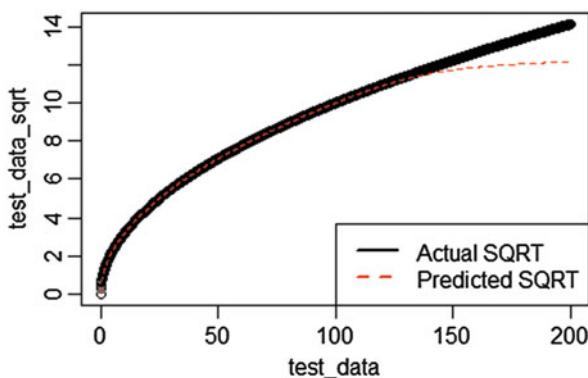


Fig. 11.11 Plots illustrating the agreement of the NN-predicted and the analytical square-root function



11.4 Case Study 2: Google Trends and the Stock Market – Classification

In practice, ANN models are also useful as classifiers. Let's demonstrate this by using again the Stock Market data. We will binarize the samples according to their RealEstate values. For those higher than the 75%, we will label them 0; For those lower than the 25%, we will label them 2; all others will be labeled 1. Even in the classification setting, the response still must be numeric.

```
google_class = google_norm
id1 = which(google_class$RealEstate>quantile(google_class$RealEstate,0.75))
id2 = which(google_class$RealEstate<quantile(google_class$RealEstate,0.25))
id3 = setdiff(1:nrow(google_class), union(id1,id2))
google_class$RealEstate[id1]=0
google_class$RealEstate[id2]=1
google_class$RealEstate[id3]=2
summary(as.factor(google_class$RealEstate))

##    0    1    2
## 179 178 374
```

Here, we divide the data to training and testing sets. We need three more column indicators that correspond to the three derived RealEstate labels.

```

set.seed(2017)
train = sample(1:nrow(google_class), 0.7*nrow(google_class))
google_tr = google_class[train,]
google_ts = google_class[-train,]
train_x = google_tr[,c(1:2,4:8)]
train_y = google_tr[,3]
colnames(train_x)

## [1] "Unemployment" "Rental"      "Mortgage"     "Jobs"
## [5] "Investing"     "DJI_Index"   "StdDJI"

test_x = google_ts[,c(1:2,4:8)]
test_y = google_ts[3]
train_y_ind = model.matrix(~factor(train_y)-1)
colnames(train_y_ind) = c("High", "Median", "Low")
train = cbind(train_x, train_y_ind)

```

We use non-linear output and display every 2,000 iterations.

```

nn_single = neuralnet(High+Median+Low~Unemployment+Rental+Mortgage+Jobs+Investing+DJI_Index+StdDJI,
  data = train,
  hidden=4,
  linear.output=FALSE,
  lifespan='full', lifespan.step=2000)
## hidden: 4    thresh:0.01    rep:1/1    steps:2000  min thresh: 0.13702015
48
## 4000  min thresh: 0.08524054094
## 6000  min thresh: 0.08524054094
## 8000  min thresh: 0.08524054094
## 10000 min thresh: 0.08524054094
...
## 40000 min thresh: 0.02427719823
## 42000 min thresh: 0.02158221449
## 44000 min thresh: 0.01831644589
## 46000 min thresh: 0.01682874388
## 48000 min thresh: 0.01572773551
## 50000 min thresh: 0.01311388938
## 52000 min thresh: 0.01241004281
## 54000 min thresh: 0.01131407008
## 55420 error: 7.01191  time: 19.33 secs

```

Below is the prediction function translating this model to generate forecasting results.

```

pred = function(nn, dat) {
  # compute uses the trained neural net (nn=nn_single), and
  # new testing data (dat=google_ts) to generate predictions (y_hat)
  # compute returns a list containing:
  # (1) neurons: a list of the neurons' output for each layer of the
  neural network, and
  # (2) net.result: a matrix containing the overall result of the
  neural network.
  yhat = compute(nn, dat)$net.result

```

```

# find the maximum in each row (1) in the net.result matrix
# to determine the first occurrence of a specific element in each row (1)
# we can use the apply function with which.max
yhat = apply(yhat, 1, which.max)-1
return(yhat)
}
mean(pred(nn_single, google_ts[,c(1:2,4:8)]) != as.factor(google_ts[,3]))
## [1] 0.03181818182

```

Now let's inspect the structure of the Neural Network.

```
plot(nn_single)
```

Similarly, we can change hidden to utilize multiple hidden layers. However, a more complicated model won't necessarily guarantee an improved performance.

```

nn_single = neuralNet(High+Median+Low~Unemployment+Rental+Mortgage+Jobs+Investing+DJI_Index+StdDJI,
  data = train,
  hidden=c(4,5),
  linear.output=FALSE,
  lfitesign='full', lfitesign.step=2000)

## hidden: 4, 5      thresh: 0.01      rep:1/1      steps:2000      min thresh: 0.307
##      4000      min thresh: 0.2875517033
##      6000      min thresh: 0.1383720887
##      8000      min thresh: 0.1115440575
##     10000      min thresh: 0.09233958192
##     12000      min thresh: 0.0766173347
##     14000      min thresh: 0.05763223509
##     16000      min thresh: 0.03417989426
##     18000      min thresh: 0.01473872843
##     20000      min thresh: 0.01101646653
##     20741      error: 7.00627      time: 11.3 secs

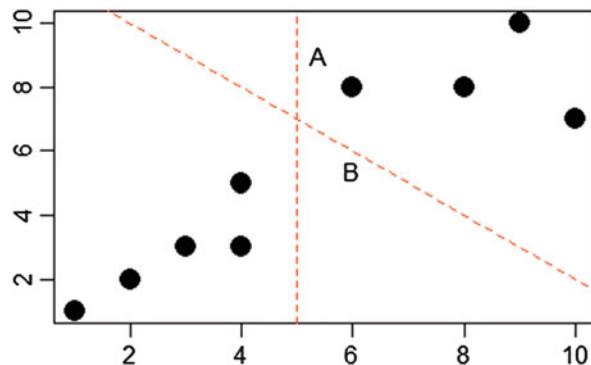
mean(pred(nn_single, google_ts[,c(1:2,4:8)]) != as.factor(google_ts[,3]))
## [1] 0.03181818182

```

11.5 Support Vector Machines (SVM)

Recall that the Lazy learning methods in Chap. 6 assigned class labels using geometrical distances of different features. In multidimensional spaces (multiple features), we can use spheres with centers determined by the training dataset. Then, we can assign labels to testing data according to their nearest spherical center. Let's see if we make a choose other hypersurfaces that may separate nD data and indicate a classification scheme.

Fig. 11.12 Schematic representation of linear-kernel SVM classification



11.5.1 Classification with Hyperplanes

The easiest shape would be a plane. Support Vector Machine (SVM) can use hyperplanes to separate data into several groups or classes. This is used for datasets that are linearly separable. Assume that we have only two features, will you use the A or B hyperplane to separate the data on Fig. 11.12? Perhaps even another hyperplane, C?

Finding the Maximum Margin

To answer the above question, we need to search for the **Maximum Margin Hyperplane (MMH)**. That is the hyperplane that creates the greatest separation between the two closest observations.

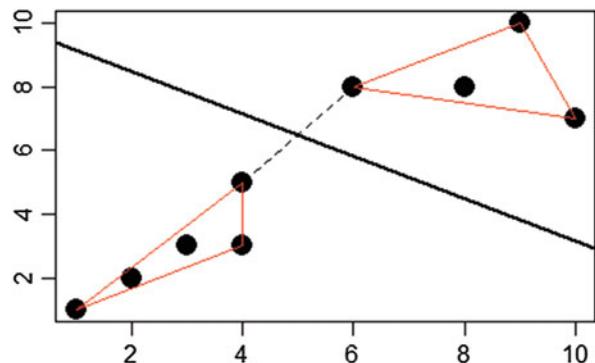
We define support vectors as the points from each class that are closest to the MMH. Each class must have at least one observation as a support vector.

Using support vectors alone is not sufficient for finding the MMH. Although tricky mathematical calculations are involved, the fundamental process is fairly simple. Let's look at linearly separable data and non-linearly separable data individually.

Linearly Separable Data

If the dataset is linearly separable, we can find the outer boundaries of our two groups of data points. These boundaries are called convex hull (red lines in the following graph). The MMH (black solid line) is the line that is perpendicular to the shortest line between the two convex hulls (Fig. 11.13).

Fig. 11.13 Convex hulls of the linearly separable groups of points



An alternative way would be picking two parallel planes that can separate the data into two groups while the distance between two planes is as far as possible.

To mathematically define a plane, we need to use vectors. In n -dimensional spaces planes could be expressed by the following equation:

$$\vec{w} \cdot \vec{x} + b = 0,$$

where \vec{w} (weights) and \vec{x} both have n coordinates, and b is a single number known as the *bias*.

To clarify this notation, let's look at the situation in a 3D space where we can express (embed) 2D Euclidean planes given a point $((x_o, y_o, z_o))$ and a normal-vector $((a, b, c))$ form. This is just a linear equation, where $d = -(ax_o + by_o + cz_o)$:

$$ax + by + cz + d = 0,$$

or equivalently

$$w_1x_1 + w_2x_2 + w_3x_3 + b = 0.$$

We can see that it is equivalent to the vector notation.

Using the vector notation, we can specify two hyperplanes as follows:

$$\vec{w} \cdot \vec{x} + b \geq +1$$

and

$$\vec{w} \cdot \vec{x} + b \leq -1.$$

We require that all class 1 observations fall above the first plane and all observations in the other class fall below the second plane.

The distance between two planes is calculated as:

$$\frac{2}{\|\vec{w}\|},$$

where $\|\vec{w}\|$ is the Euclidean norm. To maximize the distance, we need to minimize the Euclidean norm.

To sum up we are going to find $\min \frac{\|\vec{w}\|}{2}$ with the following constrain:

$$y_i(\vec{w} \cdot \vec{x} - b) \geq 1, \forall \vec{x}_i,$$

where \forall means “for all”.

For each nonlinear programming problem, called the **primal problem**, there is a related nonlinear programming problem, called the **Lagrangian dual problem**. Under certain assumptions for convexity and suitable constraints, the primal and dual problems have equal optimal objective values. Primal optimization problems are typically described as:

$$\min_x f(x)$$

subject to

$$\begin{cases} g_i(x) \leq 0 \\ h_j(x) = 0 \end{cases}.$$

The Lagrangian dual problem is defined as a parallel nonlinear programming problem:

$$\begin{array}{ll} \min_{u, v} & (\theta(u, v)) \\ \text{subject to} & u \geq 0, \end{array}$$

where

$$\theta(u, v) = \inf_x \left(f(x) + \sum_i u_i g_i(x) + \sum_j v_j h_j(x) \right).$$

Chapter 21 provides additional technical details about optimization duality.

Suppose the Lagrange primal is:

$$L_p = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i [y_i(w_0 + x_i^T w) - 1], \text{ where } \alpha_i \geq 0.$$

To optimize that objective function, we can set the partial derivatives equal to zero:

$$\frac{\partial}{\partial w} : w = \sum_{i=1}^n \alpha_i y_i x_i$$

$$\frac{\partial}{\partial b} : 0 = \sum_{i=1}^n \alpha_i y_i.$$

Substituting into the Lagrange primal, we obtain the Lagrange dual:

$$L_D = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \alpha_i \alpha'_i y_i y'_i x_i^T x'_i.$$

We maximize L_D subject to $\alpha_i \geq 0$ and $\sum_{i=1}^n \alpha_i y_i = 0$.

The Karush-Kuhn-Tucker optimization conditions suggest that we have $\hat{\alpha} [y_i(\hat{b} + x_i^T \hat{w}) - 1] = 0$.

Which implies that if $y_i \hat{f}(x_i) > 1$, then $\hat{\alpha}_i = 0$. The **support** of a function (f) is the smallest subset of the domain containing only arguments (x) which are not mapped to zero ($f(x) \neq 0$). In our case, the solution \hat{w} is defined in terms of a linear combination of the **support points**:

$$\hat{f}(x) = w^T x = \sum_{i=1}^n \alpha_i y_i x_i.$$

That's where the name of Support Vector Machines (SVM) comes from.

Non-linearly Separable Data

For non-linearly separable data, we need to use a small trick. We still use a plane, but allow some of the points to be misclassified into the wrong class. To penalize for that, we add a cost term after the Euclidean norm function that we need to minimize.

Therefore, the solution changes to:

$$\begin{aligned} \min \left(\frac{\|\vec{w}\|}{2} \right) + C \sum_{i=1}^n \xi_i \\ \text{s.t. } y_i (\vec{w} \cdot \vec{x} - b) \geq 1, \forall \vec{x}_i, \xi_i \geq 0, \end{aligned}$$

where C is the cost and ξ_i is the distance between the misclassified observation i and the plane.

We have Lagrange dual problem:

$$L_p = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i(b + x_i^T w) - (1 - \xi_i)] - \sum_{i=1}^n \gamma_i \xi_i,$$

where

$$\alpha_i, \gamma_i \geq 0.$$

Similar to what we did above for the separable case, we can use the derivatives of the primal problem to solve the dual problem.

Notice the inner product in the final expression. We can replace this inner product with a kernel function that maps the feature space into a higher dimensional space (e.g., using a polynomial kernel) or an infinite dimensional space (e.g., using a Gaussian kernel).

Using Kernels for Non-linear Spaces

An alternative way to solve for the non-linear separable is called the kernel trick. That is to add some dimensions (or features) to make these non-linear separable data to be separable in a higher dimensional space.

How can we do that? We transform our data using kernel functions. A general form for kernel functions would be:

$$K(\vec{x}_i, \vec{x}_j) = \phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$$

The kernel is a mapping of the data into another space.

The linear kernel would be the simplest one that is just the dot product of the features.

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j.$$

The polynomial kernel of degree d transforms the data by adding a simple non-linear transformation of the data.

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d.$$

The sigmoid kernel is very similar to neural network. It uses a sigmoid activation function.

$$K(\vec{x}_i, \vec{x}_j) = \tanh(k\vec{x}_i \cdot \vec{x}_j - \delta).$$

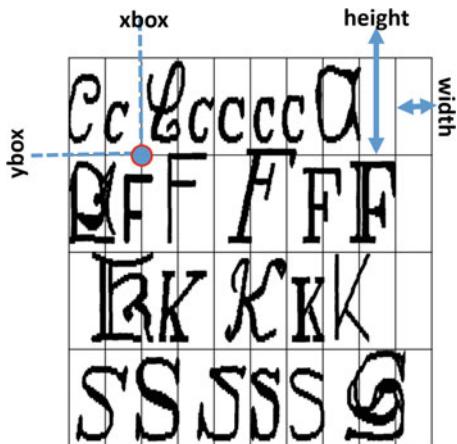
The Gaussian RBF kernel is similar to RBF neural network and is a good place to start investigating a dataset.

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(\frac{-\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right).$$

11.6 Case Study 3: Optical Character Recognition (OCR)

This example illustrates interpreting, processing and recognizing handwritten notes (text). Specifically, we will convert handwritten characters (unstructured image data) to printed text (typeset characters).

Fig. 11.14 Example of the preprocessed gridded handwritten letters



Protocol:

- Divide the image (typically optical image of handwritten notes on paper) into a fine grid where each cell contains one glyph (symbol, letter, number).
- Match the glyph in each cell to one of the possible characters in a dictionary.
- Combine individual characters together into words to reconstitute the digital representation of the optical image of the handwritten notes.

In this example, we use an optical document image (data) that has already been pre-partitioned into rectangular grid cells containing one character of the 26 English letters, A through Z.

The resulting gridded dataset is distributed by the UCI Machine Learning Data Repository. The dataset contains 20,000 examples of 26 English capital letters printed using 20 different randomly reshaped and morphed fonts (Fig. 11.14).

11.6.1 Step 1: Prepare and Explore the Data

```
# read in data and examine structure
hand_Letters <- read.csv("https://umich.instructure.com/files/2837863/download_frd=1", header = T)
str(hand_Letters)

## 'data.frame': 20000 obs. of 17 variables:
## $ Letter: Factor w/ 26 Levels "A","B","C","D",...: 20 9 4 14 7 19 2 1 10
## ...
## $ xbox  : int 2 5 4 7 2 4 4 1 2 11 ...
## $ ybox  : int 8 12 11 11 1 11 2 1 2 15 ...
## $ width : int 3 3 6 6 3 5 5 3 4 13 ...
## $ height: int 5 7 8 6 1 8 4 2 4 9 ...
```

```

## $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
## $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
## $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
## $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
## $ y2bar : int  6 4 6 6 9 6 2 6 2 ...
## $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
## $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
## $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
## $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
## $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
## $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
## $ yedgey: int  8 10 9 8 10 7 10 7 7 8 ...
# divide into training (3/4) and testing (1/4) data
hand_letters_train <- hand_letters[1:15000, ]
hand_letters_test  <- hand_letters[15001:20000, ]

```

11.6.2 Step 2: Training an SVM Model

We can specify `vanilladot` as a linear kernel, or alternatively:

- `rbfdot` Radial Basis kernel i.e., “Gaussian”
- `polydot` Polynomial kernel
- `tanhdot` Hyperbolic tangent kernel
- `laplacedot` Laplacian kernel
- `besseldot` Bessel kernel
- `anovadot` ANOVA RBF kernel
- `splinedot` Spline kernel
- `stringdot` String kernel

```

# begin by training a simple linear SVM
library(kernlab)
set.seed(123)
hand_letter_classifier <- ksvm(letter ~ ., data = hand_letters_train,
kernel = "vanilladot")

## Setting default kernel parameters

# look at basic information about the model
hand_letter_classifier

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
## parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 6618
##
```

```

## Objective Function Value : -13.2947 -19.6051 -20.8982 -5.6651 -7.2092 -31
.5151 -48.3253 -17.6236 -57.0476 -30.532 -15.7162 -31.49 -28.2706 -45.741 -1
1.7891 -33.3161 -28.2251 -16.5347 -13.2693 -30.88 -29.4259 -7.7099 -11.1685
-29.4289 -13.0857 -9.2631 -144.1105 -52.7747 -71.052 -109.7783 -158.3152 -51
.2839 -39.6499 -67.0061 -23.8637 -27.6083 -26.3461 -35.2626 -38.6346 -116.89
67 -173.8336 -214.2196 -20.7925 -10.3812 -53.1156 -12.228 -46.6132 -8.6867 -
18.9108 -11.0535 -94.5751 -26.5689 -224.0215 -70.5714 -8.3232 -4.5265 -132.5
431 -74.6876 -19.5742 -12.7352 -81.7894 -11.6983 -25.4835 -17.582 -23.934 -2
7.022 -50.7092 -10.9228 -4.3852 -13.7216 -3.8547 -3.5723 -8.419 -36.9773 -47
.1418 -172.6874 -42.457 -44.0342 -42.7695 -13.0527 -16.7534 -78.7849 -101.81
46 -32.1141 -30.3349 -104.0695 -32.1258 -24.6301 -32.6087 -17.0808 -5.1347 -
40.5505 -6.684 -16.2962 -56.364 -147.3669 -49.0907 -37.8334 -32.8068 -73.248
-127.7819 -10.5342 -5.2495 -11.9568 -30.1631 -135.5915 -51.521 -176.2669 -99
.0973 -10.295 -14.5906 -3.7822 -64.1452 -7.4813 -84.9109 -40.9146 -87.2437 -
66.8629 -69.9932 -20.5294 -12.7577 -7.0328 -22.9219 -12.3975 -223.9411 -29.9
969 -24.0552 -132.6252 -133.7033 -9.2959 -33.1873 -5.8016 -57.3392 -60.9046
-27.1766 -200.8554 -29.9334 -15.9359 -130.0183 -154.4587 -43.5779 -24.4852 -
135.7896 -74.1531 -303.5043 -131.4741 -149.5403 -30.4917 -29.8086 -47.3454 -
24.6204 -44.2792 -6.2064 -8.6708 -36.4412 -68.712 -179.7303 -44.7489 -84.860
8 -136.6786 -569.3398 -113.0779 -138.435 -303.8556 -32.8011 -60.4546 -139.35
25 -108.9841 -34.277 -64.9071 -38.6148 -7.5086 -204.222 -12.9572 -29.0252 -2
.0352 -5.9916 -14.3706 -21.5773 -57.0064 -19.6546 -178.0543 -19.812 -4.145 -
4.5318 -0.8101 -116.8649 -7.8269 -53.3445 -21.4812 -13.5066 -5.3881 -15.1061
-27.6061 -18.9239 -68.8104 -26.1223 -93.0231 -15.1693 -9.7999 -7.6137 -1.530
1 -84.9531 -5.4551 -93.187 -93.4153 -43.8334 -23.6706 -59.1468 -22.0933 -47.
8381 -219.9936 -39.5596 -47.2643 -34.0752 -20.2532 -11.239 -118.4152 -6.4126
-5.1846 -8.7272 -9.4584 -20.8522 -22.0878 -113.0806 -29.0912 -80.397 -29.620
6 -13.7422 -8.9416 -3.0785 -79.842 -6.1869 -13.9663 -63.3665 -93.2067 -11.55
93 -13.0449 -48.2558 -2.9343 -8.25 -76.4361 -33.5374 -109.112 -4.1731 -6.197
8 -1.2664 -84.1287 -18.3054 -7.2209 -45.5509 -3.3567 -16.8612 -60.5094 -43.9
956 -53.0592 -6.1407 -17.4499 -2.3741 -65.023 -102.1593 -103.4312 -23.1318 -
17.3394 -50.6654 -31.4407 -57.6065 -19.6857 -5.2667 -4.1767 -55.8445 -30.92
-57.2396 -30.1101 -7.611 -47.7711 -12.1616 -19.1572 -53.5364 -3.8024 -53.124
-225.6075 -12.6791 -11.5852 -16.6614 -9.7186 -65.824 -16.3897 -42.3931 -50.5
13 -24.752 -14.513 -40.495 -16.5124 -57.1813 -4.7974 -5.2949 -81.7477 -3.272
-6.3448 -1.1259 -114.3256 -22.3232 -339.8619 -31.0491 -31.3872 -4.9625 -82.4
936 -123.6225 -72.8463 -23.4836 -33.1608 -11.7133 -19.7607 -1.8599 -50.1148
-8.2868 -143.3592 -1.8508 -1.9699 -9.4175 -0.5202 -25.0654 -30.0489 -5.6248
## Training error : 0.129733

```

11.6.3 Step 3: Evaluating Model Performance

Let's assess the SVM prediction using the testing data.

```

# predictions on testing dataset
hand_letter_predictions<- predict(hand_letter_classifier, hand_letters_test)

head(hand_letter_predictions)

## [1] C U K U E I
## Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

table(hand_letter_predictions, hand_letters_test$letter)

##

```

```

## hand_letter_predictions A B C D E F G H I J K L
## A 191 0 1 0 0 0 0 0 0 1 3 0 0 0
## B 0 157 0 9 2 0 1 3 0 0 0 1 0 0
## C 0 0 142 0 5 0 14 3 2 0 0 2 4
## D 1 1 0 196 0 1 4 12 5 3 4 4
## E 0 0 8 0 164 2 1 1 0 0 3 5
## F 0 0 0 0 0 171 4 2 8 2 0 0 0
## G 1 1 4 1 10 3 150 2 0 0 1 2
## H 0 3 0 1 0 2 2 122 0 2 4 2
## I 0 0 0 0 0 0 0 0 0 175 10 0 0
## J 2 2 0 0 0 3 0 2 7 158 0 0
## K 2 1 11 0 0 0 4 6 0 0 148 0
## L 0 0 0 0 1 0 1 1 0 0 0 176
## M 0 0 1 1 0 0 1 2 0 0 0 0 0
## N 0 0 0 1 0 1 0 1 0 0 0 0 0
## O 0 0 1 2 0 0 2 1 0 0 2 0 0
## P 0 0 0 1 0 3 1 0 0 0 0 0 0
## Q 0 0 0 0 0 0 9 3 0 0 0 0 3
## R 2 5 0 1 1 0 2 9 0 0 11 0
## S 1 2 0 0 1 1 5 0 2 2 0 3
## T 0 0 0 0 3 6 0 1 0 0 1 0
## U 1 0 3 3 0 0 0 2 0 0 0 0
## V 0 0 0 0 0 1 6 3 0 0 0 0
## W 0 0 0 0 0 0 1 0 0 0 0 0
## X 0 1 0 0 2 0 0 1 3 0 2 6
## Y 3 0 0 0 0 0 0 1 0 0 0 0
## Z 2 0 0 0 2 0 0 3 3 0 0 0
##
## hand_letter_predictions M N O P Q R S T U V W X
## A 1 2 2 0 5 0 2 1 1 0 1 0
## B 3 0 0 2 4 8 5 0 0 3 0 1
## C 0 0 2 0 0 0 0 0 0 0 0 0
## D 0 6 5 3 1 4 0 0 0 0 0 5
## E 0 0 0 6 0 10 0 0 0 0 0 4
## F 0 0 0 18 0 0 5 2 0 0 0 1
## G 1 0 0 2 11 2 5 3 0 0 0 1
## H 2 5 23 0 2 6 0 4 1 4 0 0
## I 0 0 0 1 0 0 3 0 0 0 0 4
## J 0 0 1 1 4 0 1 0 0 0 0 2
## K 0 2 0 1 1 7 0 1 3 0 0 4
## L 0 0 0 0 1 0 4 0 0 0 0 1
## M 177 5 1 0 0 0 0 0 4 0 8 0
## N 0 172 0 0 0 3 0 0 1 0 2 0
## O 0 1 132 2 4 0 0 0 3 0 0 0
## P 0 0 3 168 1 0 0 1 0 0 0 0
## Q 0 0 5 1 163 0 5 0 0 0 0 0
## R 1 1 1 1 0 176 0 1 0 2 0 0
## S 0 0 0 0 11 0 135 2 0 0 0 2
## T 0 0 0 0 0 0 0 3 163 1 0 0
## U 0 1 0 1 0 0 0 0 0 197 0 1 1
## V 0 3 1 0 2 1 0 0 0 0 152 1 0
## W 2 0 4 0 0 0 0 0 4 7 154 0
## X 0 0 1 0 0 1 2 0 0 0 0 160
## Y 0 0 0 6 0 0 0 3 0 0 0 0
## Z 0 0 0 0 1 0 18 3 0 0 0 0
##

```

```

## hand_letter_predictions  Y  Z
##                           A  0  0
##                           B  0  0
##                           C  0  0
##                           D  3  1
##                           E  0  3
##                           F  3  0
##                           G  0  0
##                           H  3  0
##                           I  1  1
##                           J  0  11
##                           K  0  0
##                           L  0  1
##                           M  0  0
##                           N  0  0
##                           O  0  0
##                           P  1  0
##                           Q  3  0
##                           R  0  0
##                           S  0  10
##                           T  5  2
##                           U  1  0
##                           V  5  0
##                           W  0  0
##                           X  1  1
##                           Y 157  0
##                           Z  0 164

# look only at agreement vs. non-agreement
# construct a vector of TRUE/FALSE indicating correct/incorrect predictions
agreement <- hand_letter_predictions == hand_letters_test$letter

# check if characters agree





```

11.6.4 Step 4: Improving Model Performance

Replacing the vanilladot linear kernel with rbf dot Radial Basis Function kernel, i.e., “Gaussian” kernel, may improve the OCR prediction.

```

hand_Letter_classifier_rbf <- ksvm(letter ~ ., data = hand_letters_train,
kernel = "rbfdot")
hand_Letter_predictions_rbf <- predict(hand_Letter_classifier_rbf,
hand_letters_test)

agreement_rbf <- hand_Letter_predictions_rbf == hand_letters_test$letter





```

```

## agreement_rbf
## FALSE  TRUE
## 360 4640

prop.table(table(agreement_rbf))

## agreement_rbf
## FALSE  TRUE
## 0.072 0.928

```

Note the improvement of the automated (SVM) classification accuracy (0.928) for `rbfdot` compared to the previous (`vanilladot`) result (0.844).

11.7 Case Study 4: Iris Flowers

Let's have another look at the *iris data* that we saw in Chap. 2.

11.7.1 Step 1: Collecting Data

SVM requires all features to be numeric, and each feature has to be scaled into a relative small interval. We are using the Edgar Anderson's Iris Data in R for this case study. This dataset measures the length and width of sepals and petals from three Iris flower species.

11.7.2 Step 2: Exploring and Preparing the Data

Let's load the data first. In this case study we want to explore the variable `Species`.

```

data(iris)
str(iris)

## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species     : Factor w/ 3 Levels "setosa","versicolor",..: 1 1 1 1 1
1 1 1 1 ...

table(iris$Species)

##
##      setosa versicolor  virginica
##      50      50      50

```

The data look good but we still can normalize the features either by hand or using an R function.

Next, we can separate the training and testing datasets using 75%–25% rule.

```
sub<-sample(nrow(iris), floor(nrow(iris)*0.75))
iris_train<-iris[sub, ]
iris_test<-iris[-sub, ]
```

We can try the linear and non-linear kernels on the iris data (Figs. 11.15 and 11.16).

```
require(e1071)

iris.svm_1 <- svm(Species~Petal.Length+Petal.Width, data=iris_train,
                    kernel="linear", cost=1)
iris.svm_2 <- svm(Species~Petal.Length+Petal.Width, data=iris_train,
                    kernel="radial", cost=1)
par(mfrow=c(2,1))
plot(iris.svm_1, iris[,c(5,3,4)]); legend("center", "Linear")

plot(iris.svm_2, iris[,c(5,3,4)]); legend("center", "Radial")
```

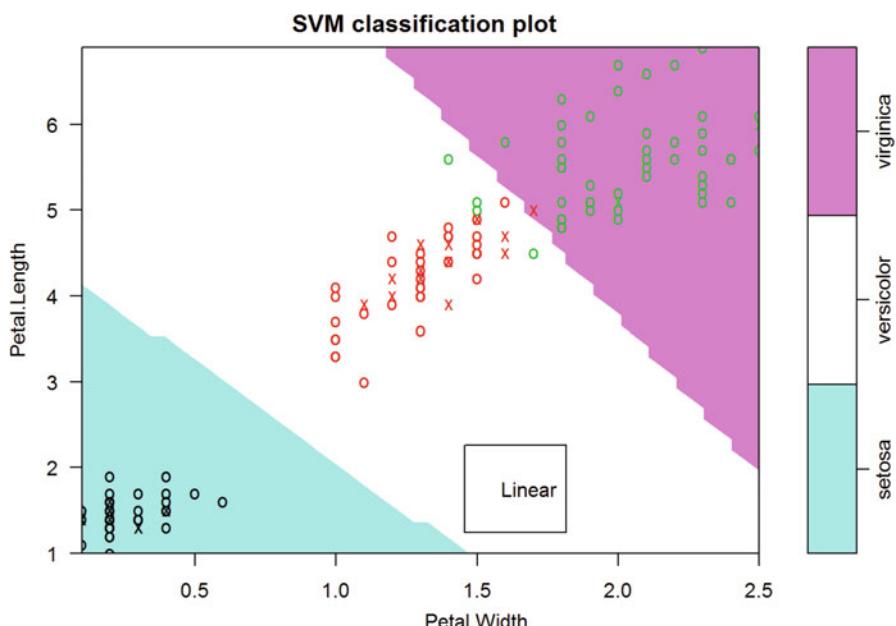


Fig. 11.15 Linear-SVM kernel classification of the iris flowers dataset

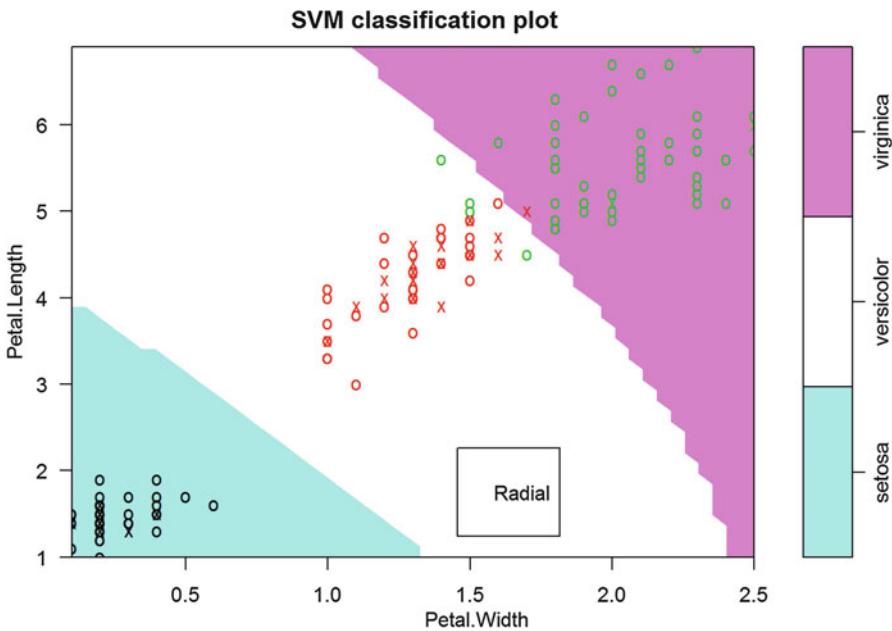


Fig. 11.16 Radial-SVM kernel classification of the iris flowers dataset

11.7.3 Step 3: Training a Model on the Data

We are going to use `kernlab` for this case study. However other packages like `e1071` and `klaR` are available if you are quite familiar with C++.

Let's break down the function `ksvm()`

```
m <- ksvm(target~predictors, data = mydata, kernel = "rbfdot", C = 1)
```

- target: the outcome variable that we want to predict.
- predictors: features that the prediction based on. In this function we can use the `"."` to represent all the variables in the dataset again.
- data: the training dataset that the *target* and *predictors* can be find.
- kernel: is the kernel mapping we want to use. By default it is the radio basis function (`rbfdot`).
- C is a number that specifies the cost of misclassification.

Let's install the package and play with the data now.

```
# install.packages("kernlab")
library(kernlab)
iris_clas<-ksvm(Species~, data=iris_train, kernel="vanilladot")
## Setting default kernel parameters
iris_clas
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc (classification)
## parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 24
##
## Objective Function Value : -1.0066 -0.3309 -13.8658
## Training error : 0.026786
```

Here, we used all the variables other than the `Species` in the dataset as predictors. We also used kernel `vanilladot`, which is the linear kernel in this model. We get a training error that is less than 0.02.

11.7.4 Step 4: Evaluating Model Performance

Again, the `predict()` function is used to forecast the species for a test data. Here, we have a factor outcome, so we need the command `table()` to show us how well do the predictions and actual data match.

```
iris.pred<-predict(iris_clas, iris_test)
table(iris.pred, iris_test$Species)

##
## iris.pred      setosa versicolor virginica
##   setosa        13        0        0
##   versicolor     0        14        0
##   virginica      0        1       10
```

We can see a single case of Iris virginica *misclassified* as Iris versicolor. The taxa of all other flowers are correctly predicted.

To see the results more clearly, we can use the proportional table to show the agreements of the categories.

```
agreement<-iris.pred==iris_test$Species
prop.table(table(agreement))

##
## agreement
##      FALSE      TRUE
## 0.02631578947 0.97368421053
```

Here $==$ means “equal to”. Over 97% of predictions are correct. Nevertheless, is there any chance that we can improve the outcome? What if we try a Gaussian kernel?

11.7.5 Step 5: RBF Kernel Function

Linear kernel is the simplest one but usually not the best one. Let’s try the **RBF** (Radial Basis “Gaussian” Function) kernel instead.

```
iris_clas1<-ksvm(Species~, data=iris_train, kernel="rbfdot")
iris_clas1

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc (classification)
## parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
## Hyperparameter : sigma = 0.877982617394805
##
## Number of Support Vectors : 52
##
## Objective Function Value : -4.6939 -5.1534 -16.2297
## Training error : 0.017857

iris.pred1<-predict(iris_clas1, iris_test)
table(iris.pred1, iris_test$Species)

##
## iris.pred1  setosa versicolor virginica
##   setosa      13        0        0
##   versicolor    0       14        2
##   virginica     0        1        8

agreement<-iris.pred1==iris_test$Species
prop.table(table(agreement))

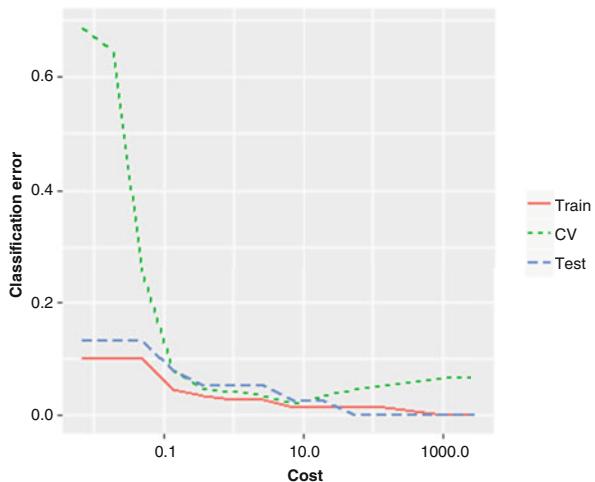
## agreement
##      FALSE      TRUE
## 0.07894736842 0.92105263158
```

Unfortunately, the model performance is actually worse than the previous one (you might get slightly different results). This is because this Iris dataset has a linear feature. In practice, we could try some alternative kernel functions and see which one fits the dataset the best.

11.7.6 Parameter Tuning

We can tune the SVM using the `tune.svm` function in the package `e1071` (Fig. 11.17).

Fig. 11.17 Training data, cross-validation, and testing data SVM classification errors of the iris flowers



```
costs = exp(-5:8)
tune.svm(Species~, kernel = "radial", data = iris_train, cost = costs)

## 
## Parameter tuning of 'svm':
## 
## - sampling method: 10-fold cross validation
## 
## - best parameters:
##   cost
##     1
## 
## - best performance: 0.03636363636
```

Further, we can draw a **Cross-Validation (CV)** plot to gauge the model performance, see cross-validation details in Chap. 21:

```
set.seed(2017)
require(sparsediscrim); require(reshape); require(ggplot2)

folds = cv_partition(iris$Species, num_folds = 5)
train_cv_error_svm = function(costC) {
  #Train
  ir.svm = svm(Species~, data=iris,
               kernel="radial", cost=costC)
  train_error = sum(ir.svm$fitted != iris$Species) / nrow(iris)
  #Test
  test_error = sum(predict(ir.svm, iris_test) != iris_test$Species) / nrow(i
ris_test)
  #CV error
  ire.cverr = sapply(folds, function(fold) {
    svmcv = svm(Species~, data = iris, kernel="radial", cost=costC, subset =
fold$training)
    svmpred = predict(svmcv, iris[fold$test,])
    return(sum(svmpred != iris$Species[fold$test]) / Length(fold$test))
  })
}
```

```

cv_error = mean(ire.cverr)
return(c(train_error, cv_error, test_error))
}

costs = exp(-5:8)
ir_cost_errors = sapply(costs, function(cost) train_cv_error_svm(cost))
df_errs = data.frame(t(ir_cost_errors), costs)
colnames(df_errs) = c('Train', 'CV', 'Test', 'Logcost')
dataL <- melt(df_errs, id="Logcost")
ggplot(dataL, aes_string(x="Logcost", y="value", colour="variable",
  group="variable", linetype="variable", shape="variable")) +
  geom_line(size=1) + labs(x = "Cost",
    y = "Classification error", colour="",group="",
    linetype="" ,shape="") + scale_x_Log10()

```

11.7.7 Improving the Performance of Gaussian Kernels

Now, let's attempt to improve the performance of a Gaussian kernel by tuning:

```

set.seed(2020)
gammas = exp(-5:5)
tune_g = tune.svm(Species~., kernel = "radial", data = iris_train, cost = costs,
  gamma = gammas)
tune_g

## Parameter tuning of 'svm':
## - sampling method: 10-fold cross validation
## - best parameters:
##       gamma      cost
## 0.01831563889 20.08553692
## - best performance: 0.025

```

We observe that the model achieves a better prediction now.

```

iris.svm_g <- svm(Species~., data=iris_train,
  kernel="radial", gamma=0.0183, cost=20)
table(iris_test$Species, predict(iris.svm_g, iris_test))

##
##           setosa versicolor virginica
##   setosa        13        0        0
##   versicolor     0       14        1
##   virginica      0        0       10

agreement<-predict(iris.svm_g, iris_test)==iris_test$Species
prop.table(table(agreement))

## agreement
##      FALSE      TRUE
## 0.02631578947 0.97368421053

```

Chapter 23 provides more details about prediction and classification using neural networks and deep learning.

11.8 Practice

11.8.1 Problem 1 Google Trends and the Stock Market

Use the Google trend data. Fit a neural network model with the same training data as case study 1. This time, use `Investing` as target and `Unemployment`, `Rental`, `RealEstate`, `Mortgage`, `Jobs`, `DJI_Index`, `StdDJI` as predictors. Use three hidden nodes. **Note:** remember to change the columns you want to include in the test dataset when predicting.

The following number is the correlation between predicted and observed values.

```
## [1,] 0.8845711444
```

You might get a slightly different results since the weights are generated randomly.

11.8.2 Problem 2: Quality of Life and Chronic Disease

Use the same data in Chap. 8 – Quality of life and chronic disease (dataset and meta-data doc).

Let's load the data first. In this case study, we want to use the variable `CHARLSONSCORE` as our target variable.

```
qol<-read.csv("https://umich.instructure.com/files/481332/downLoad?downLoad_
frd=1")
str(qol)

## 'data.frame': 2356 obs. of 41 variables:
## $ ID : int 171 171 172 179 180 180 181 182 183 186 ...
## $ INTERVIEWDATE : int 0 427 0 0 0 42 0 0 0 0 ...
## $ LANGUAGE : int 1 1 1 1 1 1 1 1 2 ...
## $ AGE : int 49 49 62 44 64 64 52 48 49 78 ...
## $ RACE_ETHNICITY : int 3 3 3 7 3 3 3 3 3 4 ...
## $ SEX : int 2 2 2 2 1 1 2 1 1 1 ...
## $ QOL_Q_01 : int 4 4 3 6 3 3 4 2 3 5 ...
## $ QOL_Q_02 : int 4 3 3 6 2 5 4 1 4 6 ...
## $ QOL_Q_03 : int 4 4 4 6 3 6 4 3 4 4 ...
## $ QOL_Q_04 : int 4 4 2 6 3 6 2 2 5 2 ...
## $ QOL_Q_05 : int 1 5 4 6 2 6 4 3 4 3 ...
## $ QOL_Q_06 : int 4 4 2 6 1 2 4 1 2 4 ...
## $ QOL_Q_07 : int 1 2 5 -1 0 5 8 4 3 7 ...
## $ QOL_Q_08 : int 6 1 3 6 6 6 3 1 2 4 ...
## $ QOL_Q_09 : int 3 4 3 6 2 2 4 2 2 4 ...
## $ QOL_Q_10 : int 3 1 3 6 3 6 3 2 4 3 ...
## $ MSA_Q_01 : int 1 3 2 6 2 3 4 1 1 2 ...
```

```

## $ MSA_Q_02      : int 1 1 2 6 1 6 4 3 2 4 ...
## $ MSA_Q_03      : int 2 1 2 6 1 2 3 3 1 2 ...
## $ MSA_Q_04      : int 1 3 2 6 1 2 1 4 1 5 ...
## $ MSA_Q_05      : int 1 1 1 6 1 2 1 6 3 2 ...
## $ MSA_Q_06      : int 1 2 2 6 1 2 1 1 2 2 ...
## $ MSA_Q_07      : int 2 1 3 6 1 1 1 1 1 5 ...
## $ MSA_Q_08      : int 1 1 1 6 1 1 1 1 2 1 ...
## $ MSA_Q_09      : int 1 1 1 6 2 2 4 6 2 1 ...
## $ MSA_Q_10      : int 1 1 1 6 1 1 1 1 1 3 ...
## $ MSA_Q_11      : int 2 3 2 6 1 1 2 1 1 5 ...
## $ MSA_Q_12      : int 1 1 2 6 1 1 2 6 1 3 ...
## $ MSA_Q_13      : int 1 1 1 6 1 6 2 1 4 2 ...
## $ MSA_Q_14      : int 1 1 1 6 1 2 1 1 3 1 ...
## $ MSA_Q_15      : int 2 1 1 6 1 1 3 2 1 3 ...
## $ MSA_Q_16      : int 2 3 5 6 1 2 1 2 1 2 ...
## $ MSA_Q_17      : int 2 1 1 6 1 1 1 1 1 3 ...
## $ PH2_Q_01      : int 3 2 1 5 1 1 3 1 2 3 ...
## $ PH2_Q_02      : int 4 4 1 5 1 2 1 1 4 2 ...
## $ TOS_Q_01      : int 2 2 2 4 1 1 2 2 1 1 ...
## $ TOS_Q_02      : int 1 1 1 4 4 4 1 2 4 4 ...
## $ TOS_Q_03      : int 4 4 4 4 4 4 4 4 4 4 ...
## $ TOS_Q_04      : int 5 5 5 5 5 5 5 5 5 5 ...
## $ CHARLSONSCORE : int 2 2 3 1 0 0 2 8 0 1 ...
## $ CHRONICDISEASESCORE: num 1.6 1.6 1.54 2.97 1.28 1.28 1.31 1.67 2.21 2
.51 ...

```

Remove the first two columns (we don't need ID variables) and rows that have missing CHARLSONSCORE values (e.g., CHARLSONSCORE equal to "-9"). Note that, `! qol$CHARLSONSCORE == -9`, implies that we only select the rows that have CHARLSONSCORE not equal to -9. The exclamation sign indicates "exclude". Also, we need to convert our categorical variable CHARLSONSCORE into a factor.

```

qol<-qol[!qol$CHARLSONSCORE== -9 , -c(1, 2)]
qol$CHARLSONSCORE<-as.factor(qol$CHARLSONSCORE)
str(qol)

## 'data.frame': 2328 obs. of 39 variables:
## $ LANGUAGE      : int 1 1 1 1 1 1 1 1 2 ...
## $ AGE           : int 49 49 62 44 64 64 52 48 49 78 ...
## $ RACE_ETHNICITY: int 3 3 3 7 3 3 3 3 3 4 ...
## $ SEX           : int 2 2 2 2 1 1 2 1 1 1 ...
## $ QOL_Q_01       : int 4 4 3 6 3 3 4 2 3 5 ...
## $ QOL_Q_02       : int 4 3 3 6 2 5 4 1 4 6 ...
## $ QOL_Q_03       : int 4 4 4 6 3 6 4 3 4 4 ...
## $ QOL_Q_04       : int 4 4 2 6 3 6 2 2 5 2 ...
## $ QOL_Q_05       : int 1 5 4 6 2 6 4 3 4 3 ...
## $ QOL_Q_06       : int 4 4 2 6 1 2 4 1 2 4 ...
## $ QOL_Q_07       : int 1 2 5 -1 0 5 8 4 3 7 ...
## $ QOL_Q_08       : int 6 1 3 6 6 6 3 1 2 4 ...
## $ QOL_Q_09       : int 3 4 3 6 2 2 4 2 2 4 ...
## $ QOL_Q_10       : int 3 1 3 6 3 6 3 2 4 3 ...
## $ MSA_Q_01       : int 1 3 2 6 2 3 4 1 1 2 ...
## $ MSA_Q_02       : int 1 1 2 6 1 6 4 3 2 4 ...

```

```

## $ MSA_Q_03      : int  2 1 2 6 1 2 3 3 3 1 2 ...
## $ MSA_Q_04      : int  1 3 2 6 1 2 1 4 1 5 ...
## $ MSA_Q_05      : int  1 1 1 6 1 2 1 6 3 2 ...
## $ MSA_Q_06      : int  1 2 2 6 1 2 1 1 2 2 ...
## $ MSA_Q_07      : int  2 1 3 6 1 1 1 1 1 5 ...
## $ MSA_Q_08      : int  1 1 1 6 1 1 1 1 2 1 ...
## $ MSA_Q_09      : int  1 1 1 6 2 2 4 6 2 1 ...
## $ MSA_Q_10      : int  1 1 1 6 1 1 1 1 1 3 ...
## $ MSA_Q_11      : int  2 3 2 6 1 1 2 1 1 5 ...
## $ MSA_Q_12      : int  1 1 2 6 1 1 2 6 1 3 ...
## $ MSA_Q_13      : int  1 1 1 6 1 6 2 1 4 2 ...
## $ MSA_Q_14      : int  1 1 1 6 1 2 1 1 3 1 ...
## $ MSA_Q_15      : int  2 1 1 6 1 1 3 2 1 3 ...
## $ MSA_Q_16      : int  2 3 5 6 1 2 1 2 1 2 ...
## $ MSA_Q_17      : int  2 1 1 6 1 1 1 1 1 3 ...
## $ PH2_Q_01      : int  3 2 1 5 1 1 3 1 2 3 ...
## $ PH2_Q_02      : int  4 4 1 5 1 2 1 1 4 2 ...
## $ TOS_Q_01      : int  2 2 2 4 1 1 2 2 1 1 ...
## $ TOS_Q_02      : int  1 1 1 4 4 4 1 2 4 4 ...
## $ TOS_Q_03      : int  4 4 4 4 4 4 4 4 4 4 ...
## $ TOS_Q_04      : int  5 5 5 5 5 5 5 5 5 5 ...
## $ CHARLSONSCORE : Factor w/ 11 Levels "0","1","2","3",...: 3 3 4 2 1
1 3 9 1 2 ...
## $ CHRONICDISEASESCORE: num 1.6 1.6 1.54 2.97 1.28 1.28 1.31 1.67 2.21
2.51 ...

```

The dataset is now ready for processing. First, separate the dataset into training and test datasets using the 75%–25% rule. Then, build a SVM model using all other variables in the dataset to be predictor variables. Try to add different cost of misclassification to the model. Rather than the default $C = 1$, we will explore the behavior of the model for $C = 2$ and $C = 3$ utilizing the radial basis kernel.

The output for $C = 2$ is included below.

```

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc (classification)
## parameter : cost C = 2
##
## Gaussian Radial Basis kernel function.
## Hyperparameter : sigma = 0.0174510649312293
##
## Number of Support Vectors : 1703
##
## Objective Function Value : -1798.9778 -666.9432 -352.2265 -46.2968 -15.92
36 -9.2176 -7.1853 -27.9366 -16.3096 -3.5681 -697.4275 -362.6579 -47.0801 -1
6.3701 -9.6556 -6.9882 -28.2074 -16.4556 -3.5121 -321.0676 -44.7405 -15.8416
-9.1439 -6.8161 -26.7174 -15.4833 -3.3944 -43.1026 -15.2923 -7.994 -6.58 -24
.8459 -14.6379 -3.4484 -13.9377 -5.2876 -5.6728 -15.2542 -9.8408 -3.255 -4.6
982 -4.8924 -9.2482 -6.5144 -2.9608 -2.7409 -6.2056 -6.0476 -2.0833 -6.1775
-4.919 -2.7715 -10.5691 -3.0835 -2.566
## Training error : 0.310997

##
## qol.pred2  0   1   2   3   4   5   6   7   8   9   10
##      0 126 76 24  5  1  0  1  0  3  0  1
##      1 88 170 47 19  9  2  1  0  4  3  0

```

```

##      2   1   0   0   1   0   0   0   0   0   0   0   0   0
##      3   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      4   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      5   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      6   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      7   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      8   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      9   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##     10   0   0   0   0   0   0   0   0   0   0   0   0   0   0

## agreement
##      FALSE      TRUE
## 0.4914089347 0.5085910653

```

The output for $C = 3$ is included below.

```

## Support Vector Machine object of class "ksvm"
## SV type: C-svc (classification)
## parameter : cost C = 3
##
## Gaussian Radial Basis kernel function.
## Hyperparameter : sigma = 0.0168577510531693
##
## Number of Support Vectors : 1695
##
## Objective Function Value : -2440.0638 -915.9967 -492.6748 -63.2895 -21.09
29 -11.9108 -10.2404 -39.1843 -21.976 -5.0624 -970.6173 -514.9584 -64.7791 -
22.0947 -12.8987 -9.8114 -39.7908 -22.2957 -4.9403 -431.5178 -59.9296 -20.94
08 -11.7468 -9.4269 -36.602 -20.1783 -4.6829 -56.9469 -19.7357 -9.238 -8.904
7 -32.6121 -18.4667 -4.8007 -17.3102 -5.4133 -6.9733 -17.2097 -10.3016 -4.37
39 -4.7816 -5.7083 -9.7236 -6.6365 -3.723 -2.7726 -6.4151 -6.4453 -2.1222 -8
.03 -5.411 -3.3088 -11.9186 -3.996 -2.8572
## Training error : 0.266896

##
## qol.pred3   0   1   2   3   4   5   6   7   8   9   10
##      0 131 79 24  6  2  0  1  0  4  1  1
##      1 83 165 47 18  8  2  1  0  3  2  0
##      2  1  2  0  1  0  0  0  0  0  0  0
##      3  0  0  0  0  0  0  0  0  0  0  0
##      4  0  0  0  0  0  0  0  0  0  0  0
##      5  0  0  0  0  0  0  0  0  0  0  0
##      6  0  0  0  0  0  0  0  0  0  0  0
##      7  0  0  0  0  0  0  0  0  0  0  0
##      8  0  0  0  0  0  0  0  0  0  0  0
##      9  0  0  0  0  0  0  0  0  0  0  0
##     10  0  0  0  0  0  0  0  0  0  0  0

## agreement
##      FALSE      TRUE
## 0.4914089347 0.5085910653

```

Can you reproduce (approximately) these results?

11.9 Appendix

Below is some additional R code demonstrating the various results reported in this **Chapter**.

```
#Picture 1
x<-runif(1000, -10, 10)
y<-ifelse(x>=0, 1, 0)
plot(x, y, xlab = "Sum of input signals", ylab = "Output signal", main = "Threshold")
abline(v=0, lty=2)
#Picture 2
x<-runif(100000, -10, 10)
y<-1/(1+exp(-x))
plot(x, y, xlab = "Sum of input signals", ylab = "Output signal", main = "Sigmoid")
#Picture 3
x<-runif(100000, -10, 10)
y1<-x
y2<-ifelse(x<=-5, -5, ifelse(x>=5, 5, x))
y3<-(exp(x)-exp(-x))/(exp(x)+exp(-x))
y4<-exp(-x^2/2)
par(mfrow=c(2, 2))
plot(x, y1, main="Linear", xlab="", ylab="")
plot(x, y2, main="Saturated Linear", xlab="", ylab="")
plot(x, y3, main="Hyperbolic tangent", xlab="", ylab="")
plot(x, y4, main = "Gaussian", xlab="", ylab="")
#Picture 4
A<-c(1, 4, 3, 2, 4, 8, 6, 10, 9)
B<-c(1, 5, 3, 2, 3, 8, 8, 7, 10)
plot(A, B, xlab="", ylab="", pch=16, cex=2)
abline(v=5, col="red", lty=2)
text(5.4, 9, labels="A")
abline(12, -1, col="red", lty=2)
text(6, 5.4, labels="B")
#Picture 5
plot(A, B, xlab="", ylab="", pch=16, cex=2)
segments(1, 1, 4, 5, lwd=1, col = "red")
segments(1, 1, 4, 3, lwd = 1, col = "red")
segments(4, 3, 4, 5, lwd = 1, col = "red")
segments(6, 8, 10, 7, lwd = 1, col = "red")
segments(6, 8, 9, 10, lwd = 1, col = "red")
segments(10, 7, 9, 10, lwd = 1, col = "red")
segments(6, 8, 4, 5, lwd = 1, lty=2)
abline(9.833, -2/3, lwd=2)
```

Try to replicate these results with other data from the list of our Case-Studies.

11.10 Assignments: 11. Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines

11.10.1 Learn and Predict a Power-Function

In Chap. 11, we learned about predicting the square-root function. It's just one instance of the power-function.

- Why did we observe a decrease in the accuracy of the NN prediction of the square-root outside the interval $[0, 1]$ (note we trained inside $[0, 1]$)? How can you improve on the prediction of the square-root network?
- Can you design a more generic NN network that can learn and predict a power-function for a given power ($\lambda \in \mathfrak{R}$)?

11.10.2 Pediatric Schizophrenia Study

Use the SOCR Normal and Schizophrenia pediatric neuroimaging study data to complete the following tasks:

- Conduct some initial data visualization and exploration.
- Use derived neuroimaging biomarkers (e.g., *Age*, *FS_IQ*, *TBV*, *GMV*, *WMV*, *CSF*, *Background*, *L_superior_frontal_gyrus*, *R_superior_frontal_gyrus*, ..., *brainstem*) to train a NN model and predict *DX* (Normals = 1; Schizophrenia = 2).
- Try one hidden layer with a different number of nodes.
- Try multiple hidden layers and compare the results to the single layer. Which model is better?
- Compare the type I (false-positive) and type II (false-negative) errors for the alternative methods.
- Train separate models to predict *DX* (diagnosis) for the *Male* and *Female* cohorts, respectively. Explain your findings.
- Train an *SVM*, using *ksvm* and *svm* in *e1071*, for *Age*, *FS_IQ*, *TBV*, *GMV*, *WMV*, *CSF*, *Background* to predict *DX*. Compare the results of linear, Gaussian, and polynomial SVM kernels.
- Add *Sex* to your models and see if this makes a difference.
- Expand the model by training on all derived neuroimaging biomarkers and re-train the SVM using *Age*, *FS_IQ*, *TBV*, *GMV*, *WMV*, *CSF*, *Background*, *L_superior_frontal_gyrus*, *R_superior_frontal_gyrus*, ..., *brainstem*. Again, try linear, Gaussian, and polynomial kernels. Compare the results.
- Are there differences between the alternative kernels?
- For *Age*, *FS_IQ*, *TBV*, *GMV*, *WMV*, *CSF*, and *Background*, tune parameters for Gaussian and polynomial kernels.

- Generate a CV (cross-validation) plot and interpret the resulting graph.
- Use different random seeds and repeat the experiment five times. Are the results stable?
- Inspecting the results above, explain why it makes sense to set a tune over a range such as $\exp(-5 : 8)$.
- How can we design alternative tuning strategies other than greedy search?

References

- Schölkopf, B, Smola, AJ. (2002) *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond Adaptive computation and machine learning*, MIT Press, ISBN 0262194759, 9780262194754.
- Du, K-L, Swamy, MNS. (2013) *Neural Networks and Statistical Learning*, SpringerLink: Bücher, ISBN 1447155718, 9781447155713.
- Abe, S. (2010) *Support Vector Machines for Pattern Classification*, Advances in Computer Vision and Pattern Recognition, Springer Science & Business Media, ISBN 1849960984, 9781849960984.

Chapter 12

Apriori Association Rules Learning



HTTP cookies are used to monitor web-traffic and track users surfing the Internet. We often notice that promotions (ads) on websites tend to match our needs, reveal our prior browsing history, or reflect our interests. That is not an accident. Nowadays, recommendation systems are highly based on machine learning methods that can learn the behavior, e.g., purchasing patterns, of individual consumers. In this chapter, we will uncover some of the mystery behind recommendation systems for transactional records. Specifically, we will (1) discuss association rules and their support and confidence; (2) the *Apriori algorithm* for association rule learning; and (3) cover step-by-step a set of case-studies, including a toy example, Head and Neck Cancer Medications, and Grocery purchases.

12.1 Association Rules

Association rules are the result of process analytics (e.g., market analysis) that specify patterns of relationships among items. One specific example would be:

$$\{charcoal, lighter, chicken\ wings\} \rightarrow \{barbecue\ sauce\}$$

In words, charcoal, lighter and chicken wings imply barbecue sauce. Those curly brackets indicate that we have a set. Items in a set are called elements. When an item-set like $\{charcoal, lighter, chicken\ wings, barbecue\ sauce\}$ appears in our dataset with some regularity, we can discover the above pattern.

Association rules are commonly used for unsupervised discovery of knowledge rather than prediction of outcomes. In biomedical research, association rules are widely used to:

- Search for interesting or frequently occurring patterns of DNA.
- Search for protein sequences in an analysis of cancer data.
- Find patterns of medical claims that occur in combination with fraudulent credit card or insurance use.

12.2 The Apriori Algorithm for Association Rule Learning

Association rules are mostly applied to transactional data, like business, trade, service or medical records. These datasets are typically very large in number of transactions and features. This will add lots of possible orders and patterns when we try to do analytics, which makes data mining a very hard task.

With the **Apriori** rule, this problem is easily solved. If we have a simple prior (belief about the properties of frequent elements), we can efficiently reduce the number of features or combinations that we need to look at.

The Apriori algorithm has a simple *apriori* belief that *all subsets of a frequent item-set must also be frequent*. This is known as the **Apriori property**. The full set in the last example, *{charcoal, lighter, chicken wings, barbecue sauce}*, can be frequent if and only if itself and all its subsets of single elements, pairs and triples occur frequently. We can see that this algorithm is designed for finding patterns in large datasets. If a pattern happens frequently, it is considered “interesting”.

12.3 Measuring Rule Importance by Using Support and Confidence

Support and confidence are the two criteria to help us decide whether a pattern is “interesting”. By setting thresholds for these two criteria, we can easily limit the number of interesting rules or item-sets reported.

For item-sets X and Y , the support of an item-set measures how frequently it appears in the data:

$$support(X) = \frac{count(X)}{N},$$

where N is the total number of transactions in the database and $count(X)$ is the number of observations (transactions) containing the item-set X . Of course, the union of item-sets is an item-set itself. For example, if $Z = X, Y$, then

$$support(Z) = support(X, Y).$$

For a rule $X \rightarrow Y$, the rule’s confidence measures the relative accuracy of the rule:

$$\text{confidence}(X \rightarrow Y) = \frac{\text{support}(X, Y)}{\text{support}(X)}$$

This measures the joint occurrence of X and Y over the X domain. If whenever X appears Y tends to be present too, we will have a high $\text{confidence}(X \rightarrow Y)$. The ranges of the support and confidence are $0 \leq \text{support}, \text{confidence} \leq 1$. Note that in probabilistic terms, $\text{Confidence}(X \rightarrow Y)$ is equivalent to the conditional probability $P(Y|X)$.

$\{\text{peanut butter}\} \rightarrow \{\text{bread}\}$ would be an example of a strong rule because it has high support as well as high confidence in grocery store transactions. Shoppers tend to purchase bread when they get peanut butter. These items tend to appear in the same baskets, which yields high confidence for the rule $\{\text{peanut butter}\} \rightarrow \{\text{bread}\}$.

12.4 Building a Set of Rules with the Apriori Principle

To build a set of rules, we need to go through two steps:

- **Step 1:** Filter all item-sets with a minimum support threshold. This is accomplished iteratively by increasing the size of the item-sets. In the first iteration, we compute the support of singletons, 1-item-sets. At the next iteration, we compute the support of pairs of items, and so on. Item-sets passing iteration i could be considered as candidates for the next iteration, $i + 1$. If $\{A\}$, $\{B\}$, $\{C\}$ are all frequent, but D is not frequent in the first singleton-selection round, then in the second iteration we only consider the support of these pairs $\{A, B\}$, $\{A, C\}$, $\{B, C\}$, ignoring all pairs including D . This substantially reduces the cardinality of the potential item-sets and ensures the feasibility of the algorithm. At the third iteration, if $\{A, C\}$, and $\{B, C\}$ are frequently occurring, but $\{A, B\}$ is not, then the algorithm may terminate, as the support of $\{A, B, C\}$ is trivial (does not pass the support threshold), given that $\{A, B\}$ was not frequent enough.
- **Step 2:** Using the item-sets selected in Step 1, generate new rules with confidence larger than a predefined minimum confidence threshold. The candidate item-sets that passed Step 1 would include all frequent item-sets. For the highly-supported item-set $\{A, C\}$, we would compute the confidence measures for $\{A\} \rightarrow \{C\}$ as well as $\{C\} \rightarrow \{A\}$ and compare these against the minimum confidence threshold. The *surviving rules are the ones with confidence levels exceeding that minimum threshold*.

12.5 A Toy Example

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item, i.e., each item, such as “butter” or “bread”, is identified by an SKU number. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together (Table 12.1).

Suppose the database of transactions consist of following item-sets, each representing a purchasing order:

```
require(knitr)
item_table = as.data.frame(t(c("{1,2,3,4}", "{1,2,4}", "{1,2}", "{2,3,4}",
  "{2,3}", "{3,4}", "{2,4}")))
colnames(item_table) <- c("choice1", "choice2", "choice3", "choice4",
  "choice5", "choice6", "choice7")
kable(item_table, caption = "Item table")
```

We will use *Apriori* to determine the frequent item-sets of this database. To do so, we will say that an item-set is frequent if it appears in at least 3 transactions of the database, i.e., the value 3 is the support threshold (Table 12.2).

The first step of Apriori is to count up the number of occurrences, i.e., the support, of each member item separately. By scanning the database for the first time, we obtain get:

```
item_table = as.data.frame(t(c(3,6,4,5)))
colnames(item_table) <- c("item1", "item2", "item3", "item4")
rownames(item_table) <- "support"
kable(item_table, caption = "Size 1 Support")
```

All the item-sets of size 1 have a support of at least 3, so they are all frequent. The next step is to generate a list of all pairs of frequent items.

For example, regarding the pair {1,2}: the first table of Example 2 shows items 1 and 2 appearing together in three of the item-sets; therefore, we say that the support of the item {1,2} is 3 (Tables 12.3 and 12.4).

Table 12.1 Item table

choice1	choice2	choice3	choice4	choice5	choice6	choice7
{1,2,3,4}	{1,2,4}	{1,2}	{2,3,4}	{2,3}	{3,4}	{2,4}

Table 12.2 Size 1 support

	item1	item2	item3	item4
support	3	6	4	5

Table 12.3 Size 2 support

	{1,2}	{1,3}	{1,4}	{2,3}	{2,4}	{3,4}
support	3	1	2	3	4	3

Table 12.4 Size 3 support

	{2,3,4}
support	2

```
item_table = as.data.frame(t(c(3,1,2,3,4,3)))
colnames(item_table) <- c("{1,2}", "{1,3}", "{1,4}", "{2,3}", "{2,4}", "{3,4}")
rownames(item_table) <- "support"
kable(item_table, caption = "Size 2 Support")
```

The pairs $\{1,2\}$, $\{2,3\}$, $\{2,4\}$, and $\{3,4\}$ all meet or exceed the minimum support of 3, so they are *frequent*. The pairs $\{1,3\}$ and $\{1,4\}$ are not and any larger set which contains $\{1,3\}$ or $\{1,4\}$ cannot be frequent. In this way, we can prune sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

```
item_table = as.data.frame(t(c(2)))
colnames(item_table) <- c("{2,3,4}")
rownames(item_table) <- "support"
kable(item_table, caption = "Size 3 Support")
```

In the example, there are no frequent triplets – the support of the item-set $\{2,3,4\}$ is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold. We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because some of their subsets were already known to be below the threshold.

12.6 Case Study 1: Head and Neck Cancer Medications

12.6.1 Step 1: Collecting Data

To demonstrate the *Apriori* algorithm in a real biomedical case-study, we will use a transactional healthcare data representing a subset of the Head and Neck Cancer Medication data, which is available in our case-studies collection as `10_medication_descriptions.csv`. It consists of inpatient medications for head and neck cancer patients.

The data is in wide format, see Chap. 2, where each row represents a patient. During the study period, each patient had records for a maximum of 5 encounters. *NA* represents no medication administration records in this specific time point for the specific patient. This dataset contains a total of 528 patients.

12.6.2 Step 2: Exploring and Preparing the Data

Different from our data imports in the previous chapters, transactional data need to be ingested in R using the `read.transactions()` function. This function will store data as a matrix with each row representing an example and each column representing a feature.

Let's load the dataset and delete the irrelevant *index* column. With the `write.csv`(*R data*, "path") function we can output our R data file into a local CSV file. To avoid generating another index column in the output CSV file, we can use the `row.names = F` option.

```
med<-read.csv("https://umich.instructure.com/files/1678540/downLoad?downLoad
_frd=1", stringsAsFactors = FALSE)
med<-med[, -1]
write.csv(med, "medication.csv", row.names=F)
```

Now we can use `read.transactions()` in the `arules` package to read the CSV file we just outputted.

```
# install.packages("arules")
library(arules)

med<-read.transactions("medication.csv", sep = ",", skip = 1, rm.duplicates=
TRUE)
## distribution of transactions with duplicates:
## items
##  1  2  3
## 79 166 248

summary(med)

## transactions as itemMatrix in sparse format with
## 528 rows (elements/itemsets/transactions) and
## 88 columns (items) and a density of 0.02085486
##
## most frequent items:
##          fentanyl injection uh  hydrocodone acetaminophen 5mg 325mg
##                                211                               165
##          cefazolin ivpb uh          heparin injection
##                                108                               105
## hydrocodone acetamin 75mg 500mg 15ml          (Other)
##                                60                               320
##
## element (itemset/transaction) Length distribution:
## sizes
##  1  2  3  4  5
## 248 166 79 23 12
##
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.000  1.000  2.000  1.835  2.000  5.000
##
## includes extended item information - examples:
##          labels
## 1          09 nacl
## 2          09 nacl bolus
## 3 acetaminophen multiroute uh
```

Here we use the option `rm.duplicates = T` because we may have similar medication administration records for two different patients. The option `skip = 1` means we skip the heading line in the CSV file. Now we get a transactional data with unique rows.

The summary of a transactional data contains rich information. The first block of information tells us that we have 528 rows and 88 different medicines in this matrix. Using the density number we can calculate how many non *NA* medication records are in the data. In total, we have $528 \times 88 = 46,464$ positions in the matrix. Thus, there are $46,464 \times 0.0209 = 971$ medicines prescribed during the study period.

The second block lists the most frequent medicines and their frequencies in the matrix. For example, `fentanyl injection uh` appeared 211 times; that is $211/528 = 40$ of the (treatment) transactions. Since fentanyl is frequently used to help prevent pain after surgery or other medical procedure, we can see that many of these patients were going through some painful medical procedures.

The last block shows statistics about the size of the transaction. 248 patients had only one medicine in the study period, while 12 of them had 5 medication records one for each time point. On average, the patients are having 1.8 different medicines.

Visualizing Item Support: Item Frequency Plots

The summary might still be fairly abstract; let's visualize the data.

```
inspect(med[1:5,])
##      items
## [1] {acetaminophen uh,
##       cefazolin ivpb uh}
## [2] {docusate,
##       fioricet,
##       heparin injection,
##       ondansetron injection uh,
##       simvastatin}
## [3] {hydrocodone acetaminophen 5mg 325mg}
## [4] {fentanyl injection uh}
## [5] {cefazolin ivpb uh,
##       hydrocodone acetaminophen 5mg 325mg}
```

The `inspect()` call shows the transactional dataset. We can see that the medication records of each patient are nicely formatted as item-sets.

We can further analyze the frequent terms using `itemFrequency()`. This will show all item frequencies alphabetically ordered from the first five outputs (Fig. 12.1).

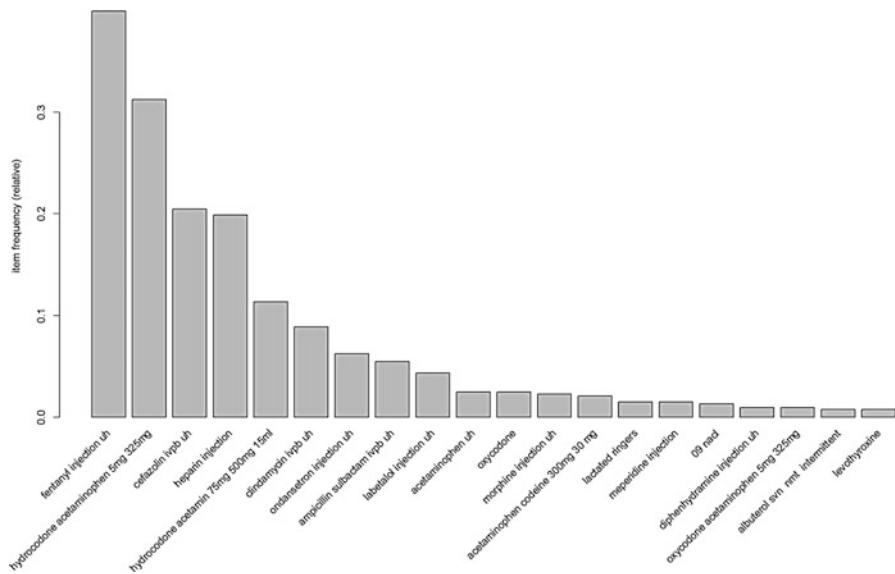


Fig. 12.1 Rank-order plot of item frequencies

```
itemFrequency(med[, 1:5])
##
##                                     09 nacl
##                                     0.013257576
##                                     09 nacl bolus
##                                     0.003787879
##     acetaminophen multiroute uh
##                                     0.001893939
## acetaminophen codeine 120 mg 12 mg 5 mL
##                                     0.001893939
##     acetaminophen codeine 300mg 30 mg
##                                     0.0208333333

itemFrequencyPlot(med, topN=20)
```

The above graph is showing us the top 20 medicines that are most frequently present in this dataset. Consistent with the prior `summary()` output, fentanyl is still the most frequent item. You can also try to plot the items with a threshold for support. Instead of `topN = 20`, just use the option `support = 0.1`, which will give you all the items have a support greater or equal to 0.1.

Visualizing Transaction Data: Plotting the Sparse Matrix

The sparse matrix will show what medications were prescribed for each patient (Fig. 12.2).

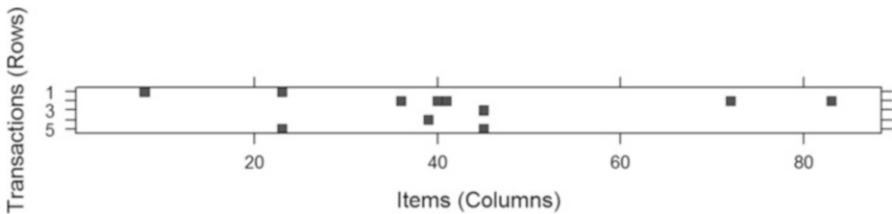
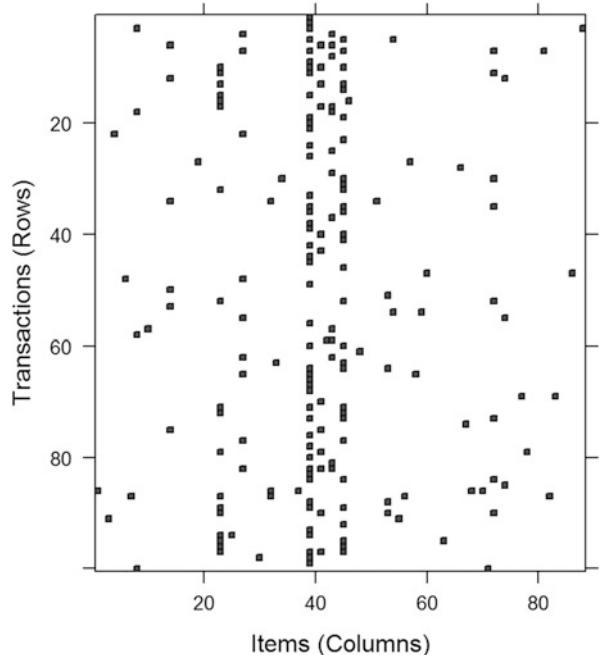


Fig. 12.2 A characteristic plot of the prescribed medications (columns) for the first 5 patients (rows)

Fig. 12.3 A characteristic plot of the prescribed medications (columns) for 100 random patients (rows)



```
image(med[1:5, ])
```

The image on Fig. 12.2 has 5 rows (we only requested the first 5 patients) and 88 columns (88 different medicines). Although the picture may be a little hard to interpret, it gives a sense of what kind of medicine is prescribed for each patient in the study.

Let's see an expanded graph including 100 randomly chosen patients (Fig. 12.3).

```
subset_int <- sample(nrow(med), 100, replace = F)
image(med[subset_int, ])
```

It shows us clearly that some medications are more popular than others. Now, let's fit the *Apriori* model.

12.6.3 Step 3: Training a Model on the Data

With the data in place, we can build the *association rules* using `apriori()` function.

```
myrules <- apriori(data=mydata, parameter=list(support=0.1, confidence=0.8,
minlen=1))
```

- Data: a sparse matrix created by `read.transactions()`.
- Support: minimum threshold for support.
- Confidence: minimum threshold for confidence.
- minlen: minimum required rule items (in our case, medications).

Setting up the threshold could be hard. You don't want it to be too high so that you get no rules or rules that everyone knows. You don't want to set it too low either, to avoid too many rules present. Let's see what we get under the default setting `support = 0.1, confidence = 0.8`:

```
apriori(med)
## Apriori
##
## Parameter specification:
##   confidence minval smax arem aval originalSupport maxtime support minlen
##   0.8      0.1     1 none FALSE           TRUE      5     0.1      1
##   maxlen target  ext
##   10    rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt Load sort verbose
##   0.1 TRUE TRUE FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 52
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[88 item(s), 528 transaction(s)] done [0.00s].
## sorting and recoding items ... [5 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
##
## set of 0 rules
```

Not surprisingly, we have 0 rules. The default setting is too high. In practice, we might need some time to fine-tune these thresholds, which may require certain familiarity with the underlying process or clinical phenomenon.

In this case study, we set `support = 0.1` and `confidence = 0.25`. This requires rules that have appeared in at least 10% of the head and neck cancer patients in the study. Also, the rules have to have least 25% accuracy. Moreover, `minlen = 2` would be a very helpful option because it removes all rules that have fewer than two items.

The results suggest we have a new `rules` object consisting of 29 rules.

```
med_rule<-apriori(med, parameter=List(support=0.01, confidence=0.25, minlen=2))

## Apriori
##
## Parameter specification:
##   confidence minval smax arem  aval originalSupport maxtime support minlen
##   0.25      0.1     1 none FALSE                  TRUE      5    0.01      2
##   maxLen target  ext
##   10   rules FALSE
##
## Algorithmic control:
##   filter tree heap memopt Load sort verbose
##   0.1 TRUE TRUE FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[88 item(s), 528 transaction(s)] done [0.00s].
## sorting and recoding items ... [16 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [29 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
```

`med_rule`

`## set of 29 rules`

12.6.4 Step 4: Evaluating Model Performance

First, we can obtain the overall summary of this set of rules.

```
summary(med_rule)

## set of 29 rules
##
## rule Length distribution (lhs + rhs):sizes
##  2 3 4
## 13 12 4
##
##   Min. 1st Qu. Median     Mean 3rd Qu.   Max.
##   2.00    2.00   3.00    2.69    3.00    4.00
##
## summary of quality measures:
##   support      confidence      lift
##   Min. :0.01136  Min. :0.2500  Min. :0.7583
##   1st Qu.:0.01705 1st Qu.:0.3390 1st Qu.:1.3333
##   Median :0.01894 Median :0.4444 Median :1.7481
##   Mean   :0.03448 Mean  :0.4491 Mean  :1.8636
##   3rd Qu.:0.03788 3rd Qu.:0.5000 3rd Qu.:2.2564
##   Max.   :0.11174 Max.  :0.8000 Max.  :3.9111
##
## mining info:
##   data ntransactions support confidence
##   med          528     0.01      0.25
```

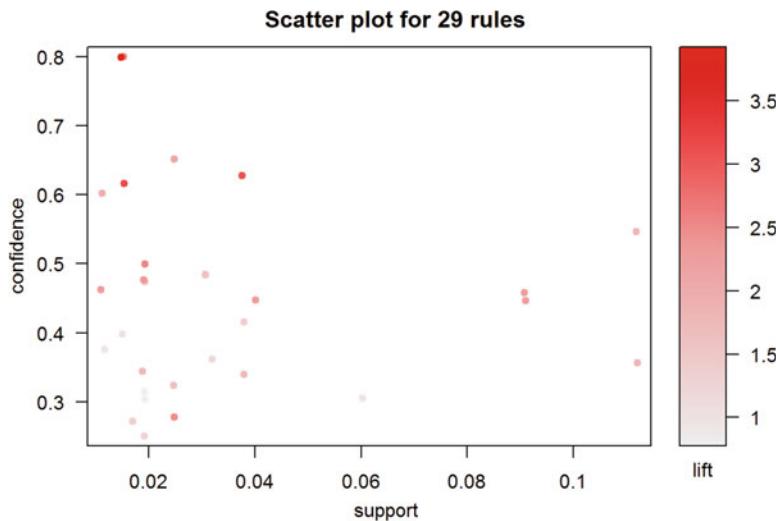


Fig. 12.4 Confidence-Support scatterplot of 29 rules

We have 13 rules that contain two items; 12 rules containing 3 items, and the remaining 4 rules contain 4 items.

The *lift* column shows how much more likely one medicine is to be prescribed to a patient given another medicine is prescribed. It is obtained by the following formula:

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}.$$

Note that $lift(X \rightarrow Y)$ is the same as $lift(Y \rightarrow X)$. The range of *lift* is $[0, \infty)$ and higher *lift* is better. We don't need to worry about support since we already set a threshold that the support will exceed.

Using the `aruleViz` package we can visualize the confidence and support scatter plots for all the rules (Fig. 12.4).

```
# install.packages("arulesViz")
library(arulesViz)
plot(sort(med_rule))
```

Again, we can utilize the `inspect()` function to see exactly what are these rules.

```
inspect(med_rule[1:3])
##      Lhs                  rhs                  support
confidence      lift
## [1] {acetaminophen uh}      => {cefazolin ivpb uh} 0.01136364
0.4615385 2.256410
## [2] {ampicillin sulbactam ivpb uh} => {heparin injection} 0.01893939
0.3448276 1.733990
## [3] {ondansetron injection uh}      => {heparin injection} 0.01704545
0.2727273 1.371429
```

Here, `lhs` and `rhs` refer to “left hand side” and “right hand side” of the rule, respectively. `Lhs` is the given condition and `rhs` is the predicted result. Using the first row as an example: If a head-and-neck patient has been prescribed acetaminophen (pain reliever and fever reducer), it is likely that the patient is also prescribed cefazolin (antibiotic that resist bacterial infections); bacterial infections are associated with fevers and some cancers.

12.6.5 Step 5: Improving Model Performance

Sorting the Set of Association Rules

Sorting the resulting association rules corresponding to high `lift` values will help us select the most useful rules.

```
inspect(sort(med_rule, by="lift")[1:3])
##      Lhs                  rhs
support confidence      lift
## [1] {fentanyl injection uh,
##      heparin injection,
##      hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.01515152 0.8000000 3.911111
## [2] {cefazolin ivpb uh,
##      fentanyl injection uh,
##      hydrocodone acetaminophen 5mg 325mg} => {heparin injection}
0.01515152 0.6153846 3.094505
## [3] {heparin injection,
##      hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.03787879 0.6250000 3.055556
```

These rules may need to be interpreted by clinicians and experts in the specific context of the study. For instance, the first row, `{fentanyl, heparin, hydrocodone acetaminophen}` implies `{cefazolin}`. Fentanyl and hydrocodone acetaminophen are both pain relievers that may be prescribed after surgery. `Heparin` is usually used before surgery to reduce the risk of blood clots. This rule may suggest that patients who have undergone surgical treatments may likely need cefazolin to prevent post-surgical bacterial infection.

Taking Subsets of Association Rules

If we are more interested in investigating associations that are linked to a specific medicine, we can narrow the rules down by making subsets. Let us try investigating rules related to fentanyl, since it appears to be the most frequently prescribed medicine.

```
fi_rules<-subset(med_rule, items %in% "fentanyl injection uh")
inspect(fi_rules)

##      lhs                                rhs
## support confidence      lift
## [1] {ondansetron injection uh}      => {fentanyl injection uh}
## 0.01893939 0.3030303 0.7582938
## [2] {fentanyl injection uh,          => {hydrocodone acetaminophen
##      ondansetron injection uh}      5mg 325mg} 0.01136364 0.6000000 1.9200000
## [3] {hydrocodone acetaminophen 5mg 325mg,      => {fentanyl injection uh}
##      ondansetron injection uh}      0.01136364 0.3750000 0.9383886
## [4] {cefazolin ivpb uh,              => {heparin injection}
##      fentanyl injection uh}      0.01893939 0.5000000 2.5142857
## [5] {fentanyl injection uh,          => {cefazolin ivpb uh}
##      heparin injection}          0.01893939 0.4761905 2.3280423
## [6] {cefazolin ivpb uh,              => {hydrocodone acetaminophen
##      fentanyl injection uh}      5mg 325mg} 0.02462121 0.6500000 2.0800000
## [7] {fentanyl injection uh,          => {cefazolin ivpb uh}
##      hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
## 0.02462121 0.3250000 1.5888889
## [8] {fentanyl injection uh,          => {hydrocodone acetaminophen
##      heparin injection}          5mg 325mg} 0.01893939 0.4761905 1.5238095
## [9] {heparin injection,              => {fentanyl injection uh}
##      hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
## 0.01893939 0.3125000 0.7819905
## [10] {fentanyl injection uh,          => {heparin injection}
##      hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
## 0.01893939 0.2500000 1.2571429
## [11] {cefazolin ivpb uh,              => {hydrocodone acetaminophen
##      fentanyl injection uh,          5mg 325mg} 0.01515152 0.8000000 2.5600000
##      heparin injection}          0.01515152 0.4000000 1.0009479
## [12] {cefazolin ivpb uh,              => {fentanyl injection uh}
##      heparin injection,          0.01515152 0.6153846 3.0945055
##      hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
## [13] {cefazolin ivpb uh,              => {heparin injection}
##      fentanyl injection uh,          0.01515152 0.6153846 3.0945055
##      hydrocodone acetaminophen 5mg 325mg} => {heparin injection}
## [14] {fentanyl injection uh,          => {cefazolin ivpb uh}
##      heparin injection,          0.01515152 0.8000000 3.9111111
##      hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
```

In R scripting, the notation `%in%` signifies “belongs to.” There are 14 rules related to this item. Let’s plot them (Fig. 12.5).

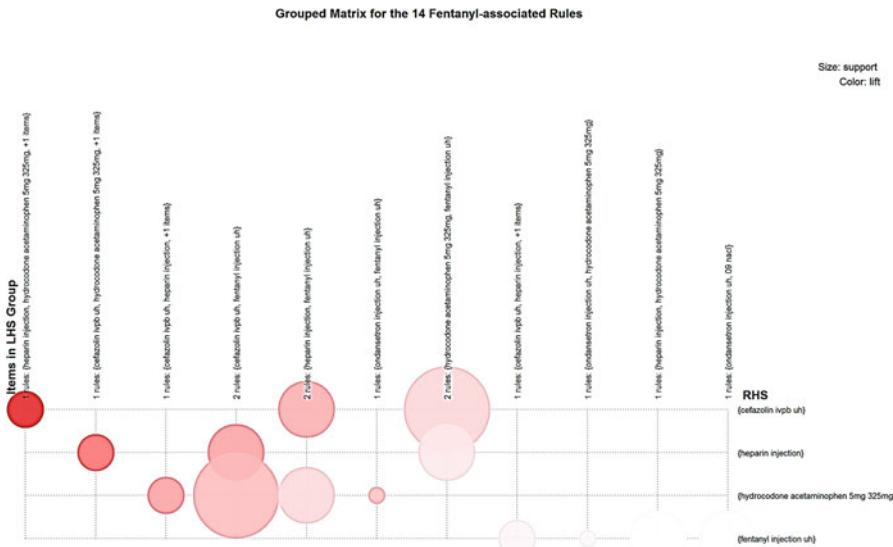


Fig. 12.5 Bubble chart of the grouped matrix for 14 rules

```
plot(sort(fi_rules, by="lift"), method="grouped", control=list(type="items"),
, main = "Grouped Matrix for the 14 Fentanyl-associated Rules")

## Available control parameters (with default values):
## main = Grouped Matrix for 14 Rules
## k     = 20
## rhs_max = 10
## lhs_items = 2
## aggr.fun = function (x, na.rm = FALSE) UseMethod("median")
## col = c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF",
## "#EE0F0FFF", "#EE1212FF", "#EE1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1E
## FF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF", "#EE31
## 31FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE
## 4444FF", "#EE4747FF", "#EE4A4AFF", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#E
## E5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6666FF",
## "#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF", "#EE7878FF
## ", "#EE7B7BFF", "#EE7E7EFF", "#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8B
## FF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999FF", "#EE9B
## 9BFF", "#EE9D9DFF", "#EE999FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EE
## A5A5FF", "#EEA7A7FF", "#EEA9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#E
## EB0B0FFF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF", "#EEB8B8FF",
## "#EEBABAFF", "#EEBCBCFF", "#EEBDDBFF", "#EEBFBBBB", "#EEC1C1FF", "#EEC3C3FF
## ", "#EEC4C4FF", "#EEC6C6FF", "#EEC8C8FF", "#EEC9C9FF", "#EECBCBFF", "#EECDCD
## FF", "#EECFCCFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EED5D5FF", "#EED7
## D7FF", "#EED9D9FF", "#EEDBDDBF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EE
## E1E1FF", "#EEE3E3FF", "#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#E
## EECCECF", "#EEEEEEFF")
## reverse = TRUE
## xlab = NULL
## ylab = NULL
## Legend = Size: support Color: Lift
## spacing = -1
```

```
## panel.function = function (row, size, shading, spacing) { size[s
ize == 0] <- NA shading[is.na(shading)] <- 1 grid.circle(x = c(1:Len
gth(size)), y = row, r = size/2 * (1 - spacing), default.units = "native", g
p = gpar(fill = shading, col = shading, alpha = 0.9)) }
## gp_main = List(cex = 1.2, fontface = "bold", font = 2)
## gp_labels = List(cex = 0.8)
## gp_labs = List(cex = 1.2, fontface = "bold", font = 2)
## gp_lines = List(col = "gray", lty = 3)
## newpage = TRUE
## interactive = FALSE
## max.shading = NA
## verbose = FALSE
```

Saving Association Rules to a File or Data Frame

We can save these rules into a CSV file using `write()`. It is similar with the function `write.csv()` that we have mentioned in the beginning of this case study.

```
write(med_rule, file = "medrule.csv", sep=",", row.names=F)
```

Sometimes it is more convenient to convert the rules into a data frame.

```
med_df<-as(med_rule, "data.frame")
str(med_df)

## 'data.frame': 29 obs. of 4 variables:
## $ rules : Factor w/ 29 Levels "{acetaminophen uh} => {cefa
## $ support : num 0.0114 0.0189 0.017 0.0189 0.0303 ...
## $ confidence: num 0.462 0.345 0.273 0.303 0.485 ...
## $ lift : num 2.256 1.734 1.371 0.758 1.552 ...
```

As we can see, the rules are converted into a factor vector.

12.7 Practice Problems: Groceries

In this practice problem, we will investigate the associations of frequently purchased groceries using the *grocery* dataset in the R base. Firstly, let's load the data.

```
data("Groceries")
summary(Groceries)

## transactions as itemMatrix in sparse format with
## 9835 rows (elements/itemsets/transactions) and
## 169 columns (items) and a density of 0.02609146
##
## most frequent items:
##      whole milk other vegetables      rolls/buns      soda
##      2513      1903      1809      1715
##      yogurt      (Other)      34055
```

```

## element (itemset/transaction) Length distribution:
## sizes
##   1   2   3   4   5   6   7   8   9   10  11  12  13  14  1
## 5
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 5
## 5
##   16  17  18  19  20  21  22  23  24  26  27  28  29  32
## 46  29  14  14   9  11   4   6   1   1   1   1   3   1
## 
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.
## 1.000 2.000 3.000 4.409 6.000 32.000
## 
## includes extended item information - examples:
##       labels level2      level1
## 1 frankfurter sausage meat and sausage
## 2 sausage sausage meat and sausage
## 3 Liver Loaf sausage meat and sausage

```

We will try to find out the top 5 frequent grocery items and plot them (Fig. 12.6).

Then, try to use `support = 0.006`, `confidence = 0.25`, `minlen = 2` to set up the grocery association rules. Sort the top 3 rules with highest lift.

```

## Apriori
##
## Parameter specification:
##   confidence minval smax arem aval originalSupport maxtime support minlen
##   0.25      0.1    1 none FALSE           TRUE      5  0.006      2
##   maxLen target  ext
##   10  rules FALSE
## 

```

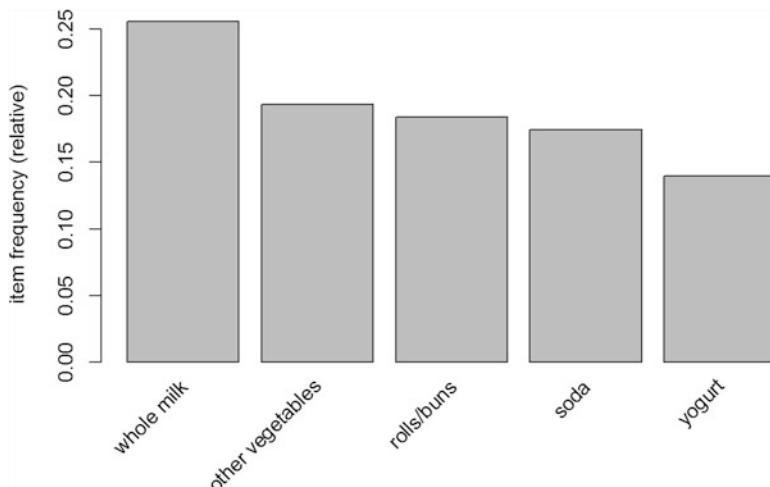


Fig. 12.6 Top-5 grocery items according to their frequencies

```

## Algorithmic control:
## filter tree heap memopt Load sort verbose
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE
##
## Absolute minimum support count: 59
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [463 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
## set of 463 rules

##      lhs                      rhs                      support confidence
lift
## [1] {herbs}          => {root vegetables} 0.007015760 0.4312500
3.956477
## [2] {berries}        => {whipped/sour cream} 0.009049314 0.2721713
3.796886
## [3] {tropical fruit,
##      other vegetables,
##      whole milk}       => {root vegetables} 0.007015760 0.4107143
3.768074

```

The number of rules (463) appears excessive. We can try stringer parameters. In practice, it's more possible to observe underlying rules if you set a higher confidence. Here we set the *confidence* = 0.6.

```

groceryrules <- apriori(Groceries, parameter = list(support = 0.006, confidence = 0.6, minlen = 2))

## Apriori
## Parameter specification:
## confidence minval smax arem aval originalSupport maxtime support minlen
## 0.6 0.1 1 none FALSE TRUE 5 0.006 2
## maxlen target ext
## 10 rules FALSE
##
## Algorithmic control:
## filter tree heap memopt Load sort verbose
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE
##
## Absolute minimum support count: 59
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
groceryrules
## set of 8 rules

```

Fig. 12.7 Live demo: association rule mining

The screenshot shows the rdrr.io live demo interface for the arules package. The URL is <https://rdrr.io/cran/arules/>. The interface has a top navigation bar with 'rdrr.io', 'R Docs', 'Browse Packages', 'Run R in your browser', 'Free R', 'Apply Notebooks', and a search bar. Below the navigation is a sidebar with links like 'arules', 'Mining Association Rules and Frequent Itemsets', 'Package Index', and 'Vignettes'. The main content area contains R code for mining association rules from a dataset named 'groceryrules'. The code includes a 'Run' button and a preview of the resulting rules. A note at the bottom says 'You should assume that any scripts or data that you put into this service are public.' and 'Read more about your data's privacy and security here.'

```
inspect(sort(groceryrules, by = "lift")[1:3])

##      lhs                  rhs          support      confidence
## [1] {butter,whipped/sour cream} => {whole milk} 0.006710727 0.6600000
## [2] {butter,yogurt}           => {whole milk} 0.009354347 0.6388889
## [3] {root vegetables,butter}  => {whole milk} 0.008235892 0.6377953
##      Lift
## [1] 2.583008
## [2] 2.500387
## [3] 2.496107
```

We observe mainly rules between dairy products. It makes sense that customers pick up milk when they walk down the dairy products isle. Experiment further with various parameter settings and try to interpret the results in the context of this grocery case-study (Fig. 12.7).

Mining association rules Demo <https://rdrr.io/cran/arules/>

```
# copy-paste this R code into the Live online demo:
# https://rdrr.io/snippets/
# press RUN, and examine the results.
# The HYPERLINK "https://archive.ics.uci.edu/ml/datasets/adult" Adult dataset includes 48842 sparse transactions
# (rows) and 115 items (columns).
library(arules)
data("Adult")
rules <- apriori(Adult,
                  parameter = list(supp = 0.5, conf = 0.9, target = "rules"))
summary(rules)
inspect(sort(rules, by = "lift")[1:3])

# Results: mining info:
# data ntransactions support confidence
# Adult 48842 0.5 0.9
# lhs rhs support confidence lift count
# [1] {sex=Male, native-country=United-States} => {race=White} 0.5415421 0.9051090 1.058554 26450
# [2] {sex=Male, capital-loss=None, native-country=United-States} => {race=White} 0.5113632 0.9032585 1.056390 24976
# [3] {race=White} => {native-country=United-States} 0.7881127 0.9217231 1.027076 38493
```

12.8 Summary

- The Apriori algorithm for association rule learning is only suitable for large transactional data. For some small datasets, it might not be very helpful.
- It is useful for discovering associations, mostly in early phases of an exploratory study.

- Some rules can be built due to chance and may need further verifications.
- See also Chap. 20 (Text Mining and NLP).

Try to replicate these results with other data from the list of our Case-Studies.

12.9 Assignments: 12. Apriori Association Rules Learning

Use the SOCR Jobs Data to practice learning via Apriori Association Rules

- Load the Jobs Data. Use this guide to load HTML data.
- Focus on the Description feature. Replace all underscore characters “_” with spaces.
- Review Chap. 8, use tm package to process text data to plain text. (Hint: need to apply stemDocument as well, we will discuss more details in Chap. 20.)
- Generate a “transaction” matrix by considering each job as one record and description words as “transaction” items. (Hint: You need to fill missing values since records do not have the same length of description.)
- Save the data using write.csv() and then use read.transactions() in arules package to read the CSV data file. Visualize the item support using item frequency plots. What terms appear as more popular?
- Fit a model: myrules <- apriori(data = jobs, parameter = list(support = 0.02, confidence = 0.6, minlen = 2)). Try out several rule thresholds trading off gain and accuracy.
- Evaluate the rules you obtained with lift and visualize their metrics.
- Mine medical related rules (e.g., rules include “treatment”, “patient”, “care”, “diagnos.” Notice that these are word stems).
- Sort the set of association rules for all and medical related subsets.
- Save these rules into a CSV file.

References

- Witten, IH, Frank, E, Hall. MA. (2011) *Data Mining: Practical Machine Learning Tools and Techniques*, The Morgan Kaufmann Series in Data Management Systems, Elsevier, ISBN 0080890369, 9780080890364.
- Soh, PJ, Woo, WL, Sulaiman, HA, Othman, MA, Saat, MS (eds). (2016) *Advances in Machine Learning and Signal Processing: Proceedings of MALSIP 2015*, Volume 387 of Lecture Notes in Electrical Engineering, Springer, ISBN 3319322133, 9783319322131.

Chapter 13

k-Means Clustering



As we learned in Chaps. 7, 8, and 9, classification could help us make predictions on new observations. However, classification requires (human supervised) predefined label classes. What if we are in the early phases of a study and/or don't have the required resources to manually define, derive or generate these class labels?

Clustering can help us explore the dataset and separate cases into groups representing similar traits or characteristics. Each group could be a potential candidate for a class. Clustering is used for exploratory data analytics, i.e., as *unsupervised learning*, rather than for confirmatory analytics or for predicting specific outcomes.

In this chapter, we will present (1) clustering as a machine learning task, (2) the *silhouette* plots for classification evaluation, (3) the *k-Means* clustering algorithm and how to tune it, (4) examples of several interesting case-studies, including Divorce and Consequences on Young Adults, Pediatric Trauma, and Youth Development, (5) demonstrate hierarchical clustering, and (6) Gaussian mixture modeling.

13.1 Clustering as a Machine Learning Task

As we mentioned before, clustering represents classification of unlabeled cases. Scatter plots depict a simple illustration of the clustering process. Assume we don't know much about the ingredients of frankfurter hot dogs and we have the following graph (Fig. 13.1).

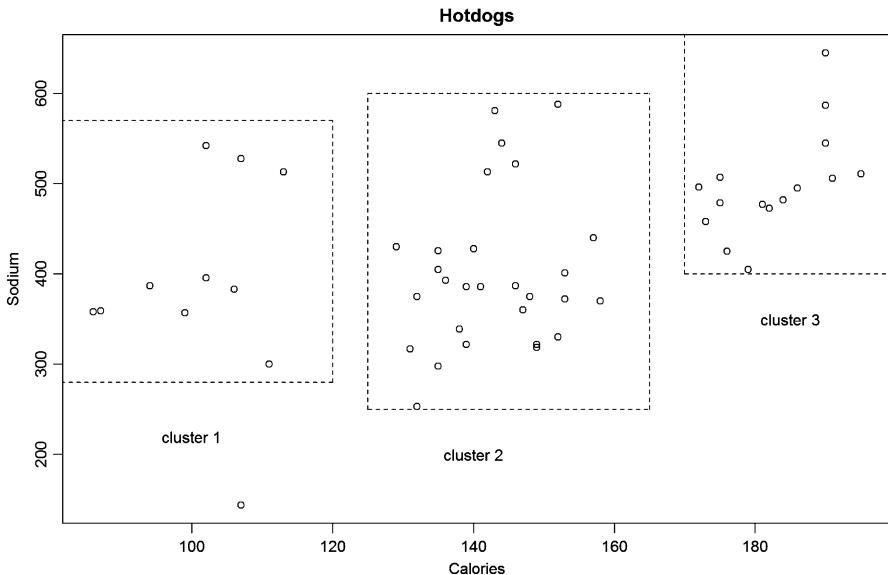


Fig. 13.1 Hotdogs dataset – scatterplot of calories and sodium content blocked by type of meat

```
# See Chapter 12 code for complete details
# install.packages("rvest")
library(rvest)
wiki_url<-
read_html("http://wiki.socr.umich.edu/index.php/SOCR_012708_ID_Data_HotDogs")
)
html_nodes(wiki_url, "#content")
hotdog<- html_table(html_nodes(wiki_url, "table"))[[1]]
plot(hotdog$Calories, hotdog$Sodium, main = "Hotdogs", xLab="Calories",
yLab="Sodium")
```

In terms of calories and sodium, these hot dogs are clearly separated into three different clusters. *Cluster 1* has hot dogs of low calories and medium sodium content; *Cluster 2* has both calorie and sodium at medium levels; *Cluster 3* has both sodium and calories at high levels. We can make a bold guess about the meats used in these three clusters of hot dogs. For Cluster 1, it could be mostly chicken meat since it has low calories. The second cluster might be beef, and the third one is likely to be pork, because beef hot dogs have considerably less calories and salt than pork hot dogs. However, this is just guessing. Some hot dogs have a mixture of two or three types of meat. The real situation is somewhat similar to what we guessed but with some random noise, especially in Cluster 2.

The following two plots show the primary type of meat used for each hot dog labeled by name (top) and color-coded (bottom) (Figs. 13.2 and 13.3).

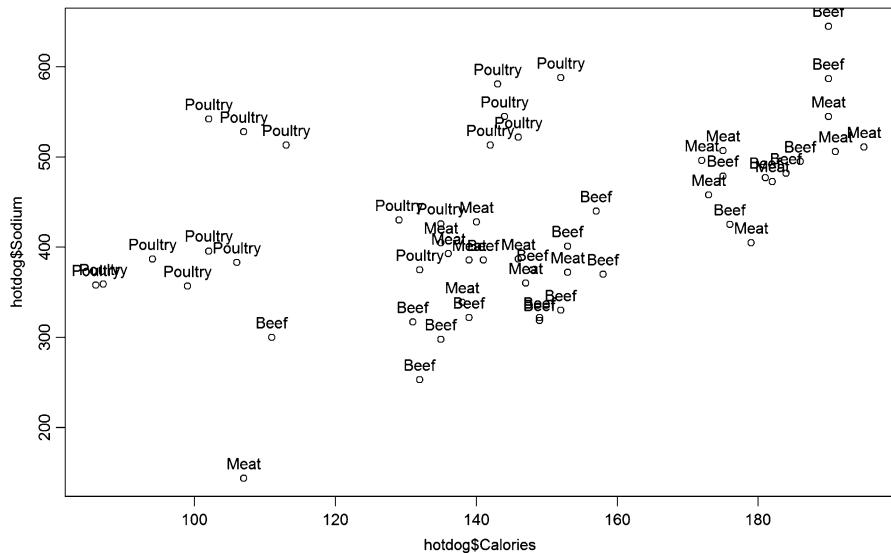


Fig. 13.2 Scatterplot of calories and sodium content with meat type labels

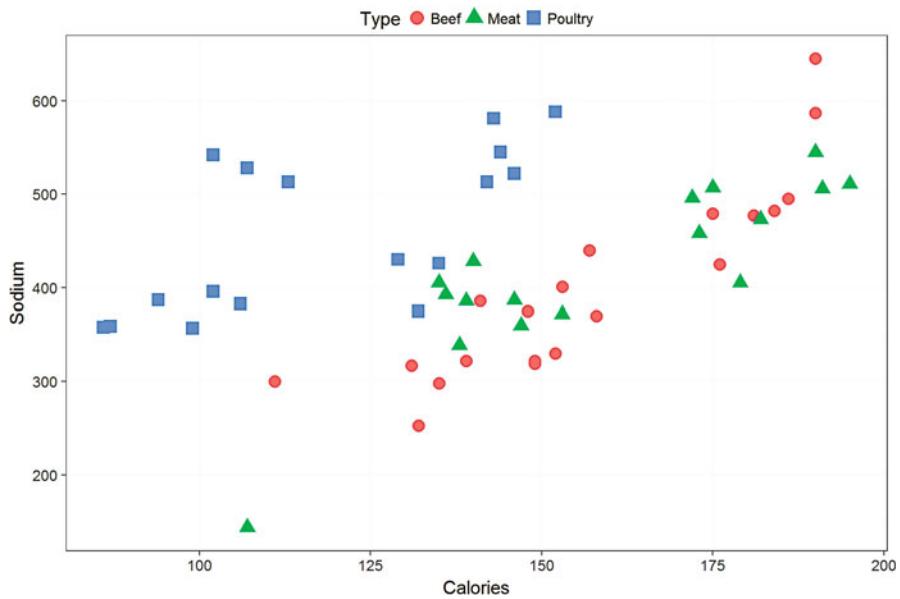


Fig. 13.3 An alternative scatterplot of the hotdogs calories and sodium

13.2 Silhouette Plots

Silhouette plots are useful for interpretation and validation of consistency of all clustering algorithms. The silhouette value, $\in [-1, 1]$, measures the similarity (cohesion) of a data point to its cluster relative to other clusters (separation). Silhouette plots rely on a distance metric, e.g., the Euclidean distance, Manhattan distance, Minkowski distance, etc.

- High silhouette value suggest that the data matches its own cluster well.
- A clustering algorithm performs well when most Silhouette values are high.
- Low value indicates poor matching within the neighboring cluster.
- Poor clustering may imply that the algorithm configuration may have too many or too few clusters.

Suppose a clustering method groups all data points (objects), $\{X_i\}_i$, into k clusters and define:

- d_i as the *average dissimilarity* of X_i with all other data points within its cluster. d_i captures the quality of the assignment of X_i to its current class label. Smaller or larger d_i values suggest better or worse overall assignment for X_i to its cluster, respectively. The average dissimilarity of X_i to a cluster C is the average distance between X_i and all points in the cluster of points labeled C .
- l_i as the *lowest average dissimilarity* of X_i to any other cluster, that X_i is not a member of. The cluster corresponding to l_i , the lowest average dissimilarity, is called the X_i **neighboring cluster**, as it is the next best fit cluster for X_i .

Then, the **silhouette** of X_i is defined by:

$$-1 \leq s_i = \frac{l_i - d_i}{\max\{l_i, d_i\}} \equiv \begin{cases} 1 - \frac{d_i}{l_i}, & \text{if } d_i < l_i \\ 0, & \text{if } d_i = l_i \\ \frac{l_i}{d_i} - 1, & \text{if } d_i > l_i \end{cases}$$

Note that:

- $-1 \leq s_i \leq 1$,
- $s_i \rightarrow 1$ when $d_i \ll l_i$, i.e., the dissimilarity of X_i to its cluster, C is much lower relative to its dissimilarity to other clusters, indicating a good (cluster assignment) match. Thus, high Silhouette values imply the data is appropriately clustered.
- Conversely, $-1 \leftarrow s_i$ when $l_i \ll d_i$, d_i is large, implying a poor match of X_i with its current cluster C , relative to neighboring clusters. X_i may be more appropriately clustered in its neighboring cluster.
- $s_i \sim 0$ means that the X_i may lie on the border between two natural clusters.

13.3 The k-Means Clustering Algorithm

The *k-means* algorithm is one of the most commonly used algorithms for clustering.

13.3.1 Using Distance to Assign and Update Clusters

This algorithm is similar to *k-nearest neighbors (KNN)* presented in Chap. 7. In clustering, we don't have a priori pre-determined labels, and the algorithm is trying to deduce intrinsic groupings in the data.

Similar to KNN, k-means uses Euclidean distance ($\|\cdot\|_2$ norm) most of the times, however Manhattan distance ($\|\cdot\|_1$ norm), or the more general Minkowski distance $\left(\left(\sum_{i=1}^n |p_i - q_i|^c\right)^{\frac{1}{c}}\right)$ may also be used. For $c = 2$, the Minkowski distance represents the classical Euclidean distance:

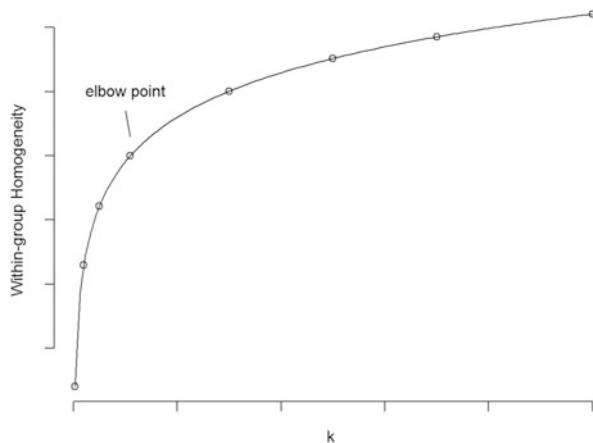
$$dist(x, y) = \sqrt{\sum_{n=1}^n (x_i - y_i)^2}.$$

How can we separate clusters using this formula? The *k-means* protocol is as follows:

- *Initiation*: First, we define k points as cluster centers. Often these points are k random points from the dataset. For example, if $k = 3$, we choose three random points in the dataset as cluster centers.
- *Assignment*: Second, we determine the maximum extent of the cluster boundaries that all have maximal distance from their cluster centers. Now the data is separated into k initial clusters. The assignment of each observation to a cluster is based on computing the least within-cluster sum of squares according to the chosen distance. Mathematically, this is equivalent to Voronoi tessellation of the space of the observations according to their mean distances.
- *Update*: Third, we update the centers of our clusters to new *means* of the cluster centroid locations. This updating phase is the essence of the *k-means* algorithm.

Although there is no guarantee that the *k-means* algorithm converges to a global optimum, in practice, the algorithm tends to converge, i.e., the assignments no longer change, to a local minimum as there are only a finite number of such Voronoi partitionings.

Fig. 13.4 Elbow plot of the within-group homogeneity against the number of groups parameter (k)



13.3.2 Choosing the Appropriate Number of Clusters

We don't want our number of clusters to be either too large or too small. If it is too large, the groups are too specific to be meaningful. On the other hand, too few groups might be too broadly general to be useful. As we mentioned in Chap. 7, $k = \sqrt{\frac{n}{2}}$ is a good place to start. However, it might generate a large number of groups. Also, the elbow method may be used to determine the relationship of k and homogeneity of the observations of each cluster. When we graph within-group homogeneity against k , we can find an “elbow point” that suggests a minimum k corresponding to relatively large within-group homogeneity (Fig. 13.4).

This graph shows that homogeneity barely increases above the “elbow point”. There are various ways to measure homogeneity within a cluster. For detailed explanations please read *On clustering validation techniques, Journal of Intelligent Information Systems* Vol. 17, pp. 107–145, by M. Halkidi, Y. Batistakis, and M. Vazirgiannis (2001).

13.4 Case Study 1: Divorce and Consequences on Young Adults

13.4.1 Step 1: Collecting Data

The dataset we will be using is the Divorce and Consequences on Young Adults dataset. This is a longitudinal study focused on examining the consequences of recent parental divorce for young adults (initially ages 18–23) whose parents had divorced within 15 months of the study's first wave (1990–91). The sample consisted of 257 White respondents with newly divorced parents. Here we have a subset of this dataset with 47 respondents in our case-studies folder, CaseStudy01_Divorce_YoungAdults_Data.csv.

Variables

- **DIVYEAR:** Year in which parents were divorced. Dichotomous variable with 1989 and 1990.
- **Child affective relations:**
 - Momint: Mother intimacy. Interval level data with four possible responses (1-extremely close, 2-quite close, 3-fairly close, 4- not close at all).
 - Dadint: Father intimacy. Interval level data with four possible responses (1-extremely close, 2-quite close, 3-fairly close, 4-not close at all).
 - Live with mom: Polytomous variable with three categories (1- mother only, 2-father only, 3- both parents).
- **momclose:** measure of how close the child is to the mother (1-extremely close, 2-quite close, 3-fairly close, 4-not close at all).
- **Depression:** Interval level data regarding feelings of depression in the past 4 weeks. Possible responses are 1-often, 2-sometimes, 3-hardly ever, 4-never.
- **Gethitched:** Polytomous variable with four possible categories indicating respondent's plan for marriage (1-Marry fairly soon, 2-marry sometime, 3-never marry, 8-don't know).

13.4.2 Step 2: Exploring and Preparing the Data

Let's load the dataset and pull out a summary of all variables.

```
divorce<-read.csv("https://umich.instructure.com/files/399118/download?download_frd=1")
summary(divorce)

##      DIVYEAR          momint          dadint          momclose
##  Min.   :89.00   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:89.00   1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000
##  Median :90.00   Median :1.000   Median :2.000   Median :2.000
##  Mean   :89.68   Mean   :1.809   Mean   :2.489   Mean   :1.809
##  3rd Qu.:90.00   3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:2.000
##  Max.   :90.00   Max.   :4.000   Max.   :4.000   Max.   :4.000
##      depression      livewithmom      gethitched
##  Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :3.000   Median :1.000   Median :2.000
##  Mean   :2.851   Mean   :1.489   Mean   :2.213
##  3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :4.000   Max.   :9.000   Max.   :8.000
```

According to the summary, DIVYEAR is actually a dummy variable (either 89 or 90). We can recode (binarize) the DIVYEAR using the `ifelse()` function (mentioned in Chap. 8). The following line of code generates a new indicator variable for *divorce year = 1990*.

```
divorce$DIVYEAR<-ifelse(divorce$DIVYEAR==89, 0, 1)
```

We also need another preprocessing step to deal with `livewithmom`, which has missing values, `livewithmom = 9`. We can impute these using `momint` and `dadint` variables for each specific participant

```
table(divorce$livewithmom)

##
##  1  2  9
## 31 15  1



```

For instance, respondents that feel much closer to their dads may be assigned `divorce$livewithmom==2`, suggesting they most likely live with their fathers. Of course, alternative imputation strategies are also possible.

```
divorce[45, 6]<-2


```

13.4.3 Step 3: Training a Model on the Data

We are only using R base functionality, so no need to install any additional packages now, however `library(stats)` may still be necessary. Then, the function `kmeans()` will provide the *k-means* clustering of the data.

```
myclusters<-kmeans(mydata, k)
```

- *mydata*: dataset in a matrix form.
- *k*: number of clusters we want to create.
- *output*:
 - `myclusters$cluster`: vector indicating the cluster number for every observation.
 - `myclusters$center`: a matrix showing the mean feature values for every center.
 - `mycluster$size`: a table showing how many observations are assigned to each cluster.

Before we perform clustering, we need to standardize the features to avoid biasing the clustering based on features that use large-scale values. Note that distance calculations are sensitive to measuring units. The method `as.data.frame()` will convert our dataset into a data frame allowing us to use the `lapply()` function. Next, we use a combination of `lapply()` and `scale()` to standardize our data.

```
di_z<- as.data.frame(lapply(divorce, scale))
str(di_z)

## 'data.frame': 47 obs. of 7 variables:
## $ DIVYEAR : num 0.677 0.677 -1.445 0.677 -1.445 ...
## $ momint : num 1.258 1.258 -0.854 1.258 -0.854 ...
## $ dadint : num -0.514 -0.514 0.536 1.586 0.536 ...
## $ momclose : num 0.225 1.401 -0.951 1.401 0.225 ...
## $ depression : num 0.164 -0.937 1.265 0.164 -2.038 ...
## $ Livewithmom: num -0.711 1.377 -0.711 -0.711 -0.711 ...
## $ gethitched : num 0.846 -0.229 -0.229 0.846 -0.229 ...
```

The resulting dataset, `di_z`, is standardized so all features are unitless and follow approximately standardized normal distribution.

Next, we need to think about selecting a proper k . We have a relatively small dataset with 47 observations. Obviously we cannot have a k as large as 10. The rule of thumb suggests $k = \sqrt{47/2} = 4.8$. This would be relatively large also because we will have less than 10 observations for each cluster. It is very likely that for some clusters we only have one observation. A better choice may be 3. Let's see if this will work.

```
library(stats)
set.seed(321)
diz_clussters<-kmeans(di_z, 3)
```

13.4.4 Step 4: Evaluating Model Performance

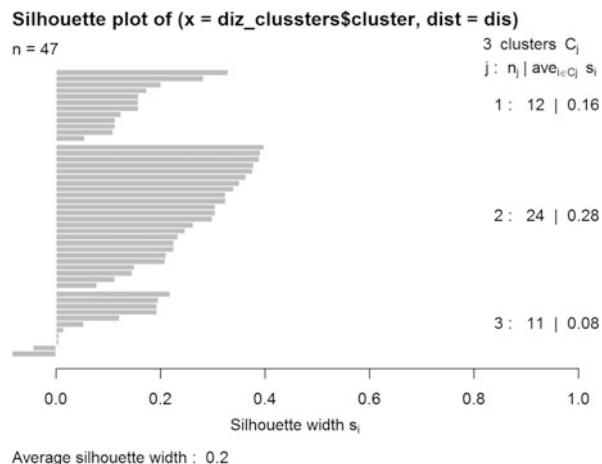
Let's look at the clusters created by the *k-means* model.

```
diz_clussters$size
## [1] 12 24 11
```

At first glance, it seems that 3 worked well for the number of clusters. We don't have any cluster that contains a small number of observations. The three clusters have relatively equal number of respondents.

Silhouette plots represent the most appropriate evaluation strategy to assess the quality of the clustering. Silhouette values are between -1 and 1 . In our case, two data points correspond to negative Silhouette values, suggesting these cases may be "mis-clustered" or perhaps are ambiguous, as the Silhouette value is close to 0. We can observe that the average Silhouette is reasonable, about 0.2 (Fig. 13.5).

Fig. 13.5 Silhouette plot for the 3 classes



```
require(cluster)
dis = dist(di_z)
sil = silhouette(diz_clussters$cluster, dis)
summary(sil)

## Silhouette of 47 units in 3 clusters from silhouette.default(x = diz_clussters$cluster, dist = dis) :
##  Cluster sizes and average silhouette widths:
##    12      24      11
## 0.16444649 0.27684356 0.07921684
## Individual silhouette widths:
##   Min. 1st Qu. Median Mean 3rd Qu. Max.
## -0.08466 0.11760 0.20080 0.20190 0.30450 0.39820

plot(sil)
```

The next step would be to interpret the clusters in the context of this social study.

```
diz_clussters$centers

##      DIVYEAR      momint      dadint      momclose depression Livewithmom
## 1  0.5004720  1.1698438 -0.07631029  1.2049200 -0.1112567  0.1591755
## 2 -0.2953914 -0.5016290  0.36107795 -0.5096937  0.1180883 -0.7107373
## 3  0.0985208 -0.1817299 -0.70455885 -0.2023993 -0.1362761  1.3770536
##  gethitched
## 1 -0.1390230
## 2 -0.1390230
## 3  0.4549845
```

This result shows:

- *Cluster 1:* divyear = mostly 90, momint = very close, dadint = not close, livewithmom = mostly mother, depression = not often, (gethitched) marry = will likely not get married. Cluster 1 represents mostly adolescents that are closer to mom than dad. These young adults do not often feel depressed and they may

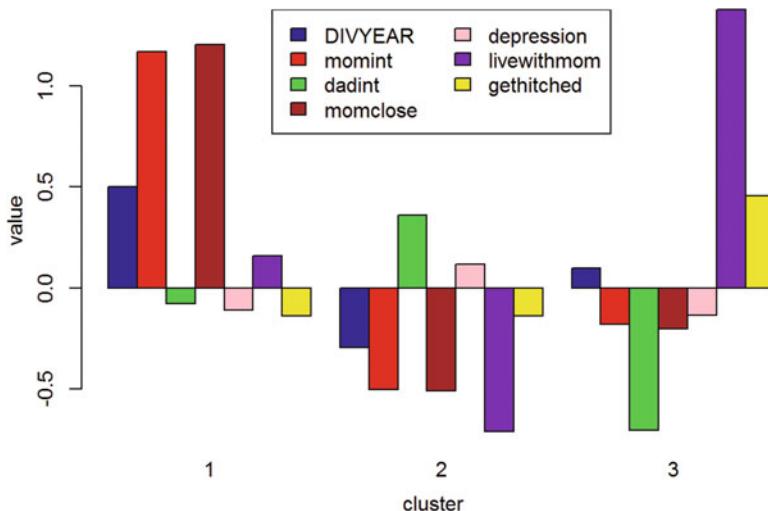


Fig. 13.6 Barplot illustrating the features discriminating between the three cohorts in the divorce consequences on young adults dataset

avoid getting married. These young adults tends to be not be too emotional and do not value family.

- *Cluster 2*: divyear = mostly 89, momint = not close, dadint = very close, livewithmom = father, depression = mild, marry = do not know/not inclined. Cluster 2 includes children that mostly live with dad and only feel close to dad. These people don't felt severely depressed and are not inclined to marry. These young adults may prefer freedom and tend to be more naive.
- *Cluster 3*: divyear = mix of 89 and 90, momint = not close, dadint = not at all, livewithmom = mother, depression = sometimes, marry = tend to get married. Cluster 3 contains children that did not feel close to either dad or mom. They sometimes felt depressed and are willing to build their own family. These young adults seem to be more independent.

We can see that these three different clusters do contain three alternative types of young adults. Bar plots provide an alternative strategy to visualize the difference between clusters (Fig. 13.6).

```
par(mfrow=c(1, 1), mar=c(4, 4, 4, 2))
myColors <- c("darkblue", "red", "green", "brown", "pink", "purple", "yellow")
barplot(t(diz_crussters$centers), beside = TRUE, xlab="cluster",
ylab="value", col = myColors)
Legend("topleft", ncol=2, Legend = c("DIVYEAR", "momint", "dadint",
"momclose", "depression", "livewithmom", "gethitched"), fill = myColors)
```

For each of the three clusters, the bars in the plot above represent the following order of features DIVYEAR, momint, dadint, momclose, depression, livewithmom, gethitched.

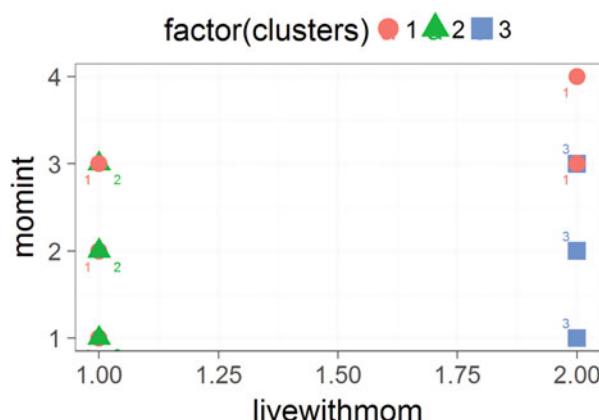
13.4.5 Step 5: Usage of Cluster Information

Clustering results could be utilized as new information augmenting the original dataset. For instance, we can add a *cluster* label in our `divorce` dataset:

```
divorce$clusters<-diz_clussters$cluster
divorce[1:5, ]
##   DIVYEAR momint dadint momclose depression Livewithmom gethitched
## 1       1      3      2      2       3           1       3
## 2       1      3      2      3       2           2       2
## 3       0      1      3      1       4           1       2
## 4       1      3      4      3       3           1       3
## 5       0      1      3      2       1           1       2
##   clusters
## 1       1
## 2       1
## 3       2
## 4       1
## 5       2
```

We can also examine the relationship between live with mom and feel close to mom by displaying a scatter plot of these two variables. If we suspect that young adults' personality might affect this relationship, then we could consider the potential personality (cluster type) in the plot. The cluster labels associated with each participant are printed in different positions relative to each pair of observations, (livewithmom, momint) (Fig. 13.7).

Fig. 13.7 Drill down for one feature (leave-with-mom) between the three cohorts



```

require(ggplot2)
ggplot(divorce, aes(livewithmom, momint), main="Scatterplot Live with mom vs
feel close to mom") +
  geom_point(aes(colour = factor(clusters), shape=factor(clusters), stroke =
8), alpha=1) +
  theme_bw(base_size=25) +
  geom_text(aes(label=ifelse(clusters%in%1, as.character(clusters), ''), hju
st=2, vjust=2, colour = factor(clusters)))+
  geom_text(aes(label=ifelse(clusters%in%2, as.character(clusters), ''), hju
st=-2, vjust=2, colour = factor(clusters))+
  geom_text(aes(label=ifelse(clusters%in%3, as.character(clusters), ''), hju
st=2, vjust=-1, colour = factor(clusters))) +
  guides(colour = guide_legend(override.aes = list(size=8))) +
  theme(legend.position="top")

```

We used `ggplot()` function in `ggplot2` package to label points with cluster types. `ggplot(divorce, aes(livewithmom, momint)) + geom_point()` gives us the scatterplot, and the three `geom_text()` functions help us label the points with the corresponding cluster identifiers.

This picture shows that live with mom does not necessarily mean young adults will feel close to mom. For “emotional” (Cluster 1) young adults, they felt close to their mom whether they live with their mom or not. “Naive” (Cluster 2) young adults feel closer to mom if they live with mom. However, they tend to be estranged from their mother. “Independent” (Cluster 3) young adults are opposite to Cluster 1. They felt closer to mom if they don’t live with her.

13.5 Model Improvement

Let’s still use the divorce data to illustrate a model improvement using **k-means++**. (Appropriate) initialization of the **k-means** algorithm is of paramount importance. The **k-means++** extension provides a practical strategy to obtain an optimal initialization for k-means clustering using a predefined `kpp_init` method.

```

# install.packages("matrixStats")
require(matrixStats)

kpp_init = function(dat, K) {
  x = as.matrix(dat)
  n = nrow(x)
  # Randomly choose a first center
  centers = matrix(NA, nrow=K, ncol=ncol(x))
  set.seed(123)
  centers[1,] = as.matrix(x[sample(1:n, 1),])
  for (k in 2:K) {
    # Calculate dist^2 to closest center for each point
    dists = matrix(NA, nrow=n, ncol=k-1)
    for (j in 1:(k-1)) {
      temp = sweep(x, 2, centers[j,], '-')
      dists[,j] = rowSums(temp^2)
    }
    dists = rowMins(dists)
  }
}

```

```

# Draw next center with probability proportional to dist^2
cumdists = cumsum(dists)
prop = runif(1, min=0, max=cumdists[n])
centers[k,] = as.matrix(x[min(which(cumdists > prop)),])
}
return(centers)
}

clust_kpp = kmeans(di_z, kpp_init(di_z, 3), iter.max=100, algorithm='Lloyd')

```

We can observe some differences.

```

clust_kpp$centers
##      DIVYEAR      momint      dadint      momclose depression Livewithmom
## 1  0.3741445  1.2578161 -0.6636602  0.5610651 -0.1505730 -0.4124815
## 2 -0.2659149 -0.5798266  0.3805174 -0.2538624  0.1639572 -0.5560862
## 3  0.3508225  0.5269697 -0.4329499  0.2251408 -0.2594488  1.3770536
## gethitched
## 1  0.9990071
## 2 -0.1489684
## 3 -0.2285310

```

This improvement is not substantial; the new overall average Silhouette value remains 0.2 for **k-means++**. Third compares to the value of 0.2 reported for the earlier k-means clustering, albeit the three groups generated by each method are quite distinct. In addition, the number of “mis-clustered” instances remains 2 although their Silhouette values are rather smaller than before, and the overall Cluster 1 Silhouette average value is low (0.006) (Fig. 13.8).

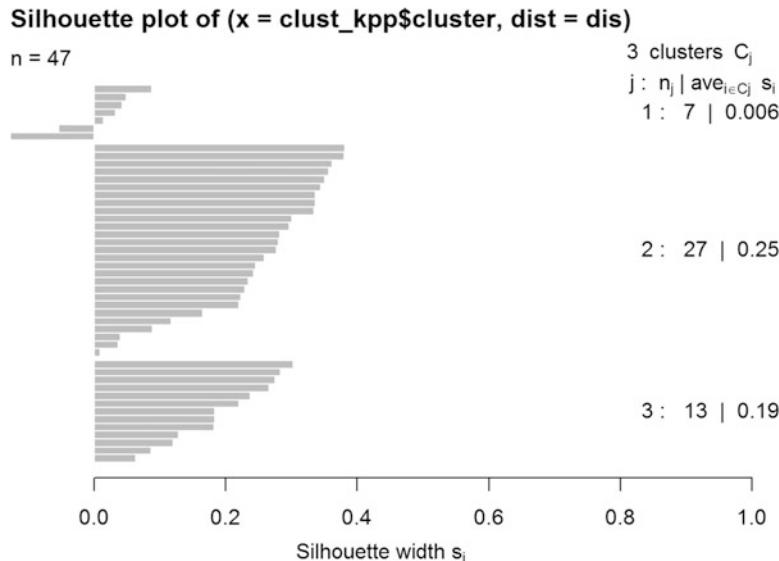


Fig. 13.8 Silhouette plot for k-means++ classification

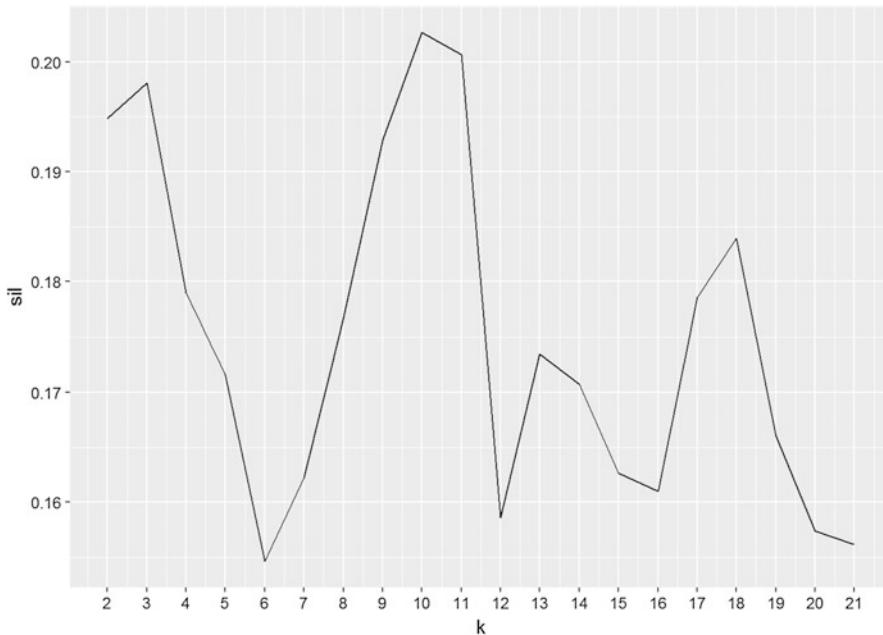


Fig. 13.9 Evolution of the average silhouette value with respect to the number of clusters

```

sil2 = silhouette(clust_kpp$cluster, dis)
summary(sil2)

## Silhouette of 47 units in 3 clusters from silhouette.default(x = clust_kp
p$cluster, dist = dis) :
## Cluster sizes and average silhouette widths:
##      7      27      13
## 0.00644352 0.24933847 0.19476785
## Individual silhouette widths:
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -0.12750  0.08781  0.22950  0.19810  0.29050  0.38120
plot(sil2)

```

13.5.1 Tuning the Parameter k

Similar to what we performed for KNN and SVM, we can tune the **k-means** parameters, including centers initialization and k (Fig. 13.9).

```

n_rows <- 21
mat = matrix(0, nrow = n_rows)
for (i in 2:n_rows){
  set.seed(321)
  clust_kpp = kmeans(di_z, kpp_init(di_z, i), iter.max=100, algorithm='Lloyd')
  sil = silhouette(clust_kpp$cluster, dis)
  mat[i] = mean(as.matrix(sil)[,3])
}
colnames(mat) <- c("Avg_Silhouette_Value")
mat

##          Avg_Silhouette_Value
## [1,]      0.0000000
## [2,]      0.1948335
## [3,]      0.1980686
## [4,]      0.1789654
## [5,]      0.1716270
## [6,]      0.1546357
## [7,]      0.1622488
## [8,]      0.1767659
## [9,]      0.1928883
## [10,]     0.2026559
## [11,]     0.2006313
## [12,]     0.1586044
## [13,]     0.1735035
## [14,]     0.1707446
## [15,]     0.1626367
## [16,]     0.1609723
## [17,]     0.1785733
## [18,]     0.1839546
## [19,]     0.1660019
## [20,]     0.1573574
## [21,]     0.1561791

ggplot(data.frame(k=2:n_rows, sil=mat[2:n_rows]), aes(x=k, y=sil))+
  geom_line()+
  scale_x_continuous(breaks = 2:n_rows)

```

This suggests that $k \sim 3$ may be an appropriate number of clusters to use in this case.

Next, let's set the maximal iteration of the algorithm and rerun the model with optimal $k = 2$, $k = 3$ or $k = 10$. Below, we just demonstrate the results for $k = 3$. There are still 2 mis-clustered observations, which is not a significant improvement on the prior model according to the average Silhouette measure (Fig. 13.10).

```

k <- 3
set.seed(31)
clust_kpp = kmeans(di_z, kpp_init(di_z, k), iter.max=200, algorithm="MacQueen")
sil3 = silhouette(clust_kpp$cluster, dis)
summary(sil3)

## Silhouette of 47 units in 3 clusters from silhouette.default(x = clust_kpp$cluster, dist = dis) :
##   Cluster sizes and average silhouette widths:
##      10      22      15
## 0.02096194 0.30414984 0.15474729
##   Individual silhouette widths:
##      Min. 1st Qu. Median 3rd Qu. Max.
## -0.1365  0.1032  0.1971  0.1962  0.3122  0.4113

plot(sil3)

```

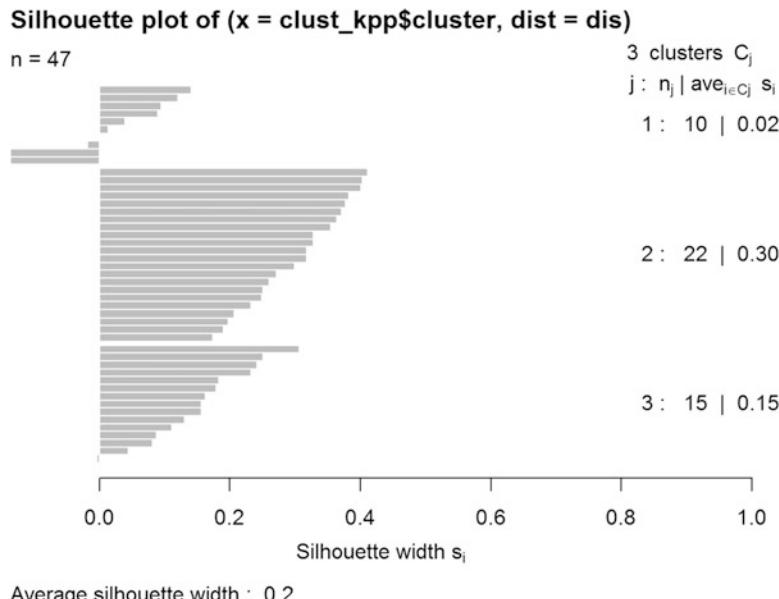


Fig. 13.10 Silhouette plot for the optimal $k = 3$ and [kpp_init](#) Initialization

Note that we now see 3 cases of group 1 that have negative silhouette values (previously we had only 2), albeit the overall average silhouette remains 0.2.

13.6 Case Study 2: Pediatric Trauma

Let's go through another example demonstrating the *k-means* clustering method using a larger dataset.

13.6.1 Step 1: Collecting Data

The dataset we will interrogate now includes Services Utilization by Trauma-Exposed Children in the US data, which is located in our case-studies folder. This case study examines associations between post-traumatic psychopathology and service utilization by trauma-exposed children.

Variables:

- **id:** Case identification number.
- **sex:** Female or male, dichotomous variable (1 = female, 0 = male).
- **age:** Age of child at time of seeking treatment services. Interval-level variable, score range = 0–18.
- **race:** Race of child seeking treatment services. Polytomous variable with 4 categories (1 = black, 2 = white, 3 = hispanic, 4 = other).

- **cmt**: The child was exposed to child maltreatment trauma - dichotomous variable (1 = yes, 0 = no).
- **traumatotype**: Type of trauma exposure the child is seeking treatment for. Polytomous variable with 5 categories ("sexabuse" = sexual abuse, "physabuse" = physical abuse, "neglect" = neglect, "psychabuse" = psychological or emotional abuse, "dvexp" = exposure to domestic violence or intimate partner violence).
- **ptsd**: The child has current post-traumatic stress disorder. Dichotomous variable (1 = yes, 0 = no).
- **dissoc**: The child currently has a dissociative disorder (PTSD dissociative subtype, DESNOS, DSNOS). Interval-level variable, score range = 0–11.
- **service**: Number of services the child has utilized in the past 6 months, including primary care, emergency room, outpatient therapy, outpatient psychiatrist, inpatient admission, case management, in-home counseling, group home, foster care, treatment foster care, therapeutic recreation or mentor, department of social services, residential treatment center, school counselor, special classes or school, detention center or jail, probation officer. Interval-level variable, score range = 0–19.
- **Note**: These data (Case_04_ChildTrauma_.Data.csv) are tab-delimited.

13.6.2 Step 2: Exploring and Preparing the Data

First, we need to load the dataset into R and report its summary and dimensions.

```
trauma<-read.csv("https://umich.instructure.com/files/399129/download?downlo
ad_frd=1", sep = " ")
summary(trauma); dim(trauma)

##      id          sex         age         ses
##  Min. : 1.0  Min. :0.000  Min. : 2.000  Min. : 0.00
##  1st Qu.:250.8 1st Qu.:0.000  1st Qu.: 7.000  1st Qu.: 0.00
##  Median :500.5 Median :1.000  Median : 9.000  Median : 0.00
##  Mean   :500.5 Mean  :0.506  Mean   : 8.982  Mean   : 0.18
##  3rd Qu.:750.2 3rd Qu.:1.000  3rd Qu.:11.000  3rd Qu.: 0.00
##  Max.   :1000.0 Max.  :1.000  Max.   :25.000  Max.   : 1.00
##      race        traumatotype      ptsd        dissoc
##  black   :200  dvexp   :250  Min.   :0.00  Min.   : 0.000
##  hispanic:100 neglect  :350  1st Qu.:0.00  1st Qu.: 0.000
##  other    :100 physabuse:100  Median :0.00  Median : 1.000
##  white    :600 psychabuse:200  Mean   :0.29  Mean   : 0.598
##                  sexabuse :100  3rd Qu.:1.00  3rd Qu.: 1.000
##                               Max.   :1.00  Max.   : 1.000
##      service
##  Min.   : 0.000
##  1st Qu.: 8.000
##  Median :10.000
##  Mean   : 9.926
##  3rd Qu.:12.000
##  Max.   :20.000
## [1] 1000   9
```

In the summary we see two factors `race` and `traumatype`. `traumatype` codes the real classes we are interested in. If the clusters created by the model are quite similar to the trauma types, our model may have a quite reasonable interpretation. Let's also create a dummy variable for each racial category.

```
trauma$black<-ifelse(trauma$race=="black", 1, 0)
trauma$hispanic<-ifelse(trauma$race=="hispanic", 1, 0)
trauma$other<-ifelse(trauma$race=="other", 1, 0)
trauma$white<-ifelse(trauma$race=="white", 1, 0)
```

Then, we will remove `traumatype` the class variable from the dataset to avoid biasing the clustering algorithm. Thus, we are simulating a real biomedical case-study where we do not necessarily have the actual class information available, i.e., classes are latent features.

```
trauma_notype<-trauma[, -c(1, 5, 6)]
```

13.6.3 Step 3: Training a Model on the Data

Similar to case-study 1, let's standardize the dataset and fit a k-means model.

```
tr_z<- as.data.frame(Lapply(trauma_notype, scale))
str(tr_z)

## 'data.frame': 1000 obs. of 10 variables:
## $ sex      : num  0.988 0.988 -1.012 -1.012 0.988 ...
## $ age      : num  -0.997 1.677 -0.997 0.674 -0.662 ...
## $ ses      : num  -0.468 -0.468 -0.468 -0.468 -0.468 ...
## $ ptsd     : num  1.564 -0.639 -0.639 -0.639 1.564 ...
## $ dissoci : num  0.819 -1.219 0.819 0.819 0.819 ...
## $ service  : num  2.314 0.678 -0.303 0.351 1.66 ...
## $ black    : num  2 2 2 2 ...
## $ hispanic: num  -0.333 -0.333 -0.333 -0.333 -0.333 ...
## $ other    : num  -0.333 -0.333 -0.333 -0.333 -0.333 ...
## $ white    : num  -1.22 -1.22 -1.22 -1.22 -1.22 ...
```

set.seed(1234)

```
trauma_clusters<-kmeans(tr_z, 6)
```

Here we use $k = 6$ in the hope that we may have 5 of these clusters match the specific 5 trauma types. In this case study, we have 1000 observations and $k = 6$ may be a reasonable option.

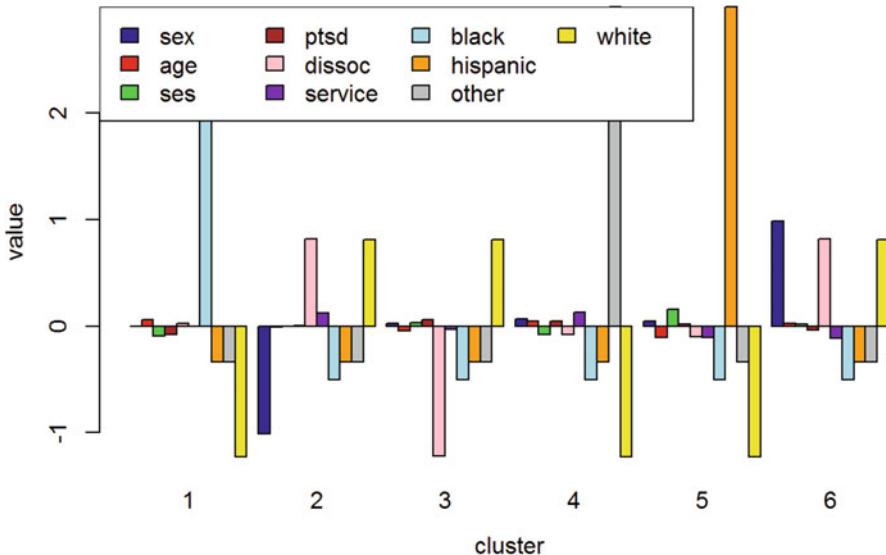


Fig. 13.11 Key predictors discriminating between the 6 cohorts in the trauma study

13.6.4 Step 4: Evaluating Model Performance

To assess the clustering model results, we can examine the resulting clusters (Fig. 13.11).

```
trauma_clusters$centers
##          sex      age      ses      ptsd      dissociation
## 1 -0.001999144  0.061154336 -0.091055799 -0.077094361  0.02446247
## 2 -1.011566709 -0.006361734 -0.000275301  0.002214351  0.81949287
## 3  0.026286613 -0.043755817  0.029890657  0.064206246 -1.21904661
## 4  0.067970886  0.046116384 -0.078047828  0.044053921 -0.07746450
## 5  0.047979449 -0.104263129  0.156095655  0.022026960 -0.09784989
## 6  0.987576985  0.028799955  0.019511957 -0.038046568  0.81949287
##          service     black   hispanic     other     white
## 1  0.001308569  1.9989997 -0.3331666 -0.3331666 -1.2241323
## 2  0.126332303 -0.4997499 -0.3331666 -0.3331666  0.8160882
## 3 -0.030083167 -0.4997499 -0.3331666 -0.3331666  0.8160882
## 4  0.128894052 -0.4997499 -0.3331666  2.9984996 -1.2241323
## 5 -0.103376956 -0.4997499  2.9984996 -0.3331666 -1.2241323
## 6 -0.111481162 -0.4997499 -0.3331666 -0.3331666  0.8160882
myColors <- c("darkblue", "red", "green", "brown", "pink", "purple", "lightblue", "orange", "grey", "yellow")
barPlot(t(trauma_clusters$centers), beside = TRUE, xlab="cluster",
yLab="value", col = myColors)
Legend("topleft", ncol=4, Legend = c("sex", "age", "ses", "ptsd", "dissoc",
"service", "black", "hispanic", "other", "white"), fill = myColors)
```

On this barplot, the bars in each cluster represents `sex`, `age`, `ses`, `ptsd`, `dissoc`, `service`, `black`, `hispanic`, `other`, and `white`, respectively. It is quite obvious that each cluster has some unique features.

Next, we can compare the *k-means* computed cluster labels to the *original labels*. Let's evaluate the similarities between the automated cluster labels and their real class counterparts using a confusion matrix table, where rows represent the k-means clusters, columns show the actual labels, and the cell values include the frequencies of the corresponding pairings.

```
trauma$clusters<-trauma_clusters$cluster
table(trauma$clusters, trauma$traumatype)

##
##      dvexp neglect physabuse psychabuse sexabuse
## 1      0        0      100        0      100
## 2     10       118        0      61        0
## 3     23       133        0      79        0
## 4    100        0        0      0        0
## 5    100        0        0      0        0
## 6     17       99        0      60        0
```

We can see that all of the children in Cluster 4 belong to `dvexp` (exposure to domestic violence or intimate partner violence). If we use the mode of each cluster to be the class for that group of children, we can classify 63 `sexabuse` cases, 279 `neglect` cases, 41 `physabuse` cases, 100 `dvexp` cases, and another 71 `neglect` cases. That is 554 cases out of 1,000 cases identified with correct class. The model has a problem in distinguishing between `neglect` and `psychabuse`, but it has a good accuracy.

Let's review the output Silhouette value summary. It works well as only a small portion of samples appear mis-clustered.

```
dis_tra = dist(tr_z)
sil_tra = silhouette(trauma_clusters$cluster, dis_tra)
summary(sil_tra)

## Silhouette of 1000 units in 6 clusters from silhouette.default(x = trauma
## _clusters$cluster, dist = dis_tra) :
## Cluster sizes and average silhouette widths:
##      200     189     235     100     100     176
## 0.2595725 0.2185706 0.1039559 0.3223076 0.3199830 0.2423110
## Individual silhouette widths:
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.008893 0.139100 0.234400 0.224500 0.303300 0.388200

#plot(sil_tra)
# report the overall mean silhouette value
mean(sil_tra[, "sil_width"])

## [1] 0.2245298

# The sil object colnames are ("cluster", "neighbor", "sil_width")
```

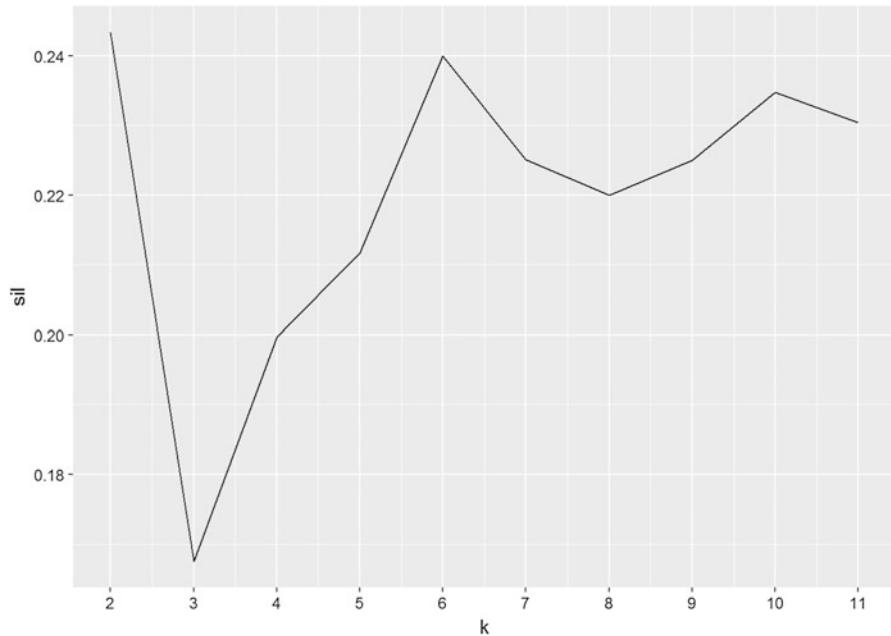


Fig. 13.12 Evolution of the average silhouette value with respect to the number of clusters

Next, let's try to tune k with **k-means++** and see if $k = 6$ appears to be optimal (Fig. 13.12).

```
mat = matrix(0, nrow = 11)
for (i in 2:11){
  set.seed(321)
  clust_kpp = kmeans(tr_z, kpp_init(tr_z, i), iter.max=100, algorithm='Lloyd')
  sil = silhouette(clust_kpp$cluster, dis_tra)
  mat[i] = mean(as.matrix(sil)[,3])
}
mat
## [1,] 0.0000000
## [2,] 0.2433222
## [3,] 0.1675486
## [4,] 0.1997315
## [5,] 0.2116534
## [6,] 0.2400086
## [7,] 0.2251367
## [8,] 0.2199859
## [9,] 0.2249569
## [10,] 0.2347122
## [11,] 0.2304451
ggplot(data.frame(k=2:11, sil=mat[2:11])),aes(x=k,y=sil))+geom_line() + scale_x_continuous(breaks = 2:11)
```

Finally, let's use **k-means++** with $k = 6$ and set the algorithm's maximal iteration before rerunning the experiment:

```
set.seed(1234)
clust_kpp = kmeans(tr_z, kpp_init(tr_z, 6), iter.max=100, algorithm='Lloyd')
sil = silhouette(clust_kpp$cluster, dis_tra)
summary(sil)

## Silhouette of 1000 units in 6 clusters from silhouette.default(x = clust_kpp$cluster, dist = dis_tra) :
## Cluster sizes and average silhouette widths:
##      422      100      178      85      15      200
## 0.2166778 0.3353976 0.1898492 0.2478090 0.2294502 0.2836607
## Individual silhouette widths:
##   Min. 1st Qu.  Median  Mean 3rd Qu.  Max.
## 0.03672 0.19730 0.23080 0.24000 0.27710 0.40650

# plot(sil)
# report the overall mean silhouette value
mean(sil[, "sil_width"])

## [1] 0.2400086
```

13.6.5 Practice Problem: Youth Development

Use the Boys Town Study of Youth Development data, second case study, CaseStudy02_Boystown_Data.csv, which we used in Chap. 7, to find clusters using variables like GPA, alcohol abuse, attitudes on drinking, social status, parent closeness, and delinquency for clustering (all variables other than gender and ID).

First, we must load the data and transfer sex, dadjob, and momjob into dummy variables.

```
boystown<-read.csv("https://umich.instructure.com/files/399119/download?down
load_frd=1", sep=" ")
boystown$sex<-boystown$sex-1
boystown$dadjob <- (-1)*(boystown$dadjob-2)
boystown$momjob <- (-1)*(boystown$momjob-2)
str(boystown)

## 'data.frame': 200 obs. of 11 variables:
## $ id      : int 1 2 3 4 5 6 7 8 9 10 ...
## $ sex     : num 0 0 0 0 1 1 0 0 1 1 ...
## $ gpa     : int 5 0 3 2 3 3 1 5 1 3 ...
## $ Alcoholuse: int 2 4 2 2 6 3 2 6 5 2 ...
## $ alcatt  : int 3 2 3 1 2 0 0 3 0 1 ...
## $ dadjob  : num 1 1 1 1 1 1 1 1 1 1 ...
## $ momjob  : num 0 0 0 0 1 0 0 0 1 1 ...
## $ dadclose : int 1 3 2 1 2 1 3 6 3 1 ...
## $ momclose : int 1 4 2 2 1 2 1 2 3 2 ...
## $ Larceny : int 1 0 0 3 1 0 0 0 1 1 ...
## $ vandalism: int 3 0 2 2 2 0 5 1 4 0 ...
```

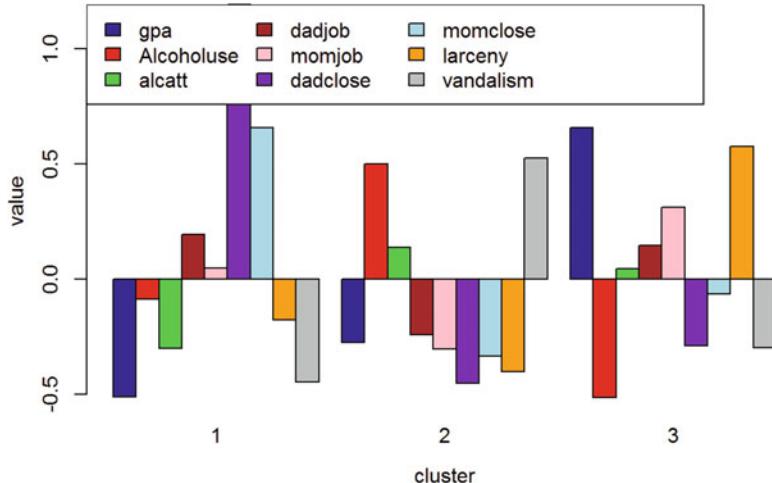


Fig. 13.13 Main features discriminating between the 3 cohorts in the divorce impact on youth study

Then, extract all the variables, except the first two columns (subject identifiers and genders).

```
boystown_sub<-boystown[, -c(1, 2)]
```

Next, we need to standardize and clustering the data with $k = 3$. You may have the following centers (numbers could be a little different) (Fig. 13.13).

```
##      gpa Alcoholuse alcatt dadjob momjob dadclose
## 1 -0.5101243 -0.08555163 -0.30098866 0.1939577 0.04868109 1.1914502
## 2 -0.2753631  0.49998217  0.13804858 -0.2421906 -0.30151766 -0.4521484
## 3  0.6590193 -0.51256447  0.04599325  0.1451756  0.31107377 -0.2896562
##      momclose larceny vandalism
## 1  0.65647213 -0.1755012 -0.4453044
## 2 -0.33341358 -0.4017282  0.5252308
## 3 -0.06343891  0.5769583 -0.2981561
```

Add *k-means* cluster labels as a new (last) column back in the original dataset.

To investigate the gender distribution within different clusters we may use `aggregate()`.

```
# Compute the averages for the variable 'sex', grouped by cluster
aggregate(data=boystown, sex~clusters, mean)

##   clusters      sex
## 1          1 0.6875000
## 2          2 0.5802469
## 3          3 0.6760563
```

Here `clusters` is the new vector indicating cluster labels. The gender distribution does not vary much between different cluster labels (Fig. 13.14).

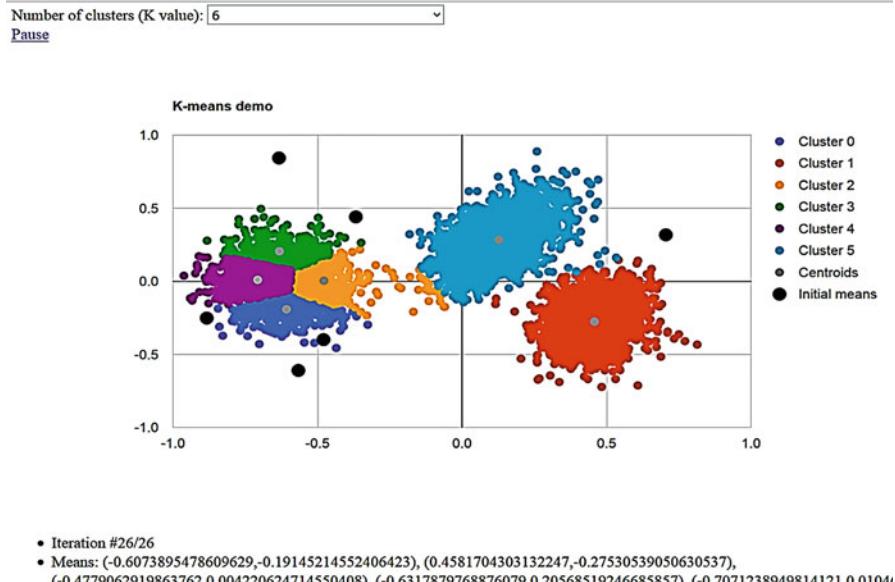


Fig. 13.14 Live demo: k-means point clustering

This k-means live demo shows point clustering (Applies Multiclass AdaBoost.M1, SAMME and Bagging algorithm) <http://olalonde.github.com/kmeans.js>.

13.7 Hierarchical Clustering

There are a number of R **hierarchical clustering** packages, including:

- `hclust` in base R.
- `agnes` in the `cluster` package.

Alternative distance measures (or linkages) can be used in all Hierarchical Clustering, e.g., *single*, *complete* and *ward*.

We will demonstrate hierarchical clustering using case-study 1 (Divorce and Consequences on Young Adults). Pre-set $k = 3$ and notice that we have to use normalized data for hierarchical clustering.

```
require(cluster)
pitch_sing = agnes(di_z, diss=FALSE, method='single')
pitch_comp = agnes(di_z, diss=FALSE, method='complete')
pitch_ward = agnes(di_z, diss=FALSE, method='ward')
sil_sing = silhouette(cutree(pitch_sing, k=3), dis)
sil_comp = silhouette(cutree(pitch_comp, k=3), dis)
# try 10 clusters, see plot above
sil_ward = silhouette(cutree(pitch_ward, k=10), dis)
```

You can generate the hierarchical plot by `ggdendrogram` in the package `ggdendro` (Figs. 13.15 and 13.16).

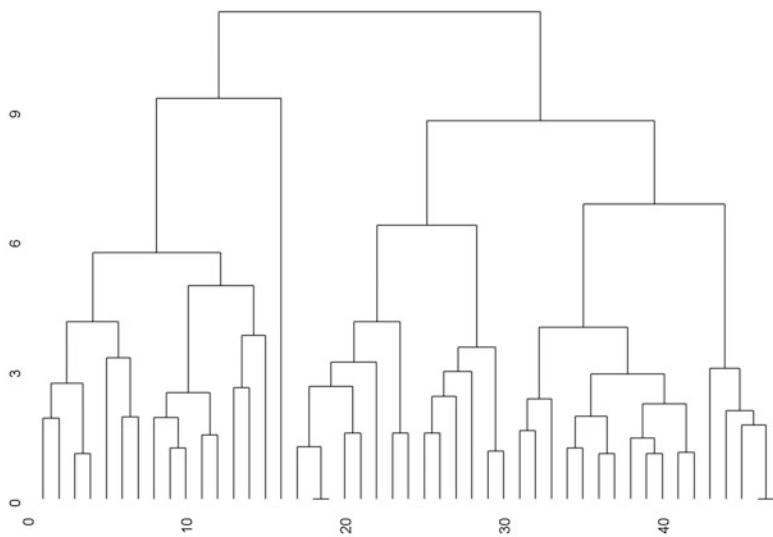


Fig. 13.15 Hierarchical clustering using the Ward method

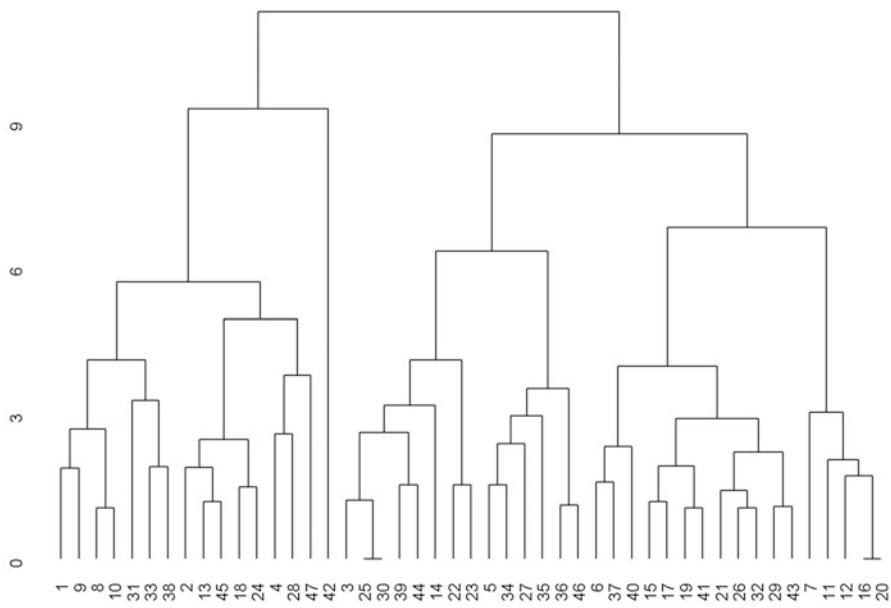


Fig. 13.16 Ten-level hierarchical clustering using the Ward method

```
# install.packages("ggdendro")
require(ggdendro)

ggdendrogram(as.dendrogram(pitch_ward), Leaf_Labels=FALSE, Labels=FALSE)
```

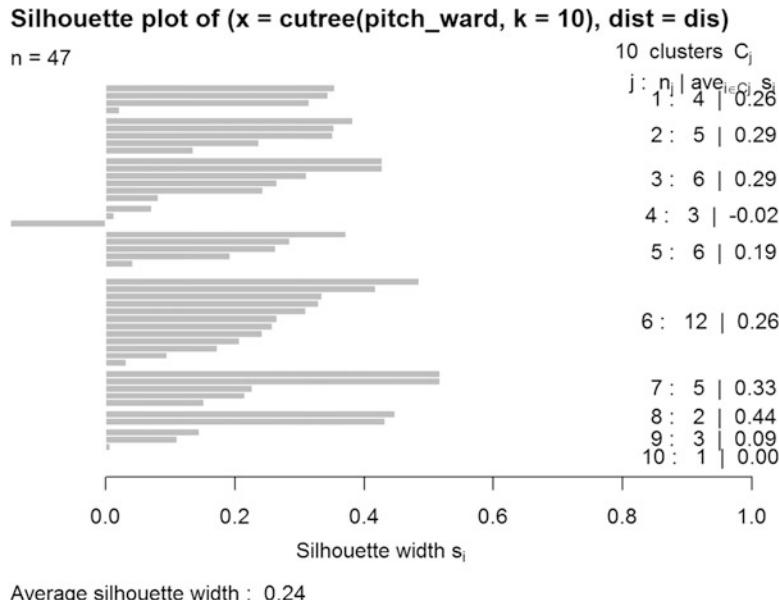


Fig. 13.17 Silhouette plot for hierarchical clustering using the Ward method

```
mean(sil_ward[, "sil_width"])
## [1] 0.2398738
ggdendrogram(as.dendrogram(pitch_ward), Leaf_Labels=TRUE, labels=T, size=10)
```

Generally speaking, the best result should come from **wald** linkage, but you should also try complete linkage (method = ‘complete’). We can see that the hierarchical clustering result (average silhouette value ~ 0.24) mostly agrees with the prior *k-means* (0.2) and *k-means++* (0.2) results (Fig. 13.17).

```
summary(sil_ward)
## Silhouette of 47 units in 10 clusters from silhouette.default(x = cutree(
pitch_ward, k = 10), dist = dis) :
## Cluster sizes and average silhouette widths:
##      4      5      6      3      6     12
## 0.25905454 0.29195989 0.29305926 -0.02079056 0.19263836 0.26268274
##      5      2      3      1
## 0.32594365 0.44074717 0.08760990 0.00000000
## Individual silhouette widths:
##   Min. 1st Qu. Median  Mean 3rd Qu.  Max.
## -0.1477  0.1231  0.2577  0.2399  0.3524  0.5176
plot(sil_ward)
```

13.8 Gaussian Mixture Models

More details about Gaussian mixture models (GMM) are provided in the supporting materials online. Below is a brief introduction to GMM using the `Mclust` function in the *R* package `mclust`.

For multivariate mixture, there are totally 14 possible models:

- "EII" = spherical, equal volume
- "VII" = spherical, unequal volume
- "EEI" = diagonal, equal volume and shape
- "VEI" = diagonal, varying volume, equal shape
- "EVI" = diagonal, equal volume, varying shape
- "VVI" = diagonal, varying volume and shape
- "EEE" = ellipsoidal, equal volume, shape, and orientation
- "EVE" = ellipsoidal, equal volume and orientation (*)
- "VEE" = ellipsoidal, equal shape and orientation (*)
- "VVE" = ellipsoidal, equal orientation (*)
- "EEV" = ellipsoidal, equal volume and equal shape
- "VEV" = ellipsoidal, equal shape
- "EVV" = ellipsoidal, equal volume (*)
- "VVV" = ellipsoidal, varying volume, shape, and orientation

For more practical details, you may refer to `Mclust`. For more theoretical details, see C. Fraley and A. E. Raftery (2002).

Let's use the *Divorce and Consequences on Young Adults* dataset for a demonstration.

```
library(mclust)
set.seed(1234)
gmm_clust = Mclust(di_z)
gmm_clust$modelName
## [1] "EEE"
```

Thus, the optimal model here is "EEE" (Figs. 13.18, 13.19, and 13.20).

```
plot(gmm_clust$BIC, LegendArgs = List(x = "bottom", ncol = 2, cex = 1))
plot(gmm_clust, what = "density")
plot(gmm_clust, what = "classification")
```

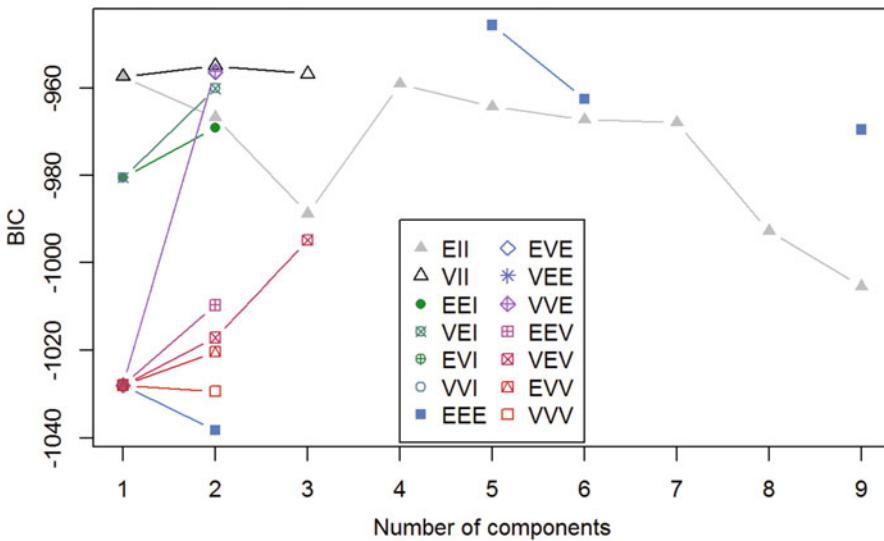


Fig. 13.18 Bayesian information criterion plots for different GMM classification models for the divorce youth data

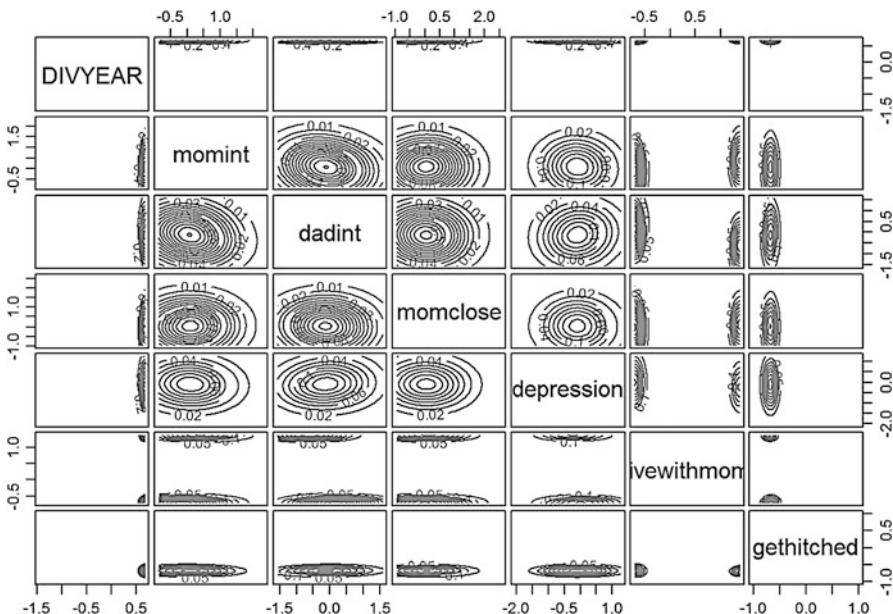


Fig. 13.19 Pairs plot of the GMM clustering density

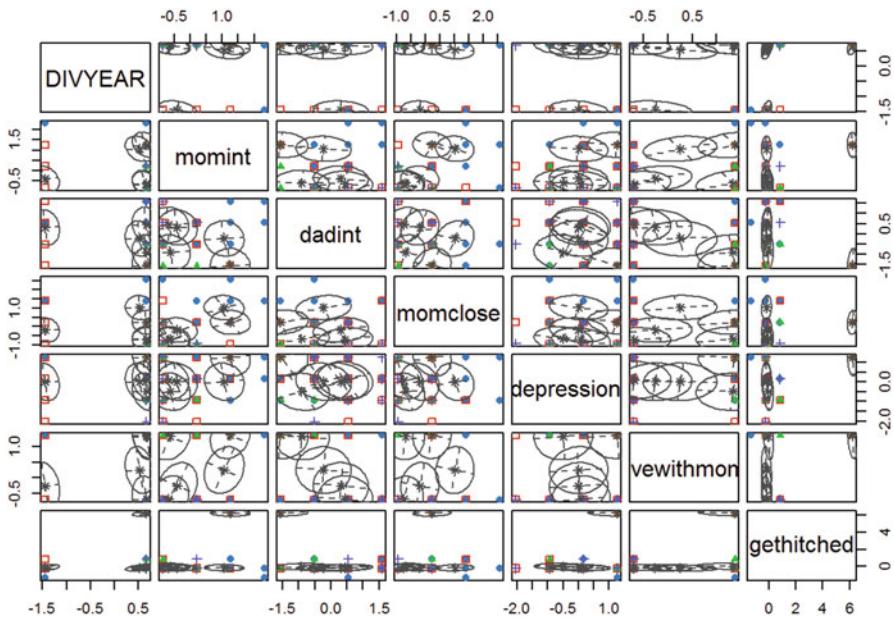


Fig. 13.20 Pairs plot of the GMM classification results

13.9 Summary

- k-means clustering may be most appropriate for exploratory data analytics. It is highly flexible and fairly efficient in terms of tessellating data into groups.
- It can be used for data that has no *Apriori* classes (labels).
- Generated clusters may lead to phenotype stratification and/or be compared against known clinical traits.

Try to use these techniques with other data from the list of our Case-Studies.

13.10 Assignments: 13. k-Means Clustering

Use the Amyotrophic Lateral Sclerosis (ALS) dataset. This case-study examines the patterns, symmetries, associations and causality in a rare but devastating disease, amyotrophic lateral sclerosis (ALS). A major clinically relevant question in this biomedical study is: *What patient phenotypes can be automatically and reliably identified and used to predict the change of the ALSFRS slope over time?*. This problem aims to explore the data set by unsupervised learning.

- Load and prepare the data.
- Perform summary and preliminary visualization.

- Train a **k-means** model on the data, select k
- as we mentioned in Chap. 13.
- Evaluate the model performance and report the center of clusters and silhouette plots. Explain details (Note: Since we have 100 dimensions, it may be difficult to use bar plots, so show the centers only).
- Tune parameters and plot with **k-means++**.
- Rerun the model with optimal parameters and interpret the clustering results.
- Apply *Hierarchical Clustering* on three different linkages and compare the corresponding **Silhouette** plots.
- Fit a Gaussian mixture model, select the optimal model and draw **BIC** and **Silhouette** plots. (Hint, you need to sample part of data or it could be very time consuming).
- Compare the result of the above methods.

References

- Wu, J. (2012) *Advances in K-means Clustering: A Data Mining Thinking*, Springer Science & Business Media, ISBN 3642298079, 9783642298073.
- Dinov, ID. (2008) Expectation Maximization and Mixture Modeling Tutorial. Statistics Online Computational Resource. UCLA: Statistics Online Computational Resource. Retrieved from: <http://escholarship.org/uc/item/1rb70972>.
- Celebi, ME (ed.) (2014) *Partitional Clustering Algorithms*, SpringerLink: Bücher, ISBN 3319092596, 9783319092591.
- Fraley, C and Raftery, AE. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association*, 97, 611–631.

Chapter 14

Model Performance Assessment



In previous chapters, we used prediction accuracy to evaluate classification models. However, having accurate predictions in one dataset does not necessarily imply that the model is perfect or that it will reproduce when tested on external data. We need additional metrics to evaluate the model performance and to make sure it is robust, reproducible, reliable, and unbiased.

In this chapter, we will discuss (1) various evaluation strategies for prediction, clustering, classification, regression, and decision trees; (2) visualization of ROC curves and performance tradeoffs; and (3) estimation of future performance, internal statistical cross-validation and bootstrap sampling.

14.1 Measuring the Performance of Classification Methods

As mentioned previously, classification model performances could not be evaluated by prediction accuracy alone. We make different classification models for different purposes. For example, in newborns screening for genetic defects we want the model to have as few true negatives as possible. We don't want to classify anyone as "no defect" when they actually have a defect gene, since early treatment might alter the destiny of this newborn.

We can use the following three types of data to evaluate the performance of a classifier model.

- Actual class values (for supervised classification).
- Predicted class values.
- Estimated probability of the prediction.

We are familiar with the first two cases. The last type of validation relies on the `predict(model, test_data)` function that we have talked about in previous classification and prediction chapters (Chaps. 7, 8, and 9). Let's revisit the model and test data we discussed in Chap. 8; the Inpatient Head and Neck Cancer Medication data.

We will demonstrate prediction probability estimation using this case-study CaseStudy14_HeadNeck_Cancer_Medication.csv

```
pred_raw<-predict(hn_classifier, hn_test, type="raw")
head(pred_raw)

##      early_stage later_stage
## [1,] 0.9381891 0.06181090
## [2,] 0.9381891 0.06181090
## [3,] 0.8715581 0.12844188
## [4,] 0.9382140 0.06178601
## [5,] 0.9675997 0.03240026
## [6,] 0.9675997 0.03240026
```

The above output includes the prediction probabilities for the first 6 rows of the data. This example is based on the Naive Bayes classifier, however the same approach works for any other machine-learning classification or prediction technique.

In addition, we can report the predicted probability with the outputs of the Naive Bayesian decision-support system (hn_classifier <- naiveBayes (hn_train, hn_med_train\$stage)):

```
(hn_classifier <- naiveBayes(hn_train, hn_med_train$stage)):

pred_nb<-predict(hn_classifier, hn_test)
head(pred_nb)

## [1] early_stage early_stage early_stage early_stage early_stage
early_stage
## Levels: early_stage later_stage
```

The general predict () method automatically subclasses to the specific predict.naiveBayes (object, newdata, type = c ("class", "raw"), threshold = 0.001, ...) call where type = "raw" and type = "class" specify the output as the conditional a-posterior probabilities for each class or the class with maximal probability, respectively. Back in Chap. 9, we discussed the C5.0 and the randomForest classifiers used to predict the chronic disease score in a (different) Quality of Life Study.

Below are the (probability) results of the C5.0 classification prediction:

```
pred_prob<-predict(qol_model, qol_test, type="prob")
head(pred_prob)

##      minor_disease severe_disease
## 10      0.1979698 0.8020302
## 12      0.1979698 0.8020302
## 26      0.3468705 0.6531295
## 37      0.1263975 0.8736025
## 41      0.7290209 0.2709791
## 43      0.3163673 0.6836327
```

These can be contrasted against the C5.0 classification label results:

```
pred_tree<-predict(qol_model, qol_test)
head(pred_tree)
## [1] severe_disease severe_disease severe_disease severe_disease
## [5] minor_disease severe_disease
## Levels: minor_disease severe_disease
```

The same complementary types of outputs can be reported for most machine-learning classification and prediction approaches

14.2 Evaluation Strategies

In Chap. 7, we saw an attempt to categorize the supervised classification and unsupervised clustering methods. Similarly, Table 14.1 summarizes the basic types of evaluation and validation strategies for different forecasting, prediction, ensembling, and clustering techniques. (Internal) Statistical Cross Validation or external validation should always be applied to ensure reliability and reproducibility of the results. The SciKit clustering performance evaluation and Classification metrics page provide details about many alternative techniques and metrics for performance evaluation of clustering and classification methods.

14.2.1 *Binary Outcomes*

More details about binary test assessment are available on the Scientific Methods for Health Sciences (SMHS) EBook site. Table 14.2 summarizes the key measures

Table 14.1 Categories of clustering validation and classification evaluation strategies

Inference	Outcome	Evaluation metrics	Example R functions
Classification & Prediction	Binary	Accuracy, Sensitivity, Specificity, PPV/Precision, NPV/Recall, LOR	<code>caret::confusionMatrix, gmodels::CrossTable, cluster::silhouette</code>
Classification & Prediction	Categorical	Accuracy, Sensitivity/Specificity, PPV, NPV, LOR, Silhouette Coefficient	<code>caret::confusionMatrix, gmodels::CrossTable, cluster::silhouette</code>
Regression Modeling	Real Quantitative	correlation coefficient, R^2 , RMSE, Mutual Information, Homogeneity and Completeness Scores	<code>cor, metrics::mse</code>

Table 14.2 Evaluation of binary (dichotomous) statistical tests, classification methods, or forecasting predictions

		Actual condition(or real class label)		Test interpretation
		Absent (H_0 is true)	Present (H_1 is true)	
Test Result (Prediction or Classifica- tion Label)	Negative (fail to reject H_0)	TN Condition absent + Negative result = True (accurate) Negative	FN Condition pre- sent + Negative result = False (invalid) Negative Type II error (proportional to β)	$NPV = \frac{TN}{TN+FN}$
	Positive(reject H_0)	FP Condition absent + Positive result = False Positive Type I error (α)	TP Condition Present + Positive result = True Positive	$PPV = Precision$ $= \frac{TP}{TP+FP}$
Test Interpretation	$Power = 1 - \beta$ $=$ $1 - \frac{FN}{FN+TP}$	$Specificity = \frac{TN}{TN+FP}$	$Power =$ $Sensitivity =$ $\frac{TP}{TP+FN}$	$LOR = \ln \left(\frac{S1/F1}{S2/F2} \right)$ $= \ln \left(\frac{(S1 \times F2)}{(S2 \times F1)} \right)$, S = success, F = failure for 2 binary variables, 1 and 2

Table 14.3 Cross-table

	Predict_T	predict_F
TRUE	TP	TN
FALSE	FP	FN

commonly used to evaluate the performance of binary tests, classifiers, or predictions.

See also SMHS EBook; Power, Sensitivity and Specificity section.

14.2.2 Confusion Matrices

We talked about this confusion matrices in Chap. 9. For binary classes, these will be 2×2 matrices. Each of the cells has specific meaning, see the 2×2 Table 14.2 where

- **True Positive(TP):** Number of observations that correctly classified as “yes” or “success”
- **True Negative(TN):** Number of observations that correctly classified as “no” or “failure”
- **False Positive(FP):** Number of observations that incorrectly classified as “yes” or “success”
- **False Negative(FN):** Number of observations that incorrectly classified as “no” or “failure”

Using Confusion Matrices to Measure Performance

The way we calculate accuracy using these four cells is summarized by the following formula:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{\text{Total number of observations}} .$$

On the other hand, the error rate, or proportion of incorrectly classified observations, is calculated using:

$$\text{errorrate} = \frac{FP + FN}{TP + TN + FP + FN} = \frac{FP + FN}{\text{Total number of observations}} \\ = 1 - \text{accuracy}.$$

If we look at the numerator and denominator carefully, we can see that the error rate and accuracy add up to 1. Therefore, 95% accuracy implies a 5% error rate.

In R, we have multiple ways to obtain confusion matrices. The simplest way would be to use `table()`. For example, in Chap. 8, to report a plain 2×2 table we used:

```
hn_test_pred<-predict(hn_classifier, hn_test)
table(hn_test_pred, hn_med_test$stage)

##
## hn_test_pred  early_stage  later_stage
##   early_stage      69        23
##   later_stage       8         0
```

Then why did we use `CrossTable()` function back in Chapter 8? Because it reports additional useful information about the model performance.

```
library(gmodels)
CrossTable(hn_test_pred, hn_med_test$stage)

##   Cell Contents
##   |-----|
##   |           N |
##   | Chi-square contribution |
##   |           N / Row Total |
##   |           N / Col Total |
##   |           N / Table Total |
##   |-----|
##   Total Observations in Table:  100
##   | hn_med_test$stage
##   hn_test_pred | early_stage | later_stage |   Row Total |
##   |-----|-----|-----|-----|
##   early_stage |        69 |        23 |        92 |
##   |        0.048 |        0.160 |           |
```

##	/	0.750	/	0.250	/	0.920	/
##	/	0.896	/	1.000	/		/
##	/	0.690	/	0.230	/		/
##	-----	-----	-----	-----	-----	-----	-----
##	Later_stage	/	8	/	0	/	8
##			0.550	/	1.840	/	
##			1.000	/	0.000	/	0.080
##			0.104	/	0.000	/	
##			0.080	/	0.000	/	
##	-----	-----	-----	-----	-----	-----	-----
##	Column Total	/	77	/	23	/	100
##			0.770	/	0.230	/	
##	-----	-----	-----	-----	-----	-----	-----

With both tables, we can calculate accuracy and error rate by hand.

```
accuracy<-(69+0)/100
accuracy

## [1] 0.69

error_rate<-(23+8)/100
error_rate

## [1] 0.31

1-accuracy
## [1] 0.31
```

For matrices larger than 2×2 , all diagonal elements are observations that have been correctly classified and off-diagonal elements are those that have been incorrectly classified.

14.2.3 Other Measures of Performance Beyond Accuracy

So far, we discussed two performance methods - table and cross-table. A third function is `confusionMatrix()` which provides the easiest way to report model performance. Notice that the first argument is an *actual vector of the labels*, i.e., `Test_Y`, and the second argument, of the same length, represents the *vector of predicted labels*.

This example was presented as the first case-study in Chap. 9.

```

library(caret)

qol_pred<-predict(qol_model, qol_test)
confusionMatrix(table(qol_pred, qol_test$cd), positive="severe_disease")

## Confusion Matrix and Statistics
##
## 
## qol_pred      minor_disease severe_disease
## minor_disease           149          89
## severe_disease           74         131
##
##                               Accuracy : 0.6321
##                               95% CI : (0.5853, 0.6771)
##                               No Information Rate : 0.5034
##                               P-Value [Acc > NIR] : 3.317e-08
##
##                               Kappa : 0.2637
## McNemar's Test P-Value : 0.2728
##
##                               Sensitivity : 0.5955
##                               Specificity : 0.6682
##                               Pos Pred Value : 0.6390
##                               Neg Pred Value : 0.6261
##                               Prevalence : 0.4966
##                               Detection Rate : 0.2957
##                               Detection Prevalence : 0.4628
##                               Balanced Accuracy : 0.6318
##
##                               'Positive' Class : severe_disease

```

14.2.4 The Kappa (κ) Statistic

The Kappa statistic was originally developed to measure the reliability between two human raters. It can be harnessed in machine-learning applications to compare the accuracy of a classifier, where one rater represents the ground truth (for labeled data, these are the actual values of each instance) and the second rater represents the results of the automated machine-learning classifier. The order of listing the **raters** is irrelevant.

Kappa statistic measures the **possibility of a correct prediction by chance alone** and answers the question of How much better is the agreement (between the ground truth and the machine-learning prediction) than would be expected by chance alone? Its value is between 0 and 1. When $\kappa = 1$, we have a perfect agreement between a **computed** prediction (typically the result of a model-based or model-free technique forecasting an outcome of interest)

and an **expected** prediction (typically random, by chance prediction). A common interpretation of the Kappa statistics includes:

- Poor agreement: less than 0.20
- Fair agreement: 0.20–0.40
- Moderate agreement: 0.40–0.60
- Good agreement: 0.60–0.80
- Very good agreement: 0.80–1

In the above `confusionMatrix` output, we have a fair agreement. For different problems, we may have different interpretations of Kappa statistics. To understand the Kappa statistic better, let's look at its definition:

$$\kappa = \frac{P(a) - P(e)}{1 - P(e)} .$$

$P(a)$ and $P(e)$ simply denote probability of **actual** and **expected** agreement between the classifier and true values.

```
table(qol_pred, qol_test$cd)
##
##   qol_pred      minor_disease  severe_disease
##   minor_disease           149          89
##   severe_disease           74         131
```

According to above table, actual agreement is the accuracy:

```
p_a<- (149+131)/(149+89+74+131)
p_a
## [1] 0.6320542
```

The manually and automatically computed accuracies coincide (0.6321). It may be trickier to obtain the expected agreement. Probability rules tell us that the probability of the union of two disjoint events equals to the sum of the individual (marginal) probabilities for these two events. Thus, we have:

$$P(\text{expect agreement for minor_disease}) = P(\text{actual type is minor_disease}) + P(\text{predicted type is minor_disease})$$

Similarly:

$$P(\text{expect agreement for severe_disease}) = P(\text{actual type is severe_disease}) + P(\text{predicted type is severe_disease}).$$

In our case:

```
p_e_minor <- (149+74)/(149+89+74+131))*((149+89)/(149+89+74+131))
p_e_severe <- ((131+74)/(149+89+74+131)) * ((89+131)/(149+89+74+131))
p_e<-p_e_minor+p_e_severe
p_e
## [1] 0.5002522
```

Plugging in p_a and p_e into the formula we get:

```
kappa<-(p_a-p_e)/(1-p_e)
kappa
## [1] 0.26
```

We get a similar value as the `confusionTable()` output. A more straightforward way of getting the Kappa statistics is by using `Kappa()` function in the `vcd` package.

```
#install.packages(vcd)
library(vcd)

## Loading required package: grid

Kappa(table(qol_pred, qol_test$cd))

##           value      ASE      z Pr(>|z|)
## Unweighted 0.2637 0.04573 5.767 8.071e-09
## Weighted   0.2637 0.04573 5.767 8.071e-09
```

The combination of `Kappa()` and `table` function yields a 2×4 matrix. The *Kappa statistic* is under the unweighted value.

Generally speaking, predicting a severe disease outcome is a more critical problem than predicting a mild disease state. Thus, weighted Kappa is also useful. We give the severe disease a higher weight. The Kappa test result is not acceptable since the classifier may make too many mistakes for the severe disease cases. The Kappa value is only -0.0714 . Notice that the range of Kappa is not $[0,1]$ for the weighted Kappa.

```
Kappa(table(qol_pred, qol_test$cd), weights = matrix(c(1,10,1,10), nrow=2))

##           value      ASE      z Pr(>|z|)
## Unweighted 0.26374 0.04573 5.767 8.071e-09
## Weighted   0.06818 0.04009 1.701 8.898e-02
```

When the predicted value is the first argument, the row and column names represent the **true labels** and the **predicted labels**, respectively.

```
table(qol_pred, qol_test$cd)

##
## qol_pred      minor_disease  severe_disease
##   minor_disease           149            89
##   severe_disease           74            131
```

Summary of the Kappa Score for Calculating Prediction Accuracy

Kappa compares an **Observed classification accuracy** (output of our ML classifier) with an **Expected classification accuracy** (corresponding to random chance classification). It may be used to evaluate single classifiers and/or to compare among a set of different classifiers. It takes into account random chance (agreement with a random classifier). That makes **Kappa** more meaningful than simply using **accuracy** as a metric. For instance, the interpretation of an Observed Accuracy of 80% is **relative** to the Expected Accuracy. Observed Accuracy of 80% is more impactful for an Expected Accuracy of 50% compared to Expected Accuracy of 75%.

14.2.5 Computation of Observed Accuracy and Expected Accuracy

Consider the following example of a classifier generating the following confusion matrix. Columns represent the **true labels** and rows represent the **classifier-derived labels** for this binary prediction example (Table 14.4).

In this example, there is a total of 150 observations ($50 + 35 + 25 + 40$). In reality, 75 are labeled as **True** ($50 + 25$) and another 75 are labeled as **False** ($35 + 40$). The classifier labeled 85 as **True** ($50 + 35$) and the other 65 as **False** ($25 + 40$).

- Observed Accuracy (OA) is the proportion of instances that were classified correctly throughout the entire confusion matrix:

$$OA = \frac{50 + 40}{150} = 0.6.$$

- Expected Accuracy (EA) is the accuracy that any random classifier would be expected to achieve based on the given confusion matrix. EA is the proportion of instances of each class (**True** and **False**), along with the number of instances that the automated classifier agreed with the ground truth label. The EA is calculated by multiplying the marginal frequencies of **True** for the true-state and the machine classified instances, and dividing by the total number of instances. The marginal frequency of **True** for the **true-state** is 75 ($50 + 25$)

Table 14.4 A simulated confusion matrix.

Class	True	False	Total
True	50	35	85
False	25	40	65
Total	75	75	150

and for the corresponding ML classifier is 85 (50 + 35). Then, the expected accuracy for the **True** outcome is:

$$EA(\text{True}) = \frac{75 \times 85}{150} = 42.5.$$

We similarly compute the $EA(\text{False})$ for the second, **False**, outcome, by using the marginal frequencies for the true-state ($(\text{False} \mid \text{true state}) = 75 = 50 + 25$) and the ML classifier ($(\text{False} \mid \text{classifier}) = 65(40 + 25)$). Then, the expected accuracy for the **True** outcome is:

$$EA(\text{False}) = \frac{75 \times 65}{150} = 32.5.$$

Finally, the $EA = \frac{EA(\text{True}) + EA(\text{False})}{150}$

$$\text{ExpectedAccuracy}(EA) = \frac{42.5 + 32.5}{150} = 0.5.$$

Note that $EA = 0.5$ whenever the `true-state` binary classification is balanced (in reality, the frequencies of **True** and **False** are equal, in our case 75).

The calculation of the **kappa statistic** relies on $OA = 0.6$ and $EA = 0.5$:

$$(\text{Kappa}) \kappa = \frac{OA - EA}{1 - EA} = \frac{0.6 - 0.5}{1 - 0.5} = 0.2.$$

14.2.6 Sensitivity and Specificity

If we take a closer look at the `confusionMatrix()` output, we find there are two important statistics “sensitivity” and “specificity”.

Sensitivity, or true positive rate, measures the proportion of “success” observations that are correctly classified.

$$\text{sensitivity} = \frac{TP}{TP + FN}.$$

Notice $TP + FN$ are the total number of true “success” observations.

On the other hand, specificity, or true negative rate, measures the proportion of “failure” observations that are correctly classified.

$$\text{specificity} = \frac{TN}{TN + FP}.$$

Accordingly, $TN + FP$ are the total number of true “failure” observations.

Using the `table()` output above and using "severe_disease" as "success", we can compute these two measures directly.

```
sens<-131/(131+89)
sens
## [1] 0.5954545
spec<-149/(149+74)
spec
## [1] 0.6681614
```

Another R package, `caret`, also provides functions to calculate sensitivity and specificity.

```
library(caret)
sensitivity(qol_pred, qol_test$cd, positive="severe_disease")
## [1] 0.5954545
```

Sensitivity and specificity both range from 0 to 1. For either measure, a value of 1 implies that the positive and negative predictions are very accurate. However, simultaneously high sensitivity and specificity may not be attainable in real world situations. There is a tradeoff between sensitivity and specificity. To compromise, some studies loosen the demands on one and focus on achieving high values on the other.

14.2.7 Precision and Recall

Very similar to sensitivity, *precision* measures the proportion of true "success" observations among predicted "success" observations.

$$\text{precision} = \frac{TP}{TP + FP}.$$

Recall is the proportion of true "positives" among all "true positive" conditions. A model with high recall captures most "interesting" cases.

$$\text{recall} = \frac{TP}{TP + FN}.$$

Again, let's calculate these by hand for the QoL data:

```
prec<-131/(131+74)
prec
## [1] 0.6390244
recall<-131/(131+89)
recall
## [1] 0.5954545
```

Another way to obtain *precision* would be `posPredValue()` under the `caret` package. Remember to specify which one is the “success” class.

```
posPredValue(qol_pred, qol_test$cd, positive="severe_disease")
## [1] 0.6390244
```

From the definitions of **precision** and **recall**, we can derive the type 1 error and type 2 errors as follow:

$$\text{error}_1 = 1 - \text{Precision} = \frac{FP}{TP + FP}, \text{ and}$$

$$\text{error}_2 = 1 - \text{Recall} = \frac{FN}{TP + FN}.$$

Thus, we can compute the type 1 error (0.36) and type 2 error (0.40).

```
error1<-74/(131+74)
error2<-89/(131+89)
error1; error2
## [1] 0.3609756
## [1] 0.4045455
```

14.2.8 The F-Measure

The F-measure or F1-score combines precision and recall using the harmonic mean assuming equal weights. High F-score means high precision and high recall. This is a convenient way of measuring model performances and comparing models.

$$F\text{-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{recall} + \text{precision}} = \frac{2 \times TP}{2 \times TP + FP + FN}.$$

Let’s calculate the F1-score by hand using the confusion matrix derived from the Quality of Life prediction:

```
F1<-(2*prec*recall)/(prec+recall); F1
## [1] 0.6164706
```

The direct calculations of the F1-statistics can be obtained using `caret`:

```
precision <- posPredValue(qol_pred, qol_test$cd, positive="severe_disease")
recall <- sensitivity(qol_pred, qol_test$cd, positive="severe_disease")
F1 <- (2 * precision * recall) / (precision + recall); F1
## [1] 0.6164706
```

14.3 Visualizing Performance Tradeoffs (ROC Curve)

Another choice for evaluating classifiers performance is by using graphs rather than quantitative statistics. Graphs are usually more comprehensive than single statistics.

In R there is a package providing user-friendly functions for visualizing model performance. Details can be found on the ROCR website.

Here, we evaluate the model performance for the Quality of Life case study, see Chap. 9.

```
#install.packages("ROCR")
library(ROCR)

pred<-ROCR::prediction(predictions=pred_prob[, 2], labels=qol_test$cd)
# avoid naming collision (ROCR::prediction), as
# there is another prediction function in neuralnet package.
```

`pred_prob[, 2]` is the probability of classifying each observation as "severe_disease". The above code saved all the model prediction information into object `pred`.

The ROC (Receiver Operating Characteristic) curves are often used to examine the tradeoff between detecting true positives and avoiding the false positives (Fig. 14.1).

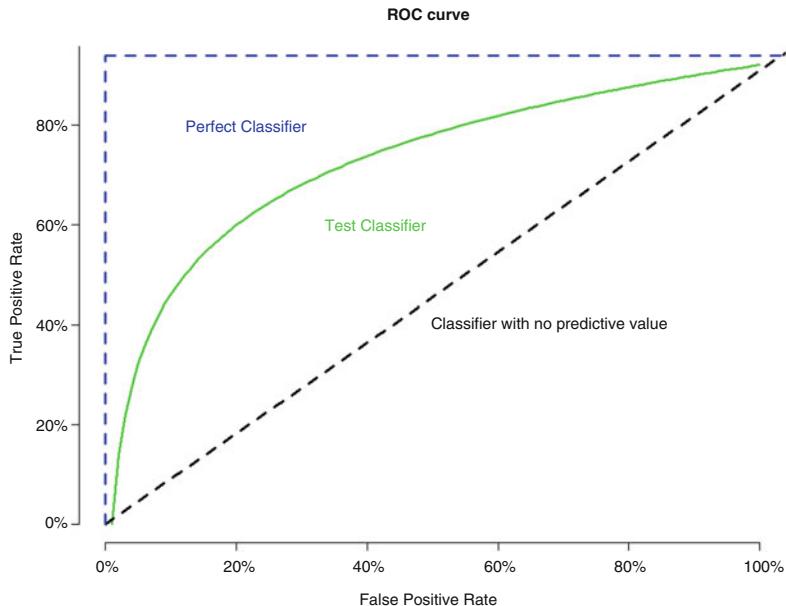


Fig. 14.1 Schematic of quantifying the efficacy of a classification method using the area under the ROC curve

```
curve(log(x), from=0, to=100, xlab="False Positive Rate", ylab="True Positive Rate", main="ROC curve", col="green", lwd=3, axes=F)
Axis(side=1, at=c(0, 20, 40, 60, 80, 100), labels = c("0%", "20%", "40%", "60%", "80%", "100%"))
Axis(side=2, at=0:5, labels = c("0%", "20%", "40%", "60%", "80%", "100%"))
segments(0, 0, 110, 5, lty=2, lwd=3)
segments(0, 0, 0, 4.7, lty=2, lwd=3, col="blue")
segments(0, 4.7, 107, 4.7, lty=2, lwd=3, col="blue")
text(20, 4, col="blue", labels = "Perfect Classifier")
text(40, 3, col="green", labels = "Test Classifier")
text(70, 2, col="black", labels= "Classifier with no predictive value")
```

The **blue line** in the above graph represents the perfect classifier where we have 0% false positive and 100% true positive. The middle **green line** is the test classifier. Most of our classifiers trained by real data will look like this. The black diagonal line illustrates a classifier with no predictive value predicts. We can see that it has the same true positive rate and false positive rate. Thus, it cannot distinguish between the two.

In terms of identifying positive value, we want our ROC curve to be as close to the perfect line as possible. Thus, we measure the area under the ROC curve (abbreviated as AUC) to show how close our curve is to the perfect classifier. To do this, we have to change the scale of the graph above. Mapping 100% to 1, we have a 1×1 square. The area under the perfect classifier would be one, and area under classifier with no predictive value would be 0.5. Then, 1 and 0.5 will be the upper and lower limits for our model ROC curve. We have the following scoring system (numbers indicate area under the curve) for predictive model ROC curves:

- Outstanding: 0.9–1.0
- Excellent/good: 0.8–0.9
- Acceptable/fair: 0.7–0.8
- Poor: 0.6–0.7
- No discrimination: 0.5–0.6.

Note that this rating system is somewhat subjective. Let's use the ROCR package to draw a ROC curve.

```
roc<-performance(pred, measure="tpr", x.measure="fpr")
```

By specifying "tpr"(True positive rate) and "fpr"(False positive rate) we made a "performance" object (Fig. 14.2).

```
plot(roc, main="ROC curve for Quality of Life model", col="blue", lwd=3)
segments(0, 0, 1, 1, lty=2)
```

The segments command draws the dotted line representing the classifier with no predictive value.

To measure this quantitatively, we need to create a new performance object with `measure = "auc"` or area under the curve.

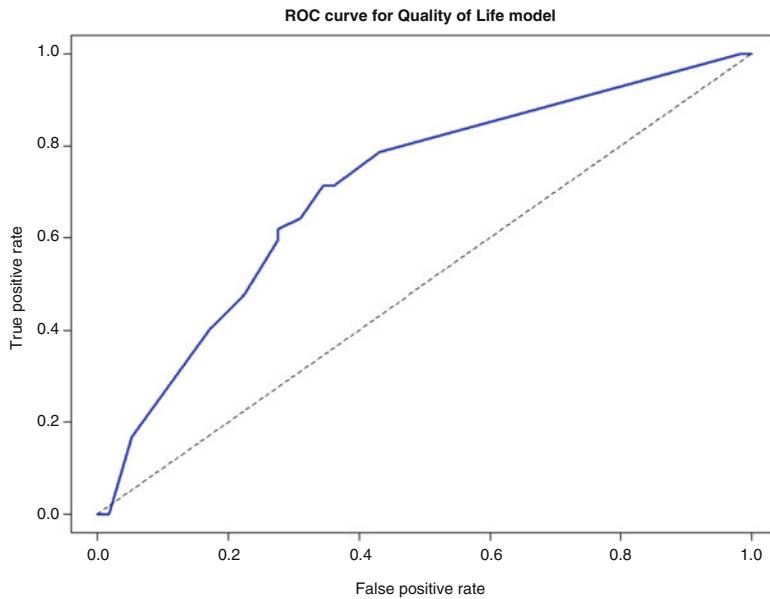


Fig. 14.2 ROC curve of the prediction of disease severity using the quality of life (QoL) data

```
roc_auc<-performance(pred, measure="auc")
```

Now the `roc_auc` is stored as a S4 object. This is quite different than data frame and matrices. First, we can use `str()` function to see its structure.

```
str(roc_auc)
## Formal class 'performance' [package "ROCR"] with 6 slots
##  ..@ x.name      : chr "None"
##  ..@ y.name      : chr "Area under the ROC curve"
##  ..@ alpha.name  : chr "none"
##  ..@ x.values    : List()
##  ..@ y.values    : List of 1
##  ..@ ...$ : num 0.65
##  ..@ alpha.values: List()
```

The ROC object has six members. The AUC value is stored in `y.values`. To extract that we use the `@` symbol according to the output of the `str()` function.

```
roc_auc@y.values
## [[1]]
## [1] 0.6496739
```

Thus, the obtained $AUC = 0.65$, which suggests a fair classifier, according to the above scoring schema.

14.4 Estimating Future Performance (Internal Statistical Validation)

The evaluation methods we have talked about are all measuring re-substitution error. That is, building the model on training data and measuring the model error on separate testing data. This is one way of dealing with unseen data. First, let's introduce the basic ideas, and more details will be presented in Chap. 21.

14.4.1 The Holdout Method

The idea is to partition the entire dataset into two separate datasets, using one of them to create the model and the other to test the model performances. In practice, we usually use a fraction (e.g., 50%, or $\frac{2}{3}$) of our data for training the model, and reserve the rest (e.g., 50%, or $\frac{1}{3}$) for testing. Note that the testing data may also be further split into proportions for internal repeated (e.g., cross-validation) testing and final external (independent) testing.

The partition has to be randomized. In R, the best way of doing this is to create a parameter that randomly draws numbers and use this parameter to extract random rows from the original dataset. In Chap. 11, we used this method to partition the *Google Trends* data.

```
sub<-sample(nrow(google_norm), floor(nrow(google_norm)*0.75))
google_train<-google_norm[sub, ]
google_test<-google_norm[-sub, ]
```

Another way of partitioning is by using `createDataPartition()` under the `caret` package. Instead of using the entire original dataset, we can use the outcome variable, `google_norm$RealEstate`, or any of the independent variables.

```
sub<-createDataPartition(google_norm$RealEstate, p=0.75, list = F)
google_train<-google_norm[sub, ]
google_test<-google_norm[-sub, ]
```

To make sure that the model can be applied to future datasets, we can partition the original dataset into three separate subsets. In this way, we have two subsets for testing. The additional validation dataset can alleviate the probability that we have a good model due to chance (non-representative subsets). A common split among training, test, and validation subsets would be 50%, 25%, and 25% respectively.

```

sub<-sample(nrow(google_norm), floor(nrow(google_norm)*0.50))
google_train<-google_norm[sub, ]
google_test<-google_norm[-sub, ]
sub1<-sample(nrow(google_test), floor(nrow(google_test)*0.5))
google_test1<-google_test[sub1, ]
google_test2<-google_test[-sub1, ]
nrow(google_norm)

## [1] 731
nrow(google_train)

## [1] 365
nrow(google_test1)

## [1] 183
nrow(google_test2)

## [1] 183

```

However, when we only have a very small dataset, it's difficult to split off too much data as this reduces the sample further. There are the following two options for evaluation of model performance using (independent) unseen data: cross-validation and holdout methods. These are implemented in the `caret` package.

14.4.2 Cross-Validation

For complete details see DSPA Cross-Validation (Chap. 21). Below, we describe the fundamentals of cross-validation as an internal statistical validation technique.

This technique is known as *k-fold cross-validation* or *k-fold CV*, which is a standard for estimating model performance. K-fold CV randomly divides the original data into *k* separate random subsets called folds.

A common practice is to use $k = 10$ or 10-fold CV to split the data into 10 different subsets. Each time using one of the subsets to be the test set and the rest to build the model. `createFolds()` under `caret` package will help us to do so. `set.seed()` insures the folds created are the same if you run the code line twice. 1234 is just a random number. You can use any number for `set.seed()`. We use the normalized Google Trend dataset in this section.

```

library("caret")
set.seed(1234)
folds<-createFolds(google_norm$RealEstate, k=10)
str(folds)
## List of 10
## $ Fold01: int [1:73] 5 9 11 12 18 19 28 29 54 65 ...
## $ Fold02: int [1:73] 14 24 35 49 52 61 63 76 99 115 ...
## $ Fold03: int [1:73] 1 8 41 45 51 74 78 92 100 104 ...

```

```
## $ Fold04: int [1:73] 30 32 37 40 43 57 59 64 70 96 ...
## $ Fold05: int [1:73] 13 16 25 53 56 68 77 81 93 95 ...
## $ Fold06: int [1:73] 4 6 15 20 36 69 71 73 79 89 ...
## $ Fold07: int [1:73] 34 42 44 84 90 98 102 110 112 117 ...
## $ Fold08: int [1:73] 2 3 48 62 82 85 86 87 88 91 ...
## $ Fold09: int [1:74] 10 21 23 27 33 39 46 55 58 75 ...
## $ Fold10: int [1:73] 7 17 22 26 31 38 47 50 60 66 ...
```

Another way to cross-validate is to use `cv_partition()` in package `sparsediscrim`.

```
# install.packages("sparsediscrim")
require(sparsediscrim)
folds2 = cv_partition(1:nrow(google_norm), num_folds=10)
```

And the structure of folds may be reported by:

```
str(folds2)
## List of 10
## $ Fold1 :List of 2
##   ..$ training: int [1:657] 4 5 6 8 9 10 11 12 16 17 ...
##   ..$ test   : int [1:74] 287 3 596 1 722 351 623 257 568 414 ...
## $ Fold2 :List of 2
##   ..$ training: int [1:658] 1 2 3 5 6 7 8 9 10 11 ...
##   ..$ test   : int [1:73] 611 416 52 203 359 195 452 258 614 121 ...
## $ Fold3 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 7 8 9 10 11 ...
##   ..$ test   : int [1:73] 182 202 443 152 486 229 88 158 178 293 ...
## $ Fold4 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ test   : int [1:73] 646 439 362 481 183 387 252 520 438 586 ...
## $ Fold5 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ test   : int [1:73] 503 665 47 603 348 125 719 11 461 361 ...
## $ Fold6 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 6 7 9 10 11 12 ...
##   ..$ test   : int [1:73] 666 411 159 21 565 298 537 262 131 600 ...
## $ Fold7 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ test   : int [1:73] 269 572 410 488 124 447 313 255 360 473 ...
## $ Fold8 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 9 11 ...
##   ..$ test   : int [1:73] 446 215 256 116 592 284 294 300 402 455 ...
## $ Fold9 :List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 9 10 ...
##   ..$ test   : int [1:73] 25 634 717 545 76 378 53 194 70 346 ...
## $ Fold10:List of 2
##   ..$ training: int [1:658] 1 2 3 4 5 6 7 8 10 11 ...
##   ..$ test   : int [1:73] 468 609 40 101 595 132 248 524 376 618 ...
```

Now, we have 10 different subsets in the `folds` object. We can use `lapply()` to fit the model. 90% of data will be used for training so we use `[-x,]` to represent

all observations not in a specific fold. In Chap. 11 we showed building a neutral network model for the *Google Trends* data. We can do the same for each fold manually; train, test, aggregate the results, and report the agreement (correlations between the predicted and observed RealEstate values).

```
library(neuralnet)

fold_cv<-lapply(folds, function(x){
  google_train<-google_norm[-x, ]
  google_test<-google_norm[x, ]
  google_model<-neuralnet(RealEstate~Unemployment+Rental+Mortgage+Jobs+Investing+DJIA_Index+StdDJI, data=google_train)
  google_pred<-compute(google_model, google_test[, c(1:2, 4:8)])
  pred_results<-google_pred$net.result
  pred_cor<-cor(google_test$RealEstate, pred_results)
  return(pred_cor)
})
str(fold_cv)

## List of 10
## $ Fold01: num [1, 1] 0.977
## $ Fold02: num [1, 1] 0.97
## $ Fold03: num [1, 1] 0.972
## $ Fold04: num [1, 1] 0.979
## $ Fold05: num [1, 1] 0.976
## $ Fold06: num [1, 1] 0.974
## $ Fold07: num [1, 1] 0.971
## $ Fold08: num [1, 1] 0.982
## $ Fold09: num [1, 1] -0.516
## $ Fold10: num [1, 1] 0.974
```

From the output, we know that in most of the folds the model predicts very well. In a typical run, one fold may yield bad results. We can use the *mean* of these 10 correlations to represent the *overall* model performance. But first, we need to use *unlist* () function to transform *fold_cv* into a vector.

```
mean(unlist(fold_cv))

## [1] 0.8258223801
```

This correlation is high, suggesting strong association between predicted and true values. Thus, the model is very good in terms of its prediction.

14.4.3 *Bootstrap Sampling*

The second method is called *bootstrap sampling*. In k-fold CV, each observation can only be used once. However, bootstrap sampling is a sampling process *with replacement*. Before selecting a new sample, it recycles every observation so that each observation could appear in multiple folds.

A very special setting of bootstrap uses at each iteration 63.2% of the original data as our training dataset and the remaining 36.8% as the test dataset. Thus, compared to k-fold CV, bootstrap sampling is less representative of the full dataset. A special case of bootstrapping, *0.632 bootstrap*, addresses this issue by changing the final performance metric using the following formula:

$$\text{error} = 0.632 \times \text{error}_{\text{test}} + 0.368 \times \text{error}_{\text{train}}.$$

This synthesizes the optimistic model performance on training data with the pessimistic model performance on test data by weighting the corresponding errors. This method is extremely good for small samples.

To see the rationale behind *0.632 bootstrap*, consider a standard training set T of cardinality n , where our bootstrap sampling generates m new training sets T_i , each of size n' . Sampling from T is uniform *with replacement*, suggests that some observations may be repeated in each sample T_i . Suppose the size of the sub-samples are of the same order as T , i.e., $n' = n$, then for large n the sample D_i is *expected* to have $(1 - \frac{1}{e}) \sim 0.632$ unique cases from the complete original collection T , the remaining proportion 0.368 are expected to be repeated duplicates. Hence, the name *0.632 bootstrap* sampling. In general, for large $n \gg n'$, the sample D_i is *expected* to have $n \left(1 - e^{-\frac{n'}{n}}\right)$ unique cases, see On Estimating the Size and Confidence of a Statistical Audit).

Having the bootstrap samples, the m models can be fitted (estimated) and aggregated, e.g., by averaging the outputs (for regression) or by using voting methods (for classification). We will discuss this more in later chapters.

Try to apply the same techniques to some of the other data in the list of Case-Studies.

14.5 Assignment: 14. Evaluation of Model Performance

The ABIDE dataset includes imaging, clinical, genetics and phenotypic data for over 1000 pediatric cases – *Autism Brain Imaging Data Exchange* (ABIDE).

- Apply *C5.0* to predict on part of data (training data).
- Evaluate the model's performance, using confusion matrices, accuracy, κ , precision, and recall, F-measure, etc.
- Explain and compare each evaluation.
- Use the ROC to examine the tradeoff between detecting true positives and avoiding the false positives and report AUC.
- Finally, apply cross validation on *C5.0* and report the CV error.
- You may apply the same analysis workflow to evaluate the performance of alternative methods (e.g., KNN, SVM, LDA, QDA, Neural Networks, etc.)

References

- SciKit: <http://scikit-learn.org/stable/modules/classes.html>
- Sammut, C, Webb, GI (eds.) (2011) Encyclopedia of Machine Learning, Springer Science & Business Media, ISBN 0387307680, 9780387307688.
- Japkowicz, N, Shah. M. (2011) Evaluating Learning Algorithms: A Classification Perspective, Cambridge University Press, ISBN 1139494147, 9781139494144.

Chapter 15

Improving Model Performance



We already explored several alternative machine learning (ML) methods for prediction, classification, clustering, and outcome forecasting. In many situations, we derive models by estimating model coefficients or parameters. The main question now is *How can we adopt the advantages of crowdsourcing and biosocial networking to aggregate different predictive analytics strategies?* Are there reasons to believe that such **ensembles** of forecasting methods may actually improve the performance (e.g., increase prediction accuracy) of the resulting consensus meta-algorithm? In this chapter, we are going to introduce ways that we can search for optimal parameters for a single ML method, as well as aggregate different methods into **ensembles** to enhance their collective performance relative to any of the individual methods part of the meta-aggregate.

After we summarize the core methods, we will present automated and customized parameter tuning, and show strategies for improving model performance based on meta-learning via bagging and boosting.

15.1 Improving Model Performance by Parameter Tuning

One of the methods for improving model performance relies on *tuning*, which is the process of searching for the best parameters for a specific method. Table 15.1 summarizes the parameters for each method we covered in previous chapters.

15.2 Using `caret` for Automated Parameter Tuning

In Chap. 7, we used KNN and plugged in random k parameters for the number of clusters. This time, we will test multiple k values simultaneously and pick the one with the highest accuracy. When using the `caret` package, we need to specify a

Table 15.1 Synopsis of the basic prediction, classification and clustering methods and their core parameters

Model	Learning task	Method	Parameters
KNN	Classification	<code>class::knn</code>	<code>data, k</code>
K-Means	Classification	<code>stats::kmeans</code>	<code>data, k</code>
Naïve Bayes	Classification	<code>e1071::naiveBayes</code>	<code>train, class, laplace</code>
Decision Trees	Classification	<code>C50::C5.0</code>	<code>train, class, trials, costs</code>
OneR Rule Learner	Classification	<code>RWeka::OneR</code>	<code>class~predictors, data</code>
RIPPER Rule Learner	Classification	<code>RWeka::JRip</code>	<code>formula, data, subset, na.action, control, options</code>
Linear Regression	Regression	<code>stats::lm</code>	<code>formula, data, subset, weights, na.action, method</code>
Regression Trees	Regression	<code>rpart::rpart</code>	<code>dep_var ~ indep_var, data</code>
Model Trees	Regression	<code>RWeka::M5P</code>	<code>formula, data, subset, na.action, control</code>
Neural Networks	Dual use	<code>nnet::nnet</code>	<code>x, y, weights, size, Wts, mask, linout, entropy, softmax, censored, skip, rang, decay, maxit, Hess, trace, MaxNWts, abstol, reltol</code>
Support Vector Machines (Polynomial Kernel)	Dual use	<code>caret::train::svmLinear</code>	<code>C</code>
Support Vector Machines (Radial Basis Kernel)	Dual use	<code>caret::train::svmRadial</code>	<code>C, sigma</code>
Support Vector Machines (general)	Dual use	<code>kernlab::ksvm</code>	<code>formula, data, kernel</code>
Random Forests	Dual use	<code>randomForest::randomForest</code>	<code>formula, data</code>

class variable, a dataset containing a class variable, predicting features, and the method we will be using. In Chap. 7, we used the Boys Town Study of Youth Development dataset, normalized all the features, stored them in `boystown_n`, and formulated the outcome class variable first (`boystown$grade`).

```

str(boystown_n)

## 'data.frame': 200 obs. of 10 variables:
## $ sex      : num 0 0 0 0 1 1 0 0 1 1 ...
## $ gpa       : num 1 0 0.6 0.4 0.6 0.6 0.2 1 0.2 0.6 ...
## $ Alcoholuse: num 0.182 0.364 0.182 0.182 0.545 ...
## $ alcatt    : num 0.5 0.333 0.5 0.167 0.333 ...
## $ dadjob   : num 1 1 1 1 1 1 1 1 1 ...
## $ momjob   : num 0 0 0 1 0 0 0 1 1 ...
## $ dadclose  : num 0.143 0.429 0.286 0.143 0.286 ...
## $ momclose  : num 0.143 0.571 0.286 0.286 0.143 ...
## $ Larceny   : num 0.25 0 0 0.75 0.25 0 0 0.25 0.25 ...
## $ vandalism : num 0.429 0 0.286 0.286 0.286 ...

boystown_n<-cbind(boystown_n, boystown[, 11])
str(boystown_n)

## 'data.frame': 200 obs. of 11 variables:
## $ sex      : num 0 0 0 0 1 1 0 0 1 1 ...
## $ gpa       : num 1 0 0.6 0.4 0.6 0.6 0.2 1 0.2 0.6 ...
## $ Alcoholuse: num 0.182 0.364 0.182 0.182 0.545 ...
## $ alcatt    : num 0.5 0.333 0.5 0.167 0.333 ...
## $ dadjob   : num 1 1 1 1 1 1 1 1 1 ...
## $ momjob   : num 0 0 0 1 0 0 0 1 1 ...
## $ dadclose  : num 0.143 0.429 0.286 0.143 0.286 ...
## $ momclose  : num 0.143 0.571 0.286 0.286 0.143 ...
## $ Larceny   : num 0.25 0 0 0.75 0.25 0 0 0.25 0.25 ...
## $ vandalism : num 0.429 0 0.286 0.286 0.286 ...
## $ boystown[, 11]: Factor w/ 2 Levels "above_avg", "avg_or_below": 2 1 2 1
2 2 1 2 1 2 ...

colnames(boystown_n)[11]<- "grade"

```

The dataset including a specific class variable and predictive features is now successfully created. We are using the KNN method as an example with the class variable `grade`. So, we plug this information into the `caret::train()` function. Note that `caret` is using the full dataset because it will automatically do the random sampling for us. To make the results reproducible, we utilize the `set.seed()` function that we previously used, see Chap. 14.

```

library(caret)

set.seed(123)
m<-train(grade~, data=boystown_n, method="knn")
m; summary(m)

## k-Nearest Neighbors
##
## 200 samples
## 10 predictor
## 2 classes: 'above_avg', 'avg_or_below'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
```

```

##   k  Accuracy  Kappa
##   5  0.7952617  0.5193402
##   7  0.8143626  0.5585191
##   9  0.8070520  0.5348281
##
## Accuracy was used to select the optimal model using the Largest value.
## The final value used for the model was k = 7.

##           Length Class      Mode
## Learn         2   -none-   List
## k            1   -none- numeric
## theDots       0   -none-  List
## xNames        10  -none- character
## problemType   1  -none- character
## tuneValue     1   data.frame List
## obsLevels     2   -none- character

```

In this case, using `str(m)` to summarize the object `m` may report out too much information. Instead, we can simply type the object name `m` to get more concise information about it.

1. Description about the dataset: number of samples, features, and classes.
2. Re-sampling process: here, we use 25 bootstrap samples with 200 observations (same size as the observed dataset) each to train the model.
3. Candidate models with different parameters that have been evaluated: by default, `caret` uses 3 different choices for each parameter, but for binary parameters, it only allows two choices, `TRUE` and `FALSE`. As KNN has only one parameter `k`, we have three candidate models reported in the output above.
4. Optimal model: the model with largest accuracy is the one corresponding to `k=5`.

Let's see how accurate this "optimal model" is in terms of the re-substitution error. Again, we will use the `predict()` function specifying the object `m` and the dataset `boystown_n`. Then, we can report the contingency table showing the agreement between the predictions and real class labels.

```

set.seed(1234)
p<-predict(m, boystown_n)
table(p, boystown_n$grade)

##
##   p           above_avg avg_or_below
##   above_avg        132        17
##   avg_or_below      2        49

```

This model has $(17 + 2)/200 = 0.09$ re-substitution error (9%). This means that in the 200 observations that we used to train this model, 91% of them were correctly classified. Note that re-substitution error is different from accuracy. The accuracy of this model is 0.8, which is reported by a model summary call. As mentioned in Chap. 14, we can obtain prediction probabilities for each observation in the original `boystown_n` dataset.

```
head(predict(m, boystown_n, type = "prob"))

##   above_avg avg_or_below
## 1 0.0000000 1.0000000
## 2 1.0000000 0.0000000
## 3 0.7142857 0.2857143
## 4 0.8571429 0.1428571
## 5 0.2857143 0.7142857
## 6 0.5714286 0.4285714
```

15.2.1 Customizing the Tuning Process

The default setting of `train()` might not meet the specific needs for every study. In our case, the optimal k might be smaller than 5. The `caret` package allows us to customize the settings for `train()`.

`caret::trainControl()` can help us to customize re-sampling methods. There are 6 popular re-sampling methods that we might want to use in the following table (Table 15.2).

These methods are helping us find representative samples to train the model. Let's use *0.632 bootstrap* for example. Just specify `method="boot632"` in the `trainControl()` function. The number of different samples to include can be customized by `number=` option. Another option in `trainControl()` is about the model performance evaluation. We can change our preferred method of evaluation to select the optimal model. The `oneSE` method chooses the simplest model within one standard error of the best performance to be the optimal model. Other methods are also available in `caret` package. For detailed information, type `best` in R console.

We can also specify a list of k values we want to test by creating a matrix or a grid.

```
ctrl<-trainControl(method="boot632", number=25, selectionFunction="oneSE")
grid<-expand.grid(.k=c(1, 3, 5, 7, 9))
# Creates a data frame from all combinations of the supplied factors
```

Table 15.2 Six complementary methods for customizing the `caret::trainControl()` re-sampling

Resampling method	Method name	Additional options and default values
Holdout sampling	LGOCV	<code>p = 0.75</code> (training data proportion)
k-fold cross-validation	cv	<code>number = 10</code> (number of folds)
Repeated k-fold cross validation	repeatedcv	<code>number = 10</code> (number of folds), <code>repeats = 10</code> (number of iterations)
Bootstrap sampling	boot	<code>number = 25</code> (resampling iterations)
0.632 bootstrap	boot632	<code>number = 25</code> (resampling iterations)
Leave-one-out cross-validation	LOOCV	None

Usually, to avoid ties, we prefer to choose an odd number of clusters k . Now the constraints are all set. We can start to select models again using `train()`.

```
set.seed(123)
m<-train(grade~., data=boystown_n, method="knn",
          metric="Kappa",
          trControl=ctrl,
          tuneGrid=grid)
m

## k-Nearest Neighbors
##
## 200 samples
## 10 predictor
## 2 classes: 'above_avg', 'avg_or_below'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   1  0.8726660  0.7081751
##   3  0.8457584  0.6460742
##   5  0.8418226  0.6288675
##   7  0.8460327  0.6336463
##   9  0.8381961  0.6094088
##
## Kappa was used to select the optimal model using the one SE rule.
## The final value used for the model was k = 1.
```

Here we added `metric="Kappa"` to include the *Kappa statistics* as one of the criteria to select the optimal model. We can see the output accuracy for all the candidate models are better than the default bootstrap sampling. The optimal model has $k=3$, a high accuracy 0.846, and a high Kappa statistic, which is much better than the model we had in Chap. 7. As you can see from the output, the SE rule no longer chooses the model with the highest accuracy or Kappa statistic to be the “optimal model”. It is a more comprehensive method than only looks at one statistic or a single quality measure.

15.2.2 *Improving Model Performance with Meta-learning*

Meta-learning involves building multiple learners (can be single or multiple learning algorithms) at the same time. It combines the output from these learners and generates more effective meta-classifiers.

To decrease the variance (bagging) or bias (boosting), *random forests* attempt in two steps to correct the general decision trees' trend to overfit the model to the training set:

1. Producing a distribution of simple ML models on subsets of the original data.
2. Combining the distribution into one “aggregated” model.

Before stepping into the details, let's briefly summarize:

- *Bagging* (stands for Bootstrap Aggregating) is a way to decrease the variance of your prediction by generating additional data for training from your original dataset. It generates multiple sets of the same cardinality/size as your original data, as combinations with repetitions. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to the expected outcome.
- *Boosting* is a two-step approach, where one first uses subsets of the original data to produce a series of moderately performing models and then “boosts” their performance by combining them together using a particular cost function (e.g., Accuracy). Unlike bagging, in classical boosting, the subset creation is not random and depends upon the performance of the previous models: every new subset contains the elements that were (likely to be) misclassified by previous models. Usually, we prefer weaker classifiers in boosting. For example, a prevalent choice is to use stump (level-one decision tree) in AdaBoost (Adaptive Boosting).

15.2.3 Bagging

One of the most well-known meta-learning method is bootstrap aggregating or *bagging*. It builds multiple models with bootstrap samples using a single algorithm. The models' predictions are combined with voting (for classification) or averaging (for numeric prediction). Voting means that bagging model's prediction is based on the majority of learners' predictions for a class. Bagging is especially good with unstable learners like decision trees or SVM models.

To illustrate the Bagging method, we will again use the Quality of Life and chronic disease dataset in Chap. 9. Just like we did in the second practice problem in Chap. 11, we will use CHARLSONSCORE as the classes labels, which has 11 different class labels.

```
qol<-read.csv("https://umich.instructure.com/files/481332/download?download_frd=1")
qol<-qol[!qol$CHARLSONSCORE==9, -c(1, 2)]
qol$CHARLSONSCORE<-as.factor(qol$CHARLSONSCORE)
```

To apply `bagging()`, we need to download the `ipred` package first. After loading the package, we build a bagging model with CHARLSONSCORE as class

label and all other variables in the dataset as predictors. We can specify the number of voters (decision tree models we want to have), the default number is 25.

```
# install.packages("ipred")
library(ipred)
set.seed(123)
mybag<-bagging(CHARLSONSCORE~, data=qol, nbagg=25)
```

Next, we shall use the `predict()` function to apply this model for prediction. For evaluation purposes, we create a table reporting the re-substitution error.

```
bt_pred<-predict(mybag, qol)
agreement<-bt_pred==qol$CHARLSONSCORE
prop.table(table(agreement))

## agreement
##      FALSE      TRUE
## 0.001718213 0.998281787
```

This model works very well with its training data. It labeled 99.8% of the cases correctly. To see its performances on feature data, we apply the `caret train()` function again with 10 repeated CV as re-sampling method. In `caret`, bagged trees method is called `treebag`.

```
library(caret)
set.seed(123)
ctrl<-trainControl(method="repeatedcv", number = 10, repeats = 10)
train(CHARLSONSCORE~, data=as.data.frame(qol), method="treebag", trControl=
ctrl)

## Bagged CART
##
## 2328 samples
##   38 predictor
##   11 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 2095, 2096, 2093, 2094, 2098, 2097, ...
## Resampling results:
##
##   Accuracy   Kappa
## 0.5234615 0.2173193
```

We got an accuracy of 52% and a fair Kappa statistics. This result is better than our previous prediction attempt in Chap. 11 using the `ksvm()` function alone (~50%). Here, we combined the prediction results of 38 decision trees to get this level of prediction accuracy.

In addition to decision tree classification, `caret` allows us to explore alternative `bag()` functions. For instance, instead of bagging based on decision trees, we can bag using an SVM model. `caret` provides a nice setting for SVM training, making

predictions and counting votes in a list object `svmBag`. We can examine these objects by using the `str()` function.

```
str(svmBag)
## List of 3
## $ fit      :function (x, y, ...)
## $ pred     :function (object, x)
## $ aggregate:function (x, type = "class")
```

Clearly, `fit` provides the training functionality, `pred` the prediction and forecasting on new data, and `aggregate` is a way to combine many models and achieve voting-based consensus. Using the member operator, the `$` sign, we can explore these three types of elements of the `svmBag` object. For instance, the `fit` element may be extracted from the `SVM` object by:

```
svmBag$fit
## function (x, y, ...)
## {
##   LoadNamespace("kernlab")
##   out <- kernlab::ksvm(as.matrix(x), y, prob.model = is.factor(y),
##   ...)
##   out
## }
## <environment: namespace:caret>
```

`fit` relies on the `ksvm()` function in the `kernlab` package, which means this package needs to be loaded. The other two methods, `pred` and `aggregate`, may be explored in a similar way. They just follow the SVM model building and testing process we discussed in Chap. 11.

This `svmBag` object could be used as an optional setting in the `train()` function. However, this option requires that all features are linearly independent with trivial covariances, which may be rare in real world data.

15.2.4 Boosting

Bagging uses equal weights for all learners we included in the model. Boosting is quite different in terms of weights. Suppose we have the first learner correctly classifying 60% of the observations. This 60% of data may be less likely to be included in the training dataset for the next learner. So, we have more learners working on “hard-to-classify” observations.

Mathematically, we are using a weighted sum of functions to predict the outcome class labels. We can try to fit the true model using weighted additive modeling. We start with a random learner that can classify some of the observations correctly, possibly with some errors.

$$\hat{y}_1 = l_1.$$

This l_1 is our first learner and \hat{y}_1 denotes its predictions (this equation is in matrix form). Then, we can calculate the residuals of our first learner.

$$\epsilon_1 = y - v_1 \times \hat{y}_1,$$

where v_1 is a shrinkage parameter to avoid overfitting. Next, we fit the residual with another learner. This learner minimizes the following function $\sum_{i=1}^N \|y_i - L_{k-1} - l_k\|$, here $k=2$. Then we obtain a second model l_2 with:

$$\hat{y}_2 = l_2.$$

After that, we can update the residuals:

$$\epsilon_2 = \epsilon_1 - v_2 \times \hat{y}_2.$$

We repeat this residual fitting until adding another learner l_k results in an updated residual ϵ_k that is smaller than a small predefined threshold. At the end, we will have an additive model like:

$$L = v_1 \times l_1 + v_2 \times l_2 + \dots + v_k \times l_K,$$

where we have k weak learners, but a very strong ensemble model.

Schapire and Freund found that although individual learners trained on the pilot observations might be very weak in predicting in isolation, boosting the collective power of all of them is expected to generate a model no worse than the best of all individual constituent models included in the boosting ensemble. Usually, the boosting results are quite a bit better than the best single model.

Boosting can be used for almost all models. Most commonly, it is applied to decision trees.

15.2.5 Random Forests

Random forests, or decision tree forests, represent a boosting method focusing on decision tree learners.

Training Random Forests

One approach to train and build random forests relies on using `randomForest()` under the `randomForest` package. It has the following components:

```
m<-randomForest(expression, data, ntree=500, mtry=sqrt(p))
```

- *expression*: the class variable and features we want to include in the model.
- *data*: training data containing class and features.
- *ntree*: number of voting decision trees.
- *mtry*: optional integer specifying the number of features to randomly select at each split. The *p* stands for number of features in the data.

Let's build a random forest using the Quality of Life dataset.

```
# install.packages("randomForest")
library(randomForest)

set.seed(123)
rf<-randomForest(CHARLSONSCORE~, data=qol)
rf

##
## Call:
## randomForest(formula = CHARLSONSCORE ~ ., data = qol)
##                 Type of random forest: classification
##                         Number of trees: 500
## No. of variables tried at each split: 6
##
##                 OOB estimate of  error rate: 46.13%
## Confusion matrix:
##      0 1 2 3 4 5 6 7 8 9 10 class.error
## 0 574 301 2 0 0 0 0 0 0 0 0 0.3454960
## 1 305 678 1 0 0 0 0 0 0 0 0 0.3109756
## 2 90 185 2 0 0 0 0 0 0 0 0 0.9927798
## 3 25 101 1 0 0 0 0 0 0 0 0 1.0000000
## 4 5 19 0 0 0 0 0 0 0 0 0 1.0000000
## 5 3 4 0 0 0 0 0 0 0 0 0 1.0000000
## 6 1 4 0 0 0 0 0 0 0 0 0 1.0000000
## 7 1 1 0 0 0 0 0 0 0 0 0 1.0000000
## 8 7 8 0 0 0 0 0 0 0 0 0 1.0000000
## 9 3 5 0 0 0 0 0 0 0 0 0 1.0000000
## 10 1 1 0 0 0 0 0 0 0 0 0 1.0000000
```

By default the model contains 500 decision trees and tried 6 variables at each split. Its OOB, or out-of-bag, error rate is about 46%, which corresponds to a poor accuracy rate (54%). Note that the OOB error rate is not re-substitution error. The confusion matrix next to it is reflecting OOB error rate for specific classes. All of these error rates are reasonable estimates of future performances with unseen data. We can see that this model is so far the best of all models, although it is still not good at predicting high CHARLSONSCORE.

Evaluating Random Forest Performance

The *caret* package also supports random forest model building and evaluation. It reports more detailed model performance evaluations. As usual, we need to specify a re-sampling method and a parameter grid. As an example, we use the 10-fold CV

re-sampling method. The grid for this model contains information about the `mtry` parameter (the only tuning parameter for random forest). Previously we tried the default value $\sqrt{38} = 6$ (38 is the number of features). This time we could compare multiple `mtry` parameters.

```
library(caret)
ctrl<-trainControl(method="cv", number=10)
grid_rf<-expand.grid(.mtry=c(2, 4, 8, 16))
```

Next, we apply the `train()` function with our `ctrl` and `grid_rf` settings.

```
set.seed(123)
m_rf <- train(CHARLSONSCORE ~ ., data = qol, method = "rf",
metric = "Kappa", trControl = ctrl,
tuneGrid = grid_rf)
m_rf

## Random Forest
##
## 2328 samples
##   38 predictor
##   11 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2095, 2096, 2093, 2094, 2098, 2097, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.5223871  0.1979731
##   4     0.5403799  0.2309963
##   8     0.5382674  0.2287595
##  16    0.5421562  0.2367477
##
## Kappa was used to select the optimal model using the Largest value.
## The final value used for the model was mtry = 16.
```

This call may take a while to complete. The result appears to be a good model, when `mtry=16` we reached a relatively high accuracy and good kappa statistic. This is a very good result for a learner with 11 classes.

15.2.6 Adaptive Boosting

We may achieve even higher accuracy using **AdaBoost**. Adaptive boosting (AdaBoost) can be used in conjunction with many other types of learning algorithms to improve their performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the

boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by the previous classifiers.

For binary classification, we could use `ada()` in package `ada` and for multiple classes (multinomial/polytomous outcomes) we can use the package `adabag`. The `boosting()` function allows us to select a type method by setting `coeflearn`. Two prevalent types of adaptive boosting methods can be used. One is AdaBoost.M1 algorithm including Breiman and Freund, and the other is Zhu's SAMME algorithm. Let's see some examples:

```
set.seed(123)
qol<-read.csv("https://umich.instructure.com/files/481332/download?download_
frd=1")
qol<-qol[!qol$CHARLSONSCORE==9, -c(1, 2)]
qol$CHARLSONSCORE<-as.factor(qol$CHARLSONSCORE)
```

The key parameter in the `adabag::boosting()` method is `coeflearn`:

- *Breiman* (default), corresponding to $\alpha = \frac{1}{2} \times \ln\left(\frac{1-err}{err}\right)$, using the AdaBoost.M1 algorithm, where α is the weight updating coefficient
- *Freund*, corresponding to $\alpha = \ln\left(\frac{1-err}{err}\right)$, or
- *Zhu*, corresponding to $\alpha = \ln\left(\frac{1-err}{err}\right) + \ln(n\text{classes} - 1)$.

The generalizations of AdaBoost for multiple classes (≥ 2) include AdaBoost.M1 (where individual trees are required to have an error $< \frac{1}{2}$) and SAMME (where individual trees are required to have an error $< 1 - \frac{1}{n\text{classes}}$).

```
# install.packages("ada"); install.packages("adabag")
library("ada"); library("adabag")

qol_boost <- boosting(CHARLSONSCORE~., data=qol, mfinal = 100, coeflearn =
'Breiman')
mean(qol_boost$class==qol$CHARLSONSCORE)

## [1] 0.5425258

qol_boost <- boosting(CHARLSONSCORE~., data=qol, mfinal = 100, coeflearn =
'Freund')
mean(qol_boost$class==qol$CHARLSONSCORE)

## [1] 0.5524055

qol_boost <- boosting(CHARLSONSCORE~., data=qol, mfinal = 100, coeflearn =
'Zhu')
mean(qol_boost$class==qol$CHARLSONSCORE)

## [1] 0.6542096
```

We observe that in this case, the Zhu approach achieves the best results. Notice that the default method is M1 Breiman and `mfinal` is the number of iterations for which boosting is run or the number of trees to use.

Try applying model improvement techniques using other data from the list of our Case-Studies (Fig. 15.1).

15.3 Assignment: 15. Improving Model Performance

Use some of the methods below to do classification, prediction, and model performance evaluation (Table 15.3).

This is a live demo of the Iris flowers classification using adabag-package:
(Applies Multiclass AdaBoost.M1, SAMME and Bagging algorithm)

<https://rdr.io/cran/adabag/man/adabag-package.html>



Examples

```
library(adabag)
# rpart library should be loaded
iris.adaboost <- boostitg(Species~, data=iris, boos=TRUE,
                           mfinal=10)
importance(plot(iris.adaboost)

sub <- c(sample(1:150, 35), sample(1:100, 35), sample(101:150, 35))
iris.bagging <- bagging(Species ~ ., data=iris[, sub], mfinal=10)
iris.bagging <- bagging(Species ~ ., data=iris[, sub], mfinal=10)
iris.predbagging <- predict.bagging(iris.bagging, newdata=iris[, -sub])
iris.predbagging
#Predicting with unlabeled data
iris.predbagging <- predict.bagging(iris.bagging, newdata=iris[, -sub, -5])
iris.predbagging
```

Run

You should assume that any scripts or data that you put into this service are public.
Read more about your data's privacy and security here.

loading required package: rpart
loading required package: rpart
loading required package: rpart

Fig. 15.1 Live demo: Iris flowers classification using adabag

Table 15.3 Performance evaluation for several classification, prediction, and clustering methods

Model	Learning task	Method	Parameters
KNN	Classification	knn	k
Naïve Bayes	Classification	nb	fL, usekernel
Decision Trees	Classification	C5.0	model, trials, winnow
OneR Rule Learner	Classification	OneR	None
RIPPER Rule Learner	Classification	JRip	NumOpt
Linear Regression	Regression	lm	None
Regression Trees	Regression	rpart	cp
Model Trees	Regression	M5	pruned, smoothed, rules
Neural Networks	Dual use	nnet	size, decay
Support Vector Machines (Linear Kernel)	Dual use	svmLinear	C
Support Vector Machines (Radial Basis Kernel)	Dual use	svmRadial	C, sigma
Random Forests	Dual use	rf	mtry

15.3.1 ***Model Improvement Case Study***

From the course datasets, use the 05_PPMI_top_UPDRS_Integrated_LongFormat1.csv case-study data to perform a multi-class prediction.

Use ResearchGroup as response, which have “PD”, “Control” and “SWEDD” three classes.

- Delete ID column, impute missing value with mean or median and justify your choice.
- Normalize the covariates.
- Implement automated parameter tuning process and report the optimal accuracy and κ .
- Set arguments and rerun the tuning, trying different method and number settings.
- Train a random forest, tune the parameters, report the result and output cross table.
- Use bagging algorithm and report the accuracy and κ .
- Perform randomForest and report the accuracy and κ .
- Report the accuracy by AdaBoost and make sure to try all three methods.
- Finally, give a brief summary about all the model improvement approaches.
- Try the procedure on other data in the list of Case-Studies, e.g., Traumatic Brain Injury Study and the corresponding dataset.

References

- Zhu, J., Zou, H., Rosset, S., Hastie, T. (2009) *Multi-class AdaBoost*, Statistics and Its Interface, 2, 349–360.
- Breiman, L. (1998): *Arcing classifiers*, The Annals of Statistics, 26(3), 801–849.
- Freund, Y., Schapire, RE. (1996) *Experiments with a new boosting algorithm*, In Proceedings of the Thirteenth International Conference on Machine Learning, 148–156, Morgan Kaufmann

Chapter 16

Specialized Machine Learning Topics



This chapter presents some technical details about data formats, streaming, optimization of computation, and distributed deployment of optimized learning algorithms. Chapter 22 provides additional optimization details. We show format conversion and working with XML, SQL, JSON, 15 CSV, SAS and other data objects. In addition, we illustrate SQL server queries, describe protocols for managing, classifying and predicting outcomes from data streams, and demonstrate strategies for optimization, improvement of computational performance, parallel (MPI) and graphics (GPU) computing.

The Internet of Things (IoT) leads to a paradigm shift of scientific inference – from static data interrogated in a batch or distributed environment to on-demand service-based Cloud computing. Here, we will demonstrate how to work with specialized data, data-streams, and SQL databases, as well as develop and assess on-the-fly data modeling, classification, prediction and forecasting methods. Important examples to keep in mind throughout this chapter include high-frequency data delivered real time in hospital ICU's (e.g., microsecond Electroencephalography signals, EEGs), dynamically changing stock market data (e.g., Dow Jones Industrial Average Index, DJI), and weather patterns.

We will present (1) format conversion of XML, SQL, JSON, CSV, SAS and other data objects, (2) visualization of bioinformatics and network data, (3) protocols for managing, classifying and predicting outcomes from data streams, (4) strategies for optimization, improvement of computational performance, parallel (MPI) and graphics (GPU) computing, and (5) processing of very large datasets.

16.1 Working with Specialized Data and Databases

Unlike the case studies we saw in the previous chapters, some real world data may not always be nicely formatted, e.g., as CSV files. We must collect, arrange, wrangle, and harmonize scattered information to generate computable data objects that can be further processed by various techniques. Data wrangling and preprocessing may take

over 80% of the time researchers spend interrogating complex multi-source data archives. The following procedures will enhance your skills in collecting and handling heterogeneous real world data. Multiple examples of handling long-and-wide data, messy and tidy data, and data cleaning strategies can be found in this JSS Tidy Data article by Hadley Wickham.

16.1.1 Data Format Conversion

The R package `rio` imports and exports various types of file formats, e.g., tab-separated (`.tsv`), comma-separated (`.csv`), JSON (`.json`), Stata (`.dta`), SPSS (`.sav` and `.por`), Microsoft Excel (`.xls` and `.xlsx`), Weka (`.arff`), and SAS (`.sas7bdat` and `.xpt`).

`rio` provides three important functions `import()`, `export()` and `convert()`. They are intuitive, easy to understand, and efficient to execute. Take Stata (`.dta`) files as an example. First, we can download `02_Nof1_Data.dta` from our datasets folder.

```
# install.packages("rio")
library(rio)
# Download the SAS .DTA file first locally
# Local data can be loaded by:
#nof1<-import("02_Nof1_Data.dta")
# the data can also be loaded from the server remotely as well:
nof1<-read.csv("https://umich.instructure.com/files/330385/download?download_frd=1")
str(nof1)

## 'data.frame': 900 obs. of 10 variables:
## $ ID      : int 1 1 1 1 1 1 1 1 1 ...
## $ Day     : int 1 2 3 4 5 6 7 8 9 10 ...
## $ Tx      : int 1 1 0 0 1 1 0 0 1 1 ...
## $ SelfEff : int 33 33 33 33 33 33 33 33 33 33 ...
## $ SelfEff25: int 8 8 8 8 8 8 8 8 8 8 ...
## $ WPSS    : num 0.97 -0.17 0.81 -0.41 0.59 -1.16 0.3 -0.34 -0.74 -0.38 ...
...
## $ SocSuppt : num 5 3.87 4.84 3.62 4.62 2.87 4.33 3.69 3.29 3.66 ...
## $ PMss    : num 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 ...
## $ PMss3   : num 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 ...
## $ PhyAct   : int 53 73 23 36 21 0 21 0 73 114 ...
```

The data are automatically stored as a data frame. Note that `rio` sets `stringAsFactors=FALSE` as default.

`rio` can help us export files into any other format we choose. To do this we have to use the `export()` function.

```
#Sys.getenv("R_ZIPCMD", "zip") # Get the C Zip application
Sys.setenv(R_ZIPCMD="E:/Tools/ZIP/bin/zip.exe")
Sys.getenv("R_ZIPCMD", "zip")

## [1] "E:/Tools/ZIP/bin/zip.exe"
export(nof1, "02_Nof1.xlsx")
```

This line of code exports the *Nof1* data in `xlsx` format located in the R working directory. Mac users may have a problem exporting `*.xlsx` files using `rio` because of a lack of a zip tool, but still can output other formats such as `".csv"`. An alternative strategy to save an `xlsx` file is to use package `xlsx` with default `row.name=TRUE`.

`rio` also provides a one step process to convert and save data into alternative formats. The following simple code allows us to convert and save the `02_Nof1_Data.dta` file we just downloaded into a CSV file.

```
# convert("02_Nof1_Data.dta", "02_Nof1_Data.csv")
convert("02_Nof1.xlsx",
"02_Nof1_Data.csv")
```

You can see a new CSV file popup in the current working directory. Similar transformations are available for other data formats and types.

16.1.2 Querying Data in SQL Databases

Let's use as an example the CDC Behavioral Risk Factor Surveillance System (BRFSS) Data, 2013-2015. This file for the combined landline and cell phone data set was exported from SAS V9.3 in the XPT transport format. This file contains 330 variables and can be imported into SPSS or STATA. Please note: some of the variable labels get truncated in the process of converting to the XPT format.

Be careful – this compressed (ZIP) file is over 315MB in size!

```
# install.packages("Hmisc")
library(Hmisc)

memory.size(max=7)

## [1] 115.81
pathToZip <- tempfile()
download.file("http://www.socr.umich.edu/data/DSPA/BRFSS_2013_2014_2015.zip",
, pathToZip)
# let's just pull two of the 3 years of data (2013 and 2015)
brfss_2013 <- sasxport.get(unzip(pathToZip)[1])

## Processing SAS dataset LLCP2013 ..
brfss_2015 <- sasxport.get(unzip(pathToZip)[3])

## Processing SAS dataset LLCP2015 ..
dim(brfss_2013); object.size(brfss_2013)

## [1] 491773    336
## 685581232 bytes
```

```

# summary(brfss_2013[1:1000, 1:10]) # subsample the data

# report the summaries for
summary(brfss_2013$has_plan)

## Length Class Mode
##      0    NULL  NULL

brfss_2013$x.race <- as.factor(brfss_2013$x.race)
summary(brfss_2013$x.race)

##      1      2      3      4      5      6      7      8      9    NA's
## 376451 39151 7683 9510 1546 2693 9130 37054 8530      25

# clean up
unlink(pathToZip)

```

Let's try to use logistic regression to find out if self-reported race/ethnicity predicts the binary outcome of having a health care plan.

```

brfss_2013$has_plan <- brfss_2013$hlthpln1 == 1

system.time(
  gml1 <- glm(has_plan ~ as.factor(x.race), data=brfss_2013,
               family=binomial)
) # report execution time

##      user    system   elapsed
##  2.20     0.23    2.46

summary(gml1)

##
## Call:
## glm(formula = has_plan ~ as.factor(x.race), family = binomial,
##      data = brfss_2013)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.1862  0.4385  0.4385  0.4385  0.8047
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)           2.293549  0.005649 406.044 <2e-16 ***
## as.factor(x.race)2 -0.721676  0.014536 -49.647 <2e-16 ***
## as.factor(x.race)3 -0.511776  0.032974 -15.520 <2e-16 ***
## as.factor(x.race)4 -0.329489  0.031726 -10.386 <2e-16 ***
## as.factor(x.race)5 -1.119329  0.060153 -18.608 <2e-16 ***
## as.factor(x.race)6 -0.544458  0.054535 -9.984 <2e-16 ***
## as.factor(x.race)7 -0.510452  0.030346 -16.821 <2e-16 ***
## as.factor(x.race)8 -1.332005  0.012915 -103.138 <2e-16 ***
## as.factor(x.race)9 -0.582204  0.030604 -19.024 <2e-16 ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##      Null deviance: 353371  on 491747  degrees of freedom
## Residual deviance: 342497  on 491739  degrees of freedom
## (25 observations deleted due to missingness)
## AIC: 342515
##
## Number of Fisher Scoring iterations: 5
```

Next, we'll examine the odds (rather the log odds ratio, LOR) of having a health care plan (HCP) by race (R). The LORs are calculated for two array dimensions, separately for each *race* level (presence of *health care plan* (HCP) is binary, whereas *race* (R) has 9 levels, *R1*, *R2*, ..., *R9*). For example, the odds ratio of having a HCP for *R1* : *R2* is:

$$OR(R1 : R2) = \frac{\frac{P(HCP|R1)}{1-P(HCP|R1)}}{\frac{P(HCP|R2)}{1-P(HCP|R2)}}.$$

```
#load the vcd package to compute the LOR
library("vcd")

## Loading required package: grid

Lor_HCP_by_R <- loddsratio(has_plan ~ as.factor(x.race), data = brfss_2013)
Lor_HCP_by_R

## Log odds ratios for has_plan and as.factor(x.race)
##
##      1:2          2:3          3:4          4:5          5:6          6:7
## -0.72167619  0.20990061  0.18228646 -0.78984000  0.57487142  0.03400611
##      7:8          8:9
## -0.82155382  0.74980101
```

Now, let's see an example of querying a database containing structured relational collection of data records. A *query* is a machine instruction (typically represented as text) sent by a user to a remote database requesting a specific database operation (e.g., search or summary). One database communication protocol relies on SQL (Structured query language). MySQL is an instance of a database management system that supports SQL communication and is utilized by many web applications, e.g., YouTube, Flickr, Wikipedia, biological databases like GO, ensembl, etc. Below is an example of an SQL query using the package RMySQL. An alternative way to interface an SQL database is using the package RODBC.

```
# install.packages("DBI"); install.packages("RMySQL")
# install.packages("RODBC"); library(RODBC)
library(DBI)
library(RMySQL)

ucscGenomeConn <- dbConnect(MySQL(),
  user='genome',
  dbname='hg38',
  host='genome-mysql.cse.ucsc.edu')
```

```

result <- dbGetQuery(ucscGenomeConn, "show databases;");

# List the DB tables
allTables <- dbListTables(ucscGenomeConn); length(allTables)

# Get dimensions of a table, read and report the head
dbListFields(ucscGenomeConn, "affyU133Plus2")
affyData <- dbReadTable(ucscGenomeConn, "affyU133Plus2"); head(affyData)

# Select a subset, fetch the data, and report the quantiles
subsetQuery <- dbSendQuery(ucscGenomeConn, "select * from affyU133Plus2
where misMatches between 1 and 3")
affySmall <- fetch(subsetQuery); quantile(affySmall$misMatches)

# Get repeat mask
bedFile <- 'repUCSC.bed'
df <- dbSendQuery(ucscGenomeConn, 'select genoName,genoStart,genoEnd,
repName,swScore, strand,repClass, repFamily from rmsk') %>%
  dbFetch(n=-1) %>%
  mutate(genoName = str_replace(genoName, 'chr', '')) %>%
 tbl_df %>%
  write_tsv(bedFile, col_names=F)
message('written ', bedFile)

# Once done, close the connection
dbDisconnect(ucscGenomeConn)

```

To complete the above database SQL commands, it requires access to the remote UCSC SQL Genome server and user-specific credentials. You can see this functional example on the DSPA website. Below is another example that can be done by all readers, as it relies only on local services.

```

# install.packages("RSQLite")
library("RSQLite")

# generate an empty DB and store it in RAM
myConnection <- dbConnect(RSQLite::SQLite(), ":memory:")
myConnection

## <SQLiteConnection>
##   Path: :memory:
##   Extensions: TRUE

dbListTables(myConnection)

## character(0)

# Add tables to the local SQL DB
data(USArrests); dbWriteTable(myConnection, "USArrests", USArrests)

## [1] TRUE

dbWriteTable(myConnection, "brfss_2013", brfss_2013)

## [1] TRUE

dbWriteTable(myConnection, "brfss_2015", brfss_2015)

## [1] TRUE

```

```
# Check again the DB content
dbListFields(myConnection, "brfss_2013")

## [1] "x.state"    "fmonth"     "idate"      "imonth"     "iday"
## [6] "iyear"       "dispcode"    "seqno"      "x.psu"      "ctelenum"
## [11] "pvtresd1"   "colghous"   "stateres"   "cellfon3"   "ladult"
## [16] "numadult"   "nummen"     "numwomen"   "genhlth"    "physhlth"
## [21] "menthlth"   "poorhlth"   "hlthpln1"   "persdoc2"   "medcost"
...
## [331] "rcsbrac1"   "rcsrace1"   "rchisla1"   "rcsbirth"   "typeinds"
## [336] "typework"   "has_plan"

dbListTables(myConnection);

## [1] "USAArrests" "brfss_2013" "brfss_2015"

# Retrieve the entire DB table (for the smaller USAArrests table)
dbGetQuery(myConnection, "SELECT * FROM USAArrests")

##   Murder Assault UrbanPop Rape
## 1    13.2     236      58 21.2
## 2    10.0     263      48 44.5
## 3     8.1     294      80 31.0
## 4     8.8     190      50 19.5
## 5     9.0     276      91 40.6
## 6     7.9     204      78 38.7
## 7     3.3     110      77 11.1
## 8     5.9     238      72 15.8
## 9    15.4     335      80 31.9
## 10    17.4     211      60 25.8
## 11    5.3      46       83 20.2
## 12    2.6     120      54 14.2
## 13   10.4     249      83 24.0
## 14    7.2     113      65 21.0
## 15    2.2      56       57 11.3
## 16    6.0     115      66 18.0
## 17    9.7     109      52 16.3
## 18   15.4     249      66 22.2
## 19    2.1      83       51  7.8
## 20   11.3     300      67 27.8
## 21    4.4     149      85 16.3
## 22   12.1     255      74 35.1
## 23    2.7      72       66 14.9
## 24   16.1     259      44 17.1
## 25    9.0     178      70 28.2
## 26    6.0     109      53 16.4
## 27    4.3     102      62 16.5
## 28   12.2     252      81 46.0
## 29    2.1      57       56  9.5
## 30    7.4     159      89 18.8
## 31   11.4     285      70 32.1
## 32   11.1     254      86 26.1
## 33   13.0     337      45 16.1
## 34    0.8      45       44  7.3
## 35    7.3     120      75 21.4
## 36    6.6     151      68 20.0
## 37    4.9     159      67 29.3
## 38    6.3     106      72 14.9
```

```
## 39    3.4    174    87  8.3
## 40   14.4    279    48 22.5
## 41    3.8     86    45 12.8
## 42   13.2   188    59 26.9
## 43   12.7   201    80 25.5
## 44    3.2   120    80 22.9
## 45    2.2     48    32 11.2
## 46    8.5   156    63 20.7
## 47    4.0   145    73 26.2
## 48    5.7     81    39  9.3
## 49    2.6     53    66 10.8
## 50    6.8   161    60 15.6

# Retrieve just the average of one feature
myQuery <- dbGetQuery(myConnection, "SELECT avg(Assault) FROM USArrests"); myQuery

##   avg(Assault)
## 1 170.76

myQuery <- dbGetQuery(myConnection, "SELECT avg(Assault) FROM USArrests GROUP BY UrbanPop"); myQuery
##   avg(Assault)
## 1      48.00
## 2      81.00
## 3     152.00
## 4     211.50
## 5     271.00
## 6     190.00
## 7      83.00
## 8     109.00
## 9     109.00
## 10    120.00
## 11    57.00
## 12    56.00
## 13    236.00
## 14    188.00
## 15    186.00
## 16    102.00
## 17    156.00
## 18    113.00
## 19    122.25
## 20    229.50
## 21    151.00
## 22    231.50
## 23    172.00
## 24    145.00
## 25    255.00
## 26    120.00
## 27    110.00
## 28    204.00
## 29    237.50
## 30    252.00
## 31    147.50
## 32    149.00
## 33    254.00
## 34    174.00
## 35    159.00
## 36    276.00
```

```

# Or do it in batches (for the much larger brfss_2013 and brfss_2015 tables)
myQuery <- dbGetQuery(myConnection, "SELECT * FROM brfss_2013")

# extract data in chunks of 2 rows, note: dbGetQuery vs. dbSendQuery
# myQuery <- dbSendQuery(myConnection, "SELECT * FROM brfss_2013")
# fetch2 <- dbFetch(myQuery, n = 2); fetch2
# do we have other cases in the DB remaining?
# extract all remaining data
# fetchRemaining <- dbFetch(myQuery, n = -1); fetchRemaining
# We should have all data in DB now
# dbHasCompleted(myQuery)
# compute the average (poorhlth) grouping by Insurance (hlthpln1)
# Try some alternatives: numadult nummen numwomen genhlth physhlth menthlth
poorhlth hlthpln1
myQuery1_13 <- dbGetQuery(myConnection, "SELECT avg(poorhlth) FROM brfss_201
3 GROUP BY hlthpln1"); myQuery1_13

##   avg(poorhlth)
## 1      56.25466
## 2      53.99962
## 3      58.85072
## 4      66.26757

# Compare 2013 vs. 2015: Health grouping by Insurance
myQuery1_15 <- dbGetQuery(myConnection, "SELECT avg(poorhlth) FROM brfss_201
5 GROUP BY hlthpln1"); myQuery1_15

##   avg(poorhlth)
## 1      55.75539
## 2      55.49487
## 3      61.35445
## 4      67.62125

myQuery1_13 - myQuery1_15

##   avg(poorhlth)
## 1      0.4992652
## 2     -1.4952515
## 3     -2.5037326
## 4     -1.3536797

# reset the DB query
# dbClearResult(myQuery)

# clean up
dbDisconnect(myConnection)

## [1] TRUE

```

16.1.3 Real Random Number Generation

We are already familiar with (pseudo) random number generation (e.g., `rnorm(100, 10, 4)` or `runif(100, 10, 20)`), which generate *algorithmically* computer values subject to specified distributions. There are also web services, e.g., random.org, that can provide *true random* numbers based on atmospheric

noise, rather than using a pseudo random number generation protocol. Below is one example of generating a total of 300 numbers arranged in 3 columns, each of 100 rows of random integers (in decimal format) between 100 and 200.

```
#https://www.random.org/integers/?num=300&min=100&max=200&col=3&base=10&
format=plain&rnd=new
siteURL <- "http://random.org/integers/" # base URL
shortQuery<- "num=300&min=100&max=200&col=3&base=10&format=plain&rnd=new"
completeQuery <- paste(siteURL, shortQuery, sep="?") # concat url and
submit query string
rngNumbers <- read.table(file=completeQuery) # and read the data
rngNumbers

##      V1   V2   V3
## 1 144 179 131
## 2 127 160 150
## 3 142 169 109
...
## 98 178 103 134
## 99 173 178 156
## 100 117 118 110
```

16.1.4 Downloading the Complete Text of Web Pages

RCurl package provides an amazing tool for extracting and scraping information from websites. Let's install it and extract information from a SOCR website.

```
# install.packages("RCurl")
library(RCurl)

## Loading required package: bitops

web<-getURL("http://wiki.socr.umich.edu/index.php/SOCR_Data", followlocation
= TRUE)
str(web, nchar.max = 200)

## chr "<!DOCTYPE html>\n<html Lang=\"en\" dir=\"ltr\" class=\"client-nojs\">\n<head>\n<meta charset=\"UTF-8\" />\n<title>SOCR Data - SOCR</title>\n<meta http-equiv=\"X-UA-Compatible\" content=\"IE=EDGE\" />"/ __truncated__"
```

The web object looks incomprehensible. This is because most websites are wrapped in XML/HTML hypertext or include JSON formatted metadata. RCurl deals with special HTML tags and website metadata.

To deal with the web pages only, httr package would be a better choice than RCurl. It returns a list that makes much more sense.

```
#install.packages("httr")
library(httr)
web<-GET("http://wiki.socr.umich.edu/index.php/SOCR_Data")
str(web[1:3])

## List of 3
## $ url      : chr "http://wiki.socr.umich.edu/index.php/SOCR_Data"
## $ status_code: int 200
```

```

## $ headers      :List of 12
##   ..$ date           : chr "Mon, 03 Jul 2017 19:09:56 GMT"
##   ..$ server          : chr "Apache/2.2.15 (Red Hat)"
##   ..$ x-powered-by    : chr "PHP/5.3.3"
##   ..$ x-content-type-options: chr "nosniff"
##   ..$ content-Language : chr "en"
##   ..$ vary             : chr "Accept-Encoding, Cookie"
##   ..$ expires           : chr "Thu, 01 Jan 1970 00:00:00 GMT"
##   ..$ cache-control     : chr "private, must-revalidate, max-age=0"
##   ..$ last-modified     : chr "Sat, 22 Oct 2016 21:46:21 GMT"
##   ..$ connection        : chr "close"
##   ..$ transfer-encoding : chr "chunked"
##   ..$ content-type       : chr "text/html; charset=UTF-8"
##   ... attr(*, "class")= chr [1:2] "insensitive" "List"

```

16.1.5 Reading and Writing XML with the XML Package

A combination of the `RCurl` and the `XML` packages could help us extract only the plain text in our desired webpages. This would be very helpful to get information from heavy text-based websites.

```

web<-getURL("http://wiki.socr.umich.edu/index.php/SOCR_Data", followLocation
= TRUE)
#install.packages("XML")
library(XML)
web.parsed<-htmlParse(web, asText = T)
plain.text<-xpathSApply(web.parsed, "//p", xmlValue)
cat(paste(plain.text, collapse = "\n"))

## The links below contain a number of datasets that may be used for demonst
## ration purposes in probability and statistics education. There are two types
## of data - simulated (computer-generated using random sampling) and observed
## (research, observationally or experimentally acquired).
##
## The SOCR resources provide a number of mechanisms to simulate data using
## computer random-number generators. Here are some of the most commonly used S
## OCR generators of simulated data:
##
## The following collections include a number of real observed datasets from
## different disciplines, acquired using different techniques and applicable in
## different situations.
##
## In addition to human interactions with the SOCR Data, we provide several
## machine interfaces to consume and process these data.
##
## Translate this page:
##
## (default)
##
## Deutsch
...
## România
##
## Sverige

```

Here we extracted all plain text between the starting and ending *paragraph* HTML tags, `<p>` and `</p>`.

More information about extracting text from XML/HTML to text via XPath is available online.

16.1.6 Web-Page Data Scraping

The process that extracting data from complete web pages and storing it in structured data format is called **scraping**. However, before starting a data scrape from a website, we need to understand the underlying HTML structure for that specific website. Also, we have to check the terms of that website to make sure that scraping from this site is allowed.

The R package `rvest` is a very good place to start “harvesting” data from websites.

To start with, we use `read_html()` to store the SOCR data website into a `xmlnode` object.

```
library(rvest)
SOCR<-read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data")
SOCR

## {xml_document}
## <html lang="en" dir="ltr" class="client-nojs">
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=
...
## [2] <body class="mediawiki ltr sitedir-ltr ns-0 ns-subject page-SOCR_Dat
...
```

From the summary structure of SOCR, we can discover that there are two important hypertext section markups `<head>` and `<body>`. Also, notice that the SOCR data website uses `<title>` and `</title>` tags to separate title in the `<head>` section. Let’s use `html_node()` to extract title information based on this knowledge.

```
SOCR %>% html_node("head title") %>% html_text()
## [1] "SOCR Data - SOCR"
```

Here we used `%>%` operator, or pipe, to connect two functions. The above line of code creates a chain of functions to operate on the `SOCR` object. The first function in the chain `html_node()` extracts the `title` from `head` section. Then, `html_text()` translates HTML formatted hypertext into English. More on R piping can be found in the `magrittr` package.

Another function, `rvest::html_nodes()` can be very helpful in scraping. Similar to `html_node()`, `html_nodes()` can help us extract multiple nodes in an `xmlnode` object. Assume that we want to obtain the meta elements (usually page

description, keywords, author of the document, last modified, and other metadata from the SOCR data website. We apply `html_nodes()` to the `SOCR` object to extract the hypertext data, e.g., lines starting with `<meta>` in the `<head>` section of the HTML page source. It is optional to use `html_attrs()`, which extracts attributes, text and tag names from HTML, obtain the main text attributes.

```
meta<-SOCR %>% html_nodes("head meta") %>% html_attrs()
meta

## [[1]]
##           http-equiv           content
## "Content-Type" "text/html; charset=UTF-8"
##
## [[2]]
## charset
## "UTF-8"
##
## [[3]]
##           http-equiv           content
## "X-UA-Compatible" "IE=EDGE"
##
## [[4]]
##           name           content
## "generator" "MediaWiki 1.23.1"
##
## [[5]]
##           name           content
## "ResourceLoaderDynamicStyles"     ""
```

16.1.7 Parsing JSON from Web APIs

Application Programming Interfaces (APIs) allow web-accessible functions to communicate with each other. Today most API is stored in JSON (JavaScript Object Notation) format.

JSON represents a plain text format used for web applications, data structures or objects. Online JSON objects could be retrieved by packages like `RCurl` and `httr`. Let's see a JSON formatted dataset first. We can use `02_Nof1_Data.json` in the class file as an example.

```
library(httr)
nof1<-GET("https://umich.instructure.com/files/1760327/download?download_frd
=1")
nof1

## Response [https://instructure-uploads.s3.amazonaws.com/account_1770000000
0000001/attachments/1760327/02_Nof1_Data.json?response-content-disposition=a
ttachment%3B%20filename%3D%2202_Nof1_Data.json%22%3B%20filename%2A%3DUTF-8%2
%27202%255Fnof1%255FData.json&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credit
ential=AKIAJFNFNXH2V207RPCAA%2F20170703%2Fus-east-1%2F53%2Faws4_request&X-Amz-Da
te=20170703T190959Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host&X-Amz-Signa
ture=ceb3be3e71d9c370239bab558fc0191bc829b98a7ba61ac86e27a2fc3c1e8ce]
```

```
##  Date: 2017-07-03 19:10
##  Status: 200
##  Content-Type: application/json
##  Size: 109 kB
## [{"ID":1,"Day":1,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":2,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":3,"Tx":0,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":4,"Tx":0,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":5,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":6,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":7,"Tx":0,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":8,"Tx":0,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":9,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## {"ID":1,"Day":10,"Tx":1,"SelEff":33,"SelEff25":8,"WPSS":...}
## ...
```

We can see that JSON objects are very simple. The data structure is organized using hierarchies marked by square brackets. Each piece of information is formatted as a `{key:value}` pair.

The package `jsonlite` is a very useful tool to import online JSON formatted datasets into data frame directly. Its syntax is very straight-forward.

```
#install.packages("jsonlite")
library(jsonlite)
nof1_Lite<-
fromJSON("https://umich.instructure.com/files/1760327/downLoad?downLoad_frd=1")
class(nof1_Lite)
## [1] "data.frame"
```

16.1.8 Reading and Writing Microsoft Excel Spreadsheets Using XLSX

We can transfer a `xlsx` dataset into CSV and use `read.csv()` to load this kind of dataset. However, R provides an alternative `read.xlsx()` function in package `xlsx` to simplify this process. Take our `02_Nof1_Data.xls` data in the class file as an example. We need to download the file first.

```
# install.packages("xlsx")
library(xlsx)
nof1<-read.xlsx("C:/Users/Folder/02_Nof1.xlsx", 1)
str(nof1)

## 'data.frame': 900 obs. of 10 variables:
## $ ID      : num  1 1 1 1 1 1 1 1 ...
## $ Day     : num  1 2 3 4 5 6 7 8 9 10 ...
## $ TX      : num  1 1 0 0 1 1 0 0 1 1 ...
```

```
## $ SelfEff : num 33 33 33 33 33 33 33 33 33 33 ...
## $ SelfEff25: num 8 8 8 8 8 8 8 8 ...
## $ WPSS : num 0.97 -0.17 0.81 -0.41 0.59 -1.16 0.3 -0.34 -0.74 -0.38 ...
...
## $ SocSuppt : num 5 3.87 4.84 3.62 4.62 2.87 4.33 3.69 3.29 3.66 ...
## $ PMss : num 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 4.03 ...
## $ PMss3 : num 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 ...
## $ PhyAct : num 53 73 23 36 21 0 21 0 73 114 ...
```

The last argument, 1, stands for the first excel sheet, as any excel file may include a large number of tables in it. Also, we can download the `xls` or `xlsx` file into our R working directory so that it is easier to find the file path.

Sometimes more complex protocols may be necessary to ingest data from XLSX documents. For instance, if the XLSX doc is large, includes many tables and is only accessible via HTTP protocol from a web-server. Below is an example of downloading the second table, `ABIDE_Aggregated_Data`, from the multi-table Autism/ABIDE XLSX dataset:

```
# install.packages("openxlsx"); library(openxlsx)
tmp = tempfile(fileext = ".xlsx")
download.file(url = "https://umich.instructure.com/files/3225493/download?do
wnload_frd=1",
destfile = tmp, mode="wb") df_Autism <- openxlsx::read.xlsx(xlsxFile = tmp,
sheet = "ABIDE_Aggregated_Data", skipEmptyRows = TRUE)
dim(df_Autism)
## [1] 1098 2145
```

16.2 Working with Domain-Specific Data

Powerful machine-learning methods have already been applied in many applications. Some of these techniques are very specialized and some applications require unique approaches to address the corresponding challenges.

16.2.1 Working with Bioinformatics Data

Genetic data are stored in widely varying formats and usually have more feature variables than observations. They could have 1,000 columns and only 200 rows. One of the commonly used pre-processing steps for such datasets is *variable selection*. We will talk about this in Chap. 17.

The Bioconductor project created powerful R functionality (packages and tools) for analyzing genomic data, see Bioconductor for more detailed information.

16.2.2 Visualizing Network Data

Social network data and graph datasets describe the relations between nodes (vertices) using connections (links or edges) joining the node objects. Assume we have N objects, we can have $N * (N - 1)$ directed links establishing paired associations between the nodes. Let's use an example with $N=4$ to demonstrate a simple graph potentially modeling the node linkage Table 16.1.

If we change the $a \rightarrow b$ to an indicator variable (0 or 1) capturing whether we have an edge connecting a pair of nodes, then we get the graph *adjacency matrix*.

Edge lists provide an alternative way to represent network connections. Every line in the list contains a connection between two nodes (objects) (Table 16.2).

The edge list on Table 16.2 lists three network connections: object 1 is linked to object 2; object 1 is linked to object 3; and object 2 is linked to object 3. Note that edge lists can represent both *directed* as well as *undirected* networks or graphs.

We can imagine that if N is very large, e.g., social networks, the data representation and analysis may be resource intense (memory or computation). In R, we have multiple packages that can deal with social network data. One user-friendly example is provided using the `igraph` package. First, let's build a toy example and visualize it using this package (Fig. 16.1).

```
#install.packages("igraph")
library(igraph)

g<-graph(c(1, 2, 1, 3, 2, 3, 3, 4), n=10)
plot(g)
```

Here `c(1, 2, 1, 3, 2, 3, 3, 4)` is an edge list with 4 rows and `n=10` indicates that we have 10 nodes (objects) in total. The small arrows in the graph show the directed network connections. We might notice that 5-10 nodes are scattered out in the graph. This is because they are not included in the edge list, so there are no network connections between them and the rest of the network.

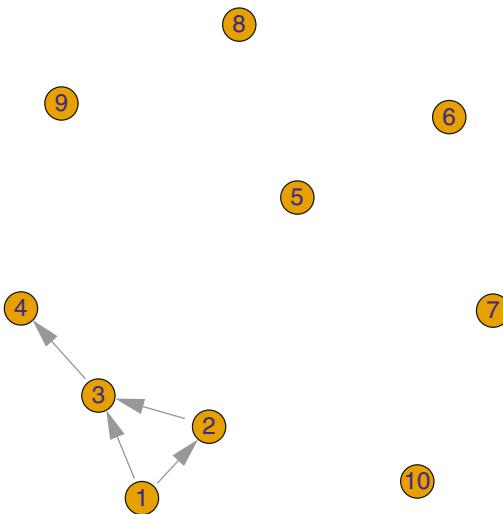
Table 16.1 Schematic matrix representation of network connectivity

Objects	1	2	3	4
1	$1 \rightarrow 2$	$1 \rightarrow 3$	$1 \rightarrow 4$
2	$2 \rightarrow 1$	$2 \rightarrow 3$	$2 \rightarrow 4$
3	$3 \rightarrow 1$	$3 \rightarrow 2$	$3 \rightarrow 4$
4	$4 \rightarrow 1$	$4 \rightarrow 2$	$4 \rightarrow 3$

Table 16.2 List-based representation of network connectivity

Vertex	Vertex
1	2
1	3
2	3

Fig. 16.1 A simple example of a social network as a graph object



Now let's examine the co-appearance network of Facebook circles. The data contains anonymized `circles` (friends lists) from Facebook collected from survey participants using a Facebook app. The dataset only includes edges (circles, 88,234) connecting pairs of nodes (users, 4,039) in the member social networks.

The values on the connections represent the number of links/edges within a circle. We have a huge edge-list made of scrambled Facebook user IDs. Let's load this dataset into R first. The data is stored in a text file. Unlike CSV files, text files in table format need to be imported using `read.table()`. We are using the `header=F` option to let R know that we don't have a header in the text file that contains only tab-separated node pairs (indicating the social connections, edges, between Facebook users).

```

soc.net.data<-read.table("https://umich.instructure.com/files/2854431/download?download_frd=1", sep=" ", header=F)
head(soc.net.data)

##   V1 V2
## 1  0  1
## 2  0  2
## 3  0  3
## 4  0  4
## 5  0  5
## 6  0  6
  
```

Now the data is stored in a data frame. To make this dataset ready for `igraph` processing and visualization, we need to convert `soc.net.data` into a matrix object.

```

soc.net.data.mat <- as.matrix(soc.net.data, ncol=2)
  
```

By using `ncol=2`, we made a matrix with two columns. The data is now ready and we can apply `graph.edgelist()`.

```
# remove the first 347 edges (to wipe out the degenerate "0" node)
graph_m<-graph.edgelist(soc.net.data.mat[-c(0:347), ], directed = F)
```

Before we display the social network graph we may want to examine our model first.

```
summary(graph_m)
## IGRAPH U--- 4038 87887 --
```

This is an extremely brief yet informative summary. The first line `U--- 4038 87887` includes potentially four letters and two numbers. The first letter could be `U` or `D` indicating undirected or directed edges. A second letter `N` would mean that the objects set has a “name” attribute. A third letter is for weighted (`W`) graph. Since we didn’t add weight in our analysis the third letter is empty (“`-`”). A fourth character is an indicator for bipartite graphs, whose vertices can be divided into two disjoint sets where each vertex from one set connects to one vertex in the other set. The two numbers following the 4 letters represent the number of nodes and the number of edges, respectively. Now let’s render the graph (Fig. 16.2).

```
plot(graph_m)
```

This graph is very complicated. We can still see that some words are surrounded by more nodes than others. To obtain such information we can use the `degree()` function, which lists the number of edges for each node.

```
degree(graph_m)
```

Skimming the table we can find that the 107-th user has as many as 1,044 connections, which makes the user a *highly-connected hub*. Likely, this node may have higher social relevance.

Some edges might be more important than other edges because they serve as a bridge to link a cloud of nodes. To compare their importance, we can use the betweenness centrality measurement. *Betweenness centrality* measures centrality in a network. High centrality for a specific node indicates influence. `betweenness()` can help us to calculate this measurement.

```
betweenness(graph_m)
```

Again, the 107-th node has the highest betweenness centrality ($3.556221e + 06$).

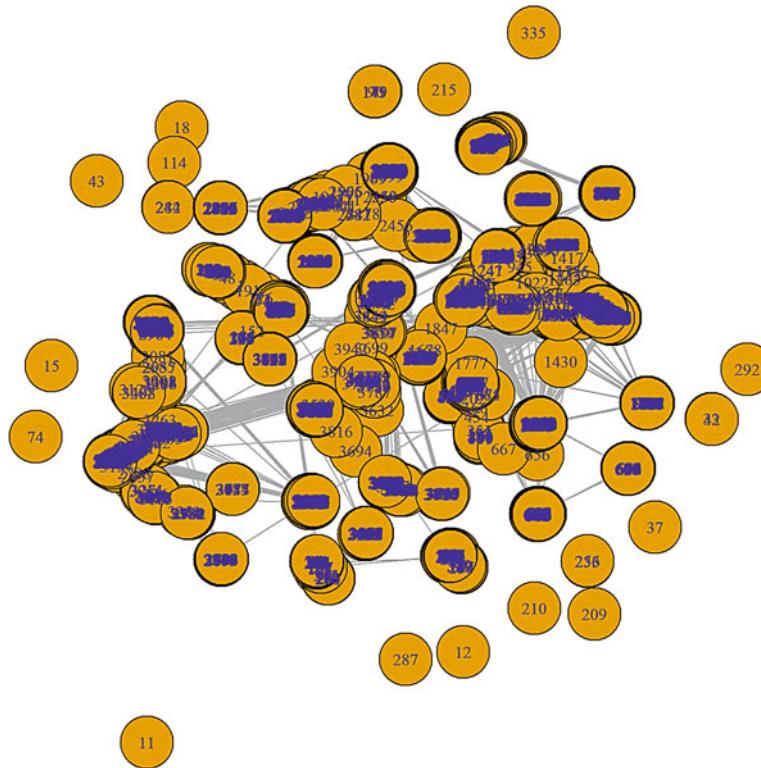


Fig. 16.2 Social network connectivity of Facebook users

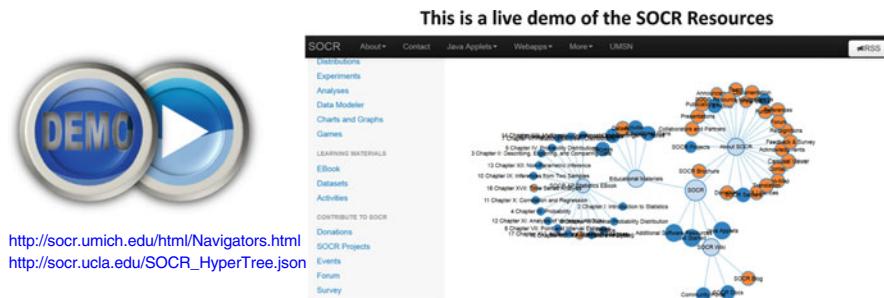


Fig. 16.3 Live demo: a dynamic graph representation of the SOCR resources

We can try another example using SOCR hierarchical data, which is also available for dynamic exploration as a tree graph. Let's read its JSON data source using the `jsonlite` package (Fig. 16.3).

```
tree.json<-fromJSON("http://socr.ucla.edu/SOCR_HyperTree.json",
simplifyDataFrame = FALSE)
```

This generates a `list` object representing the hierarchical structure of the network. Note that this is quite different from an edge list. There is one root node, its sub nodes are called *children nodes*, and the terminal nodes are call *leaf nodes*. Instead of presenting the relationship between nodes in pairs, this hierarchical structure captures the level for each node. To draw the social network graph, we need to convert it as a `Node` object. We can utilize the `as.Node()` function in the `data.tree` package to do so.

```
# install.packages("data.tree")
library(data.tree)
tree.graph<-as.Node(tree.json, mode = "explicit")
```

Here we use `mode="explicit"` option to allow “children” nodes to have their own “children” nodes. Now, the `tree.json` object has been separated into four different node structures – “About SOCR”, “SOCR Resources”, “Get Started”, and “SOCR Wiki”. Let’s plot the first one using `igraph` package (Fig. 16.4).

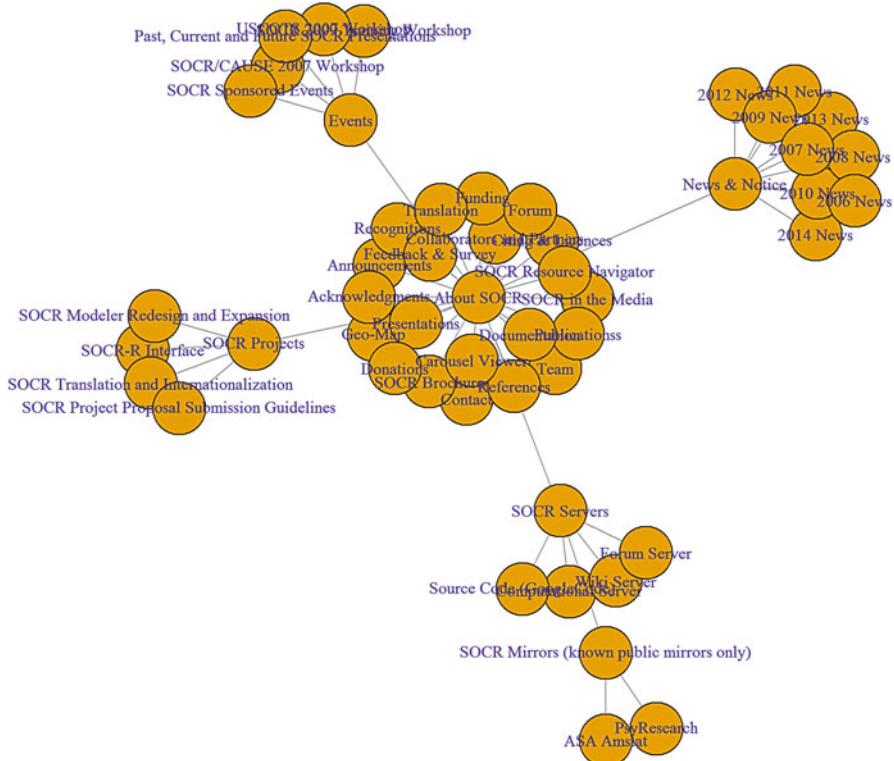


Fig. 16.4 The SOCR resourceome network plotted as a static R graph

```
plot(as.igraph(tree.graph$`About SOCR`), edge.arrow.size=5, edge.label.font=0.05)
```

In this graph, "About SOCR", which is located at the center, represents the root node of the tree graph.

16.3 Data Streaming

The proliferation of Cloud services and the emergence of modern technology in all aspects of human experiences leads to a tsunami of data much of which is streamed real-time. The interrogation of such voluminous data is an increasingly important area of research. *Data streams* are ordered, often unbounded sequences of data points created continuously by a data generator. All of the data mining, interrogation and forecasting methods we discuss here are also applicable to data streams.

16.3.1 Definition

Mathematically, a *data stream* is an ordered sequence of data points

$$Y = \{y_1, y_2, y_3, \dots, y_t, \dots\},$$

where the (time) index, t , reflects the order of the observation/record, which may be single numbers, simple vectors in multidimensional space, or objects, e.g., structured Ann Arbor Weather (JSON) and its corresponding structured form. Some streaming data is *streamed* because it's too large to be downloaded shotgun style and some is *streamed* because it's continually generated and serviced. This presents the potential problem of dealing with data streams that may be unlimited.

Notes:

- *Data sources*: Real or synthetic stream data can be used. Random simulation streams may be created by `rstream`. Real stream data may be piped from financial data providers, the WHO, World Bank, NCAR and other sources.
- *Inference Techniques*: Many of the data interrogation techniques we have seen can be employed for dynamic stream data, e.g., `factas`, for PCA, `rEMM` and `birch` for clustering, etc. Clustering and classification methods capable of processing data streams have been developed, e.g., *Very Fast Decision Trees* (VFDT), *time window-based Online Information Network* (OLIN), *On-demand Classification*, and the *APRIORI* streaming algorithm.
- *Cloud distributed computing*: Hadoop2/HadoopStreaming, SPARK, Storm3/ RStorm provide an environments to expand batch/script-based R tools to the Cloud.

16.3.2 The `stream` Package

The R `stream` package provides data stream mining algorithms using `fpc`, `clue`, `cluster`, `clusterGeneration`, `MASS`, and `proxy` packages. In addition, the package `streamMOA` provides an `rJava` interface to the Java-based data stream clustering algorithms available in the *Massive Online Analysis* (MOA) framework for stream classification, regression and clustering.

If you need a deeper exposure to data streaming in R, we recommend you go over the `stream` vignettes.

16.3.3 Synthetic Example: Random Gaussian Stream

This example shows the creation and loading of a *mixture of 5 random 2D Gaussians*, centers at (x_coords, y_coords) with paired correlations ρ_{corr} , representing a simulated data stream.

Generate the stream:

```
# install.packages("stream")
library("stream")

x_coords <- c(0.2,0.3, 0.5, 0.8, 0.9)
y_coords <- c(0.8,0.3, 0.7, 0.1, 0.5)
p_weight <- c(0.1, 0.9, 0.5, 0.4, 0.3) # A vector of probabilities that determine the likelihood of generated a data point from a particular cluster
set.seed(12345)
stream_5G <- DSD_Gaussians(k = 5, d = 2, mu=cbind(x_coords, y_coords),
p=p_weight)
```

k-Means Clustering

We will now try k-means and density-based data stream clustering algorithm, D-Stream, where micro-clusters are formed by grid cells of size `gridsize` with density of a grid cell (C_m) is least 1.2 times the average cell density. The model is updated with the next 500 data points from the stream.

```
dstream <- DSC_DStream(gridsize = .1, Cm = 1.2)
update(dstream, stream_5G, n = 500)
```

First, let's run the k-means clustering with $k = 5$ clusters and plot the resulting micro- and macro-clusters (Fig. 16.5).

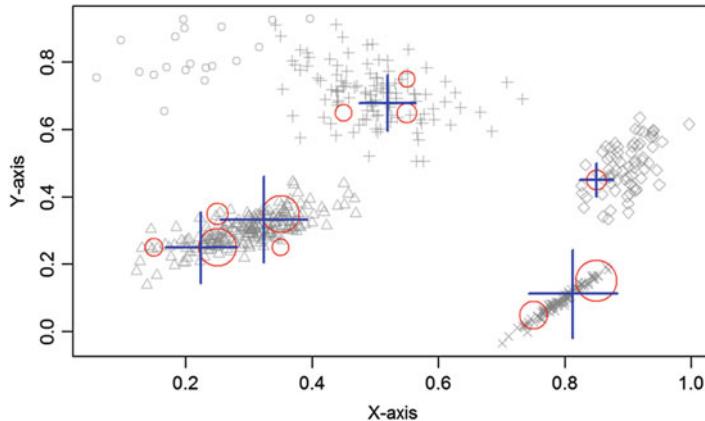


Fig. 16.5 Micro and macro clusters of a 5-means clustering of the first 500 points of the streamed simulated 2D Gaussian kernels

```
kmc <- DSC_Kmeans(k = 5)
recluster(kmc, dstream)
plot(kmc, stream_5G, type = "both", xLab="X-axis", yLab="Y-axis")
```

In this clustering plot, *micro-clusters are shown as circles and macro-clusters are shown as crosses* and their sizes represent the corresponding cluster weight estimates.

Next try the density-based data stream clustering algorithm D-Stream. Prior to updating the model with the next 1,000 data points from the stream, we specify the grid cells as micro-clusters, grid cell size (gridsize=0.1), and a micro-cluster (Cm=1.2) that specifies the density of a grid cell as a multiple of the average cell density.

```
dstream <- DSC_DStream(gridsize = 0.1, Cm = 1.2)
update(dstream, stream_5G, n=1000)
```

We can re-cluster the data using k-means with 5 clusters and plot the resulting *micro- and macro-clusters* (Fig. 16.6).

```
km_G5 <- DSC_Kmeans(k = 5)
recluster(km_G5, dstream)
plot(km_G5, stream_5G, type = "both")
```

Note the subtle changes in the clustering results between kmc and km_G5.

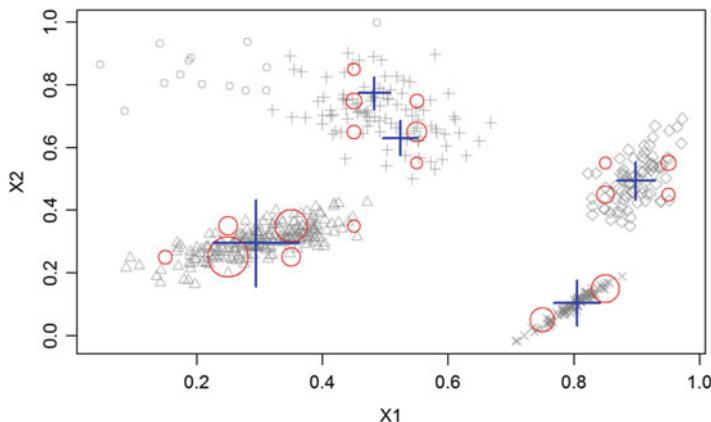


Fig. 16.6 Micro- and macro- clusters of a 5-means clustering of the next 1,000 points of the streamed simulated 2D Gaussian kernels

16.3.4 Sources of Data Streams

- Static Structure Streams**
- *DSD_BarsAndGaussians* generates two uniformly filled rectangular and two Gaussian clusters with different density.
 - *DSD_Gaussians* generates randomly placed static clusters with random multi-variate Gaussian distributions.
 - *DSD_mlbenchData* provides streaming access to machine learning benchmark data sets found in the *mlbench* package.
 - *DSD_mlbenchGenerator* interfaces the generators for artificial data sets defined in the *mlbench* package.
 - *DSD_Target* generates a ball in circle data set.
 - *DSD_UniformNoise* generates uniform noise in a d-dimensional (hyper) cube.

Concept Drift Streams

- *DSD_Benchmark* provides a collection of simple benchmark problems including splitting and joining clusters, and changes in density or size, which can be used as a comprehensive benchmark set for algorithm comparison.
- *DSD_MG* is a generator to specify complex data streams with concept drift. The shape as well as the behavior of each cluster over time can be specified using keyframes.
- *DSD_RandomRBFGeneratorEvents* generates streams using radial base functions with noise. Clusters move, merge and split.

Real Data Streams

- *DSD_Memory* provides a streaming interface to static, matrix-like data (e.g., a data frame, a matrix) in memory which represents a fixed portion of a data stream. Matrix-like objects also include large objects potentially stored on disk like `ff::ffdf`.
- *DSD_ReadCSV* reads data line by line in text format from a file or an open connection and makes it available in a streaming fashion. This way data that is larger than the available main memory can be processed.
- *DSD_ReadDB* provides an interface to an open result set from a SQL query to a relational database.

16.3.5 Printing, Plotting and Saving Streams

For DSD objects, some basic stream functions include `print()`, `plot()`, and `write_stream()`. These can save part of a data stream to disk. *DSD_Memory* and *DSD_ReadCSV* objects also include member functions like `reset_stream()` to reset the position in the stream to its beginning.

To request a new batch of data points from the stream we use `get_points()`. This chooses a *random cluster* (based on the probability weights in `p_weight`) and a point is drawn from the multivariate Gaussian distribution ($mean = \mu$, $covariance\ matrix = \Sigma$) of that cluster. Below, we pull $n = 10$ new data points from the stream (Fig. 16.7).

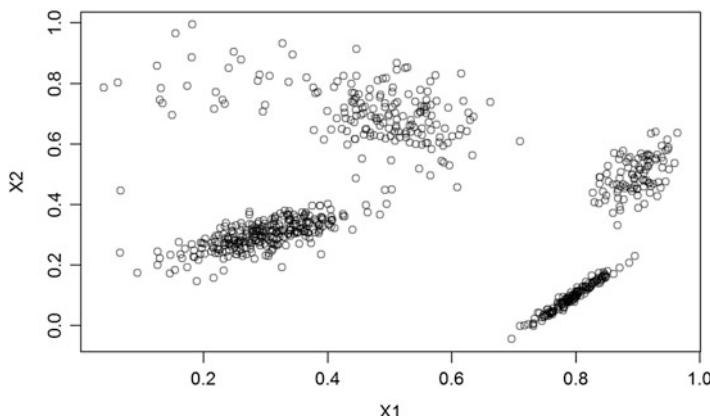


Fig. 16.7 Scatterplot of the next batch of 700 random Gaussian points in 2D

```

new_p  <-  get_points(stream_5G,  n  =  10)
new_p

##          X1          X2
## 1  0.4017803 0.2999017
## 2  0.4606262 0.5797737
## 3  0.4611642 0.6617809
## 4  0.3369141 0.2840991
## 5  0.8928082 0.5687830
## 6  0.8706420 0.4282589
## 7  0.2539396 0.2783683
## 8  0.5594320 0.7019670
## 9  0.5030676 0.7560124
## 10 0.7930719 0.0937701

new_p  <-  get_points(stream_5G,  n  =  100,  class  =  TRUE)
head(new_p,  n  =  20)

##          X1          X2  class
## 1  0.7915730 0.09533001      4
## 2  0.4305147 0.36953997      2
## 3  0.4914093 0.82120395      3
## 4  0.7837102 0.06771246      4
## 5  0.9233074 0.48164544      5
## 6  0.8606862 0.49399269      5
## 7  0.3191884 0.27607324      2
## 8  0.2528981 0.27596700      2
## 9  0.6627604 0.68988585      3
## 10 0.7902887 0.09402659      4
## 11 0.7926677 0.09030248      4
## 12 0.9393515 0.50259344      5
## 13 0.9333770 0.62817482      5
## 14 0.7906710 0.10125432      4
## 15 0.1798662 0.24967850      2
## 16 0.7985790 0.08324688      4
## 17 0.5247573 0.57527380      3
## 18 0.2358468 0.23087585      2
## 19 0.8818853 0.49668824      5
## 20 0.4255094 0.81789418      3

plot(stream_5G,  n  =  700,  method  =  "pc")

```

Note that if you add *noise* to your stream, e.g., `stream_Noise <- DSD_Gaussians(k = 5, d = 4, noise = .1, p = c(0.1, 0.5, 0.3, 0.9, 0.1))`, then the noise points that are not classified as part of any cluster will have an NA class label.

16.3.6 Stream Animation

Clusters can be animated over time by `animate_data()`. Use `reset_stream()` to start the animation at the beginning of the stream and note that this method is **not implemented** for streams of class `DSD_Gaussians`, `DSD_R`, `DSD_data.frame`, and `DSD`. We'll create a new `DSD_Benchmark` data stream (Fig. 16.8).

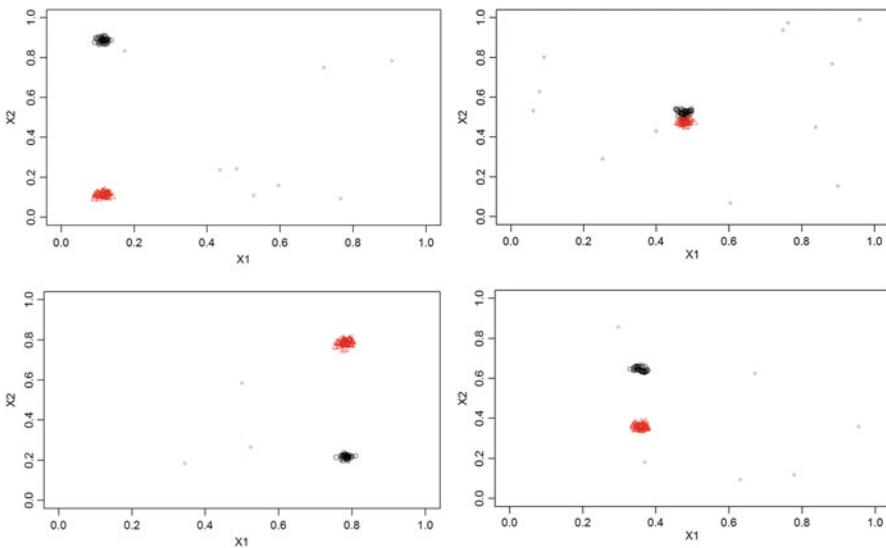


Fig. 16.8 Discrete snapshots of the animated stream clustering process

```
set.seed(12345)
stream_Bench <- DSD_Benchmark(1)
stream_Bench

## Benchmark 1: Two clusters moving diagonally from left to right, meeting
## in
## the center (5% noise).
## Class: DSD_MG, DSD_R, DSD_data.frame, DSD
## With 2 clusters in 2 dimensions. Time is 1
library("animation")
reset_stream(stream_Bench)
animate_data(stream_Bench, n=10000, horizon=100, xlim=c(0,1), ylim=c(0,1))
```

This benchmark generator creates two 2D clusters moving in 2D. One moves from *top-left* to *bottom-right*, the other from *bottom-left* to *top-right*. Then they meet at the center of the domain, the 2 clusters overlap and then split again.

Concept drift in the stream can be depicted by requesting (10) times 300 data points from the stream and animating the plot. Fast-forwarding the stream can be accomplished by requesting, but ignoring, (2000) points in between the (10) plots. The output of the animation below is suppressed to save space.

```
for(i in 1:10) {
  plot(stream_Bench, 300, xlim = c(0, 1), ylim = c(0, 1))
  tmp <- get_points(stream_Bench, n = 2000)
}
```

```
reset_stream(stream_Bench)
animate_data(stream_Bench, n=8000, horizon=120, xlim=c(0,1), ylim=c(0,1))
# Animations can be saved as HTML or GIF
#saveHTML(ani.replay(), htmlfile = "stream_Bench_Animation.html")
#saveGIF(ani.replay())
```

Streams can also be saved locally by `write_stream(stream_Bench, "dataStreamSaved.csv", n = 100, sep = ",")` and loaded back in R by `DSD_ReadCSV()`.

16.3.7 Case-Study: SOCR Knee Pain Data

These data represent the X and Y spatial knee-pain locations for over 8,000 patients, along with *labels* about the knee *Front*, *Back*, *Left* and *Right*. Let's try to read the SOCR Knee Pain Dataset as a stream.

```
library("XML"); library("xml2"); library("rvest")

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_KneePainData_041409")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top"
...
kneeRawData <- html_table(html_nodes(wiki_url, "table"))[[2]]
normalize<-function(x){
  return((x-min(x))/(max(x)-min(x)))
}
kneeRawData_df <- as.data.frame(cbind(normalize(kneeRawData$x),
normalize(kneeRawData$Y), as.factor(kneeRawData$View)))
colnames(kneeRawData_df) <- c("X", "Y", "Label")
# randomize the rows of the DF as RF, RB, LF and LB labels of classes are
sequential
set.seed(1234)
kneeRawData_df <- kneeRawData_df[sample(nrow(kneeRawData_df)), ]
summary(kneeRawData_df)

##           X                  Y                  Label
##  Min.   :0.0000   Min.   :0.0000   Min.   :1.000
##  1st Qu.:0.1331   1st Qu.:0.4566   1st Qu.:2.000
##  Median :0.2995   Median :0.5087   Median :3.000
##  Mean   :0.3382   Mean   :0.5091   Mean   :2.801
##  3rd Qu.:0.3645   3rd Qu.:0.5549   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :1.0000   Max.   :4.000

# View(kneeRawData_df)
```

We can use the `DSD::DSD_Memory` class to get a stream interface for matrix or data frame objects, like the Knee pain location dataset. The number of true clusters $k = 4$ in this dataset.

```
# use data.frame to create a stream (3rd column contains label assignment)
kneeDF <- data.frame(x=kneeRawData_df[,1], y=kneeRawData_df[,2],
  class=as.factor(kneeRawData_df[,3]))
head(kneeDF)
##          x      y class
## 1 0.1188590 0.5057803     4
## 2 0.3248811 0.6040462     2
## 3 0.3153724 0.4971098     2
## 4 0.3248811 0.4161850     2
## 5 0.6941363 0.5289017     1
## 6 0.3217116 0.4595376     2

streamKnee <- DSD_Memory(kneeDF[,c("x", "y")], class=kneeDF[,"class"],
  Loop=T)
streamKnee

## Memory Stream Interface
## Class: DSD_Memory, DSD_R, DSD_data.frame, DSD
## With NA clusters in 2 dimensions
## Contains 8666 data points - currently at position 1 - Loop is TRUE

# Each time we get a point from *streamKnee*, the stream pointer moves
# to the next position (row) in the data.
get_points(streamKnee, n=10)

##          x      y
## 1 0.11885895 0.5057803
## 2 0.32488114 0.6040462
## 3 0.31537242 0.4971098
## 4 0.32488114 0.4161850
## 5 0.69413629 0.5289017
## 6 0.32171157 0.4595376
## 7 0.06497623 0.4913295
## 8 0.12519810 0.4682081
## 9 0.32329635 0.4942197
## 10 0.30744849 0.5086705

streamKnee

## Memory Stream Interface
## Class: DSD_Memory, DSD_R, DSD_data.frame, DSD
## With NA clusters in 2 dimensions
## Contains 8666 data points - currently at position 11 - Loop is TRUE

# Stream pointer is in position 11 now

# We can redirect the current position of the stream pointer by:
reset_stream(streamKnee, pos = 200)
get_points(streamKnee, n=10)

##          x      y
## 200 0.9413629 0.5606936
## 201 0.3217116 0.5664740
## 202 0.3122029 0.6416185
## 203 0.1553090 0.6040462
## 204 0.3645008 0.5346821
## 205 0.3122029 0.5000000
## 206 0.3549921 0.5404624
## 207 0.1473851 0.5260116
## 208 0.1870048 0.6329480
## 209 0.1220285 0.4132948
```

```
streamKnee
## Memory Stream Interface
## Class: DSD_Memory, DSD_R, DSD_data.frame, DSD
## With NA clusters in 2 dimensions
## Contains 8666 data points - currently at position 210 - Loop is TRUE
```

16.3.8 Data Stream Clustering and Classification (DSC)

Let's demonstrate clustering using `DSC_DStream`, which assigns points to cells in a grid. First, initialize the clustering, as an empty cluster and then use the `update()` function to implicitly alter the mutable `DSC` object (Fig. 16.9).

```
dsc_streamKnee <- DSC_DStream(gridsize = 0.1, Cm = 0.4, attraction=T)
dsc_streamKnee

## DStream
## Class: DSC_DStream, DSC_Micro, DSC_R, DSC
## Number of micro-clusters: 0
## Number of macro-clusters: 0

# stream:::update
reset_stream(streamKnee, pos = 1)
update(dsc_streamKnee, streamKnee, n = 500)
dsc_streamKnee

## DStream
## Class: DSC_DStream, DSC_Micro, DSC_R, DSC
## Number of micro-clusters: 16
## Number of macro-clusters: 11
```

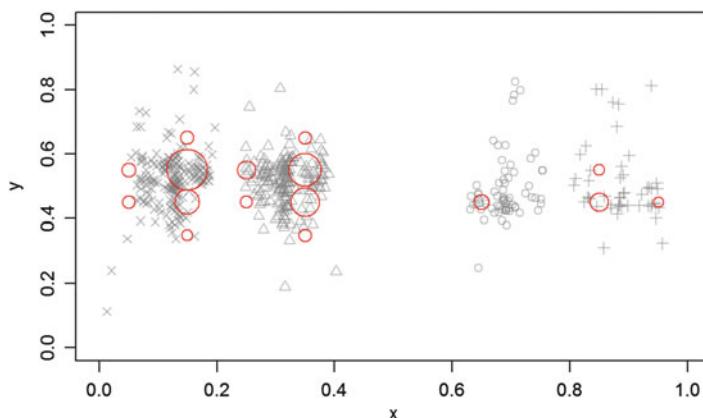


Fig. 16.9 Data stream clustering and classification of the SOCR knee-pain dataset (n=500)

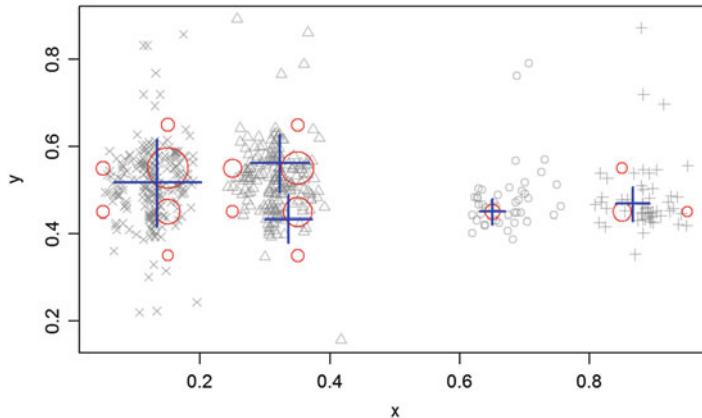


Fig. 16.10 5-Means stream clustering of the SOCR knee pain data

```

head(get_centers(dsc_streamKnee))

##      [,1] [,2]
## [1,] 0.05 0.45
## [2,] 0.05 0.55
## [3,] 0.15 0.35
## [4,] 0.15 0.45
## [5,] 0.15 0.55
## [6,] 0.15 0.65

plot(dsc_streamKnee, streamKnee, xlim=c(0,1), ylim=c(0,1))

# plot(dsc_streamKnee, streamKnee, grid = TRUE)
# Micro-clusters are plotted in red on top of gray stream data points
# The size of the micro-clusters indicates their weight - it's proportional
# to the number of data points represented by each micro-cluster.
# Micro-clusters are shown as dense grid cells (density is coded with gray
# values).

```

The **purity** metric represents an external evaluation criterion of cluster quality, which is the proportion of the total number of points that were correctly classified:

$$0 \leq \text{Purity} = \frac{1}{N} \sum_{i=1}^k \max_j |c_i \cap t_j| \leq 1,$$

where N =number of observed data points, k = number of clusters, c_i is the i^{th} cluster, and t_j is the classification that has the maximum number of points with c_i class labels. High purity suggests that we correctly label points (Fig. 16.10).

Next, we can use K-means clustering.

```
kMeans_Knee <- DSC_Kmeans(k=5) # use 4-5 clusters matching the 4 knee labels
recluster(kMeans_Knee, dsc_streamKnee)
plot(kMeans_Knee, streamKnee, type = "both")
```

Again, the graphical output of the animation sequence of frames is suppressed, however, the readers are encouraged to run the command line and inspect the graphical outcome (Figs. 16.11 and 16.12).

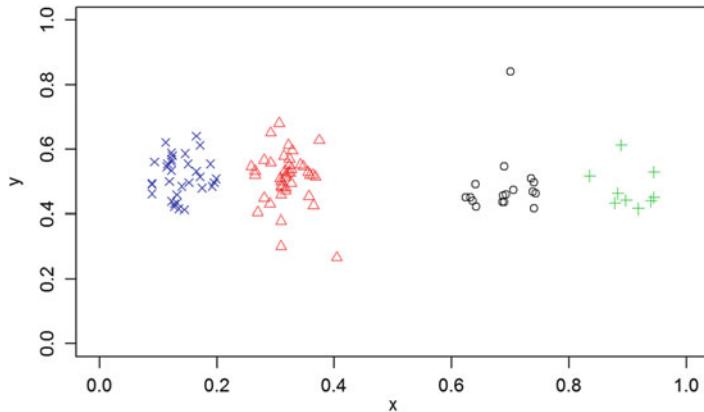


Fig. 16.11 Animated continuous 5-means stream clustering of the knee pain data

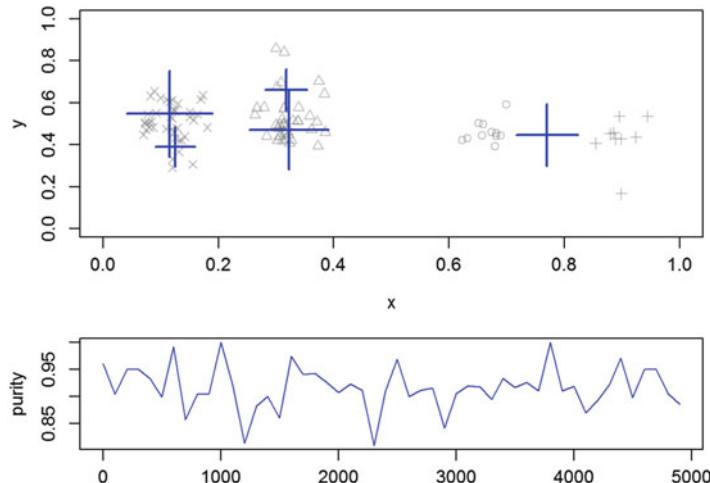


Fig. 16.12 Continuous stream clustering and purity index across iterations

```
animate_data(streamKnee, n=1000, horizon=100, xlim=c(0,1), ylim = c(0,1))
```

```
# purity <- animate_cluster(kMeans_Knee, streamKnee, n=2500, type="both",
  xlim=c(0,1), ylim=c(-,1), evaluationMeasure="purity", horizon=10)
```

```
animate_cluster(kMeans_Knee, streamKnee, horizon = 100, n = 5000,
  measure = "purity", plot.args = list(xlim = c(0, 1), ylim = c(0, 1)))
```

```
##   points   purity
## 1      1 0.9600000
## 2     101 0.9043478
## 3     201 0.9500000
...
## 49   4801 0.9047619
## 50   4901 0.8850000
```

16.3.9 Evaluation of Data Stream Clustering

Figure 16.13 shows the average clustering purity as we evaluate the stream clustering across the streaming points.

```
# Synthetic Gaussian example
# stream <- DSD_Gaussians(k = 3, d = 2, noise = .05)
# dstream <- DSC_DStream(gridsize = .1)
# update(dstream, stream, n = 2000)
# evaluate(dstream, stream, n = 100)

evaluate(dsc_streamKnee, streamKnee, measure = c("crand", "SSQ",
  "silhouette"), n = 100, type = c("auto", "micro", "macro"), assign="micro",
  assignmentMethod = c("auto", "model", "nn"), noise = c("class", "exclude"))

## Evaluation results for micro-clusters.
## Points were assigned to micro-clusters.
##   cRand      SSQ silhouette
## 0.3473634 0.3382900 0.1373143

clusterEval <- evaluate_cluster(dsc_streamKnee, streamKnee, measure =
  c("numMicroClusters", "purity"), n = 5000, horizon = 100)
head(clusterEval)

##   points numMicroClusters   purity
## 1      0           16 0.9555556
## 2     100          17 0.9733333
## 3     200          18 0.9671053
## 4     300          21 0.9687500
## 5     400          21 0.9880952
## 6     500          22 0.9750000

plot(clusterEval[, "points"], clusterEval[, "purity"], type = "l",
  ylab = "Avg Purity", xlab = "Points")
```

```
animate_cluster(dsc_streamKnee, streamKnee, horizon = 100, n = 5000,
  measure = "purity", plot.args = list(xlim = c(0, 1), ylim = c(0, 1)))
```

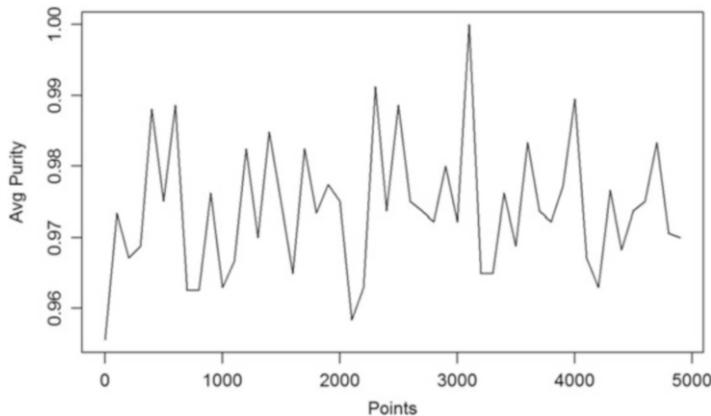


Fig. 16.13 Average clustering purity

```
##      points      purity
## 1        1 0.9714286
## 2      101 0.9833333
## 3     201 0.9722222
...
## 49    4801 0.9772727
## 50    4901 0.9777778
```

The `dsc_streamKnee` represents the result of the stream clustering, where n is the number of data points from the `streamKnee` stream. The evaluation measure can be specified as a vector of character strings. Points are assigned to clusters in `dsc_streamKnee` using `get_assignment()` and can be used to assess the quality of the classification. By default, points are assigned to *micro-clusters*, or can be assigned to *macro-cluster* centers by `assign = "macro"`. Also, new points can be assigned to clusters by the rule used in the clustering algorithm by `assignmentMethod = "model"` or using nearest-neighbor assignment (`nn`), Fig. 16.14.

16.4 Optimization and Improving the Computational Performance

Here and in previous chapters, e.g., Chap. 15, we notice that R may sometimes be slow and memory-inefficient. These problems may be severe, especially for datasets with millions of records or when using complex functions. There are packages for processing large datasets and memory optimization – `bigmemory`, `biganalytics`, `bigtabulate`, etc.

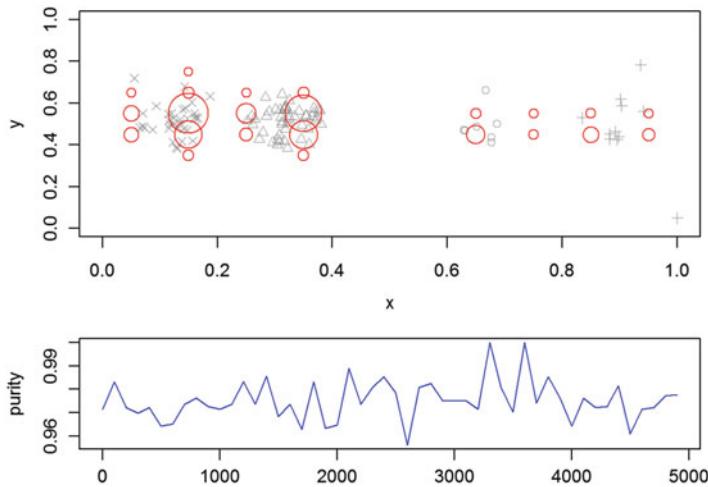


Fig. 16.14 Continuous k-means stream clustering with classification purity

16.4.1 Generalizing Tabular Data Structures with `dplyr`

We have also seen long execution times when running processes that ingest, store or manipulate huge `data.frame` objects. The `dplyr` package, created by Hadley Wickham and Romain Francois, provides a faster route to manage such large datasets in R. It creates an object called `tbl`, similar to `data.frame`, which has an in-memory column-like structure. R reads these objects a lot faster than data frames.

To make a `tbl` object we can either convert an existing data frame to `tbl` or connect to an external database. Converting from data frame to `tbl` is quite easy. All we need to do is call the function `as.tbl()`.

```
#install.packages("dplyr")
library(dplyr)

nof1_tbl<-as.tbl(nof1); nof1_tbl

## # A tibble: 900 × 10
##   ID   Day   Tx SelfEff SelfEff25  WPSS SocSuppt  PMss PMss3 PhyAct
##   <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     1     1     33     8  0.97  5.00  4.03  1.03   53
## 2     1     2     1     33     8 -0.17  3.87  4.03  1.03   73
## 3     1     3     0     33     8  0.81  4.84  4.03  1.03   23
...
## 8     1     8     0     33     8 -0.34  3.69  4.03  1.03     0
## 9     1     9     1     33     8 -0.74  3.29  4.03  1.03   73
## 10    1    10     1     33     8 -0.38  3.66  4.03  1.03  114
## # ... with 890 more rows
```

This looks like a normal data frame. If you are using R Studio, displaying the `nof1_tbl` will show the same output as `nof1`.

16.4.2 *Making Data Frames Faster with Data.Table*

Similar to `tbl`, the `data.table` package provides another alternative to data frame object representation. `data.table` objects are processed in R much faster compared to standard data frames. Also, all of the functions that can accept data frame could be applied to `data.table` objects as well. The function `fread()` is able to read a local CSV file directly into a `data.table`.

```
#install.packages("data.table")
library(data.table)

nof1<-fread("C:/Users/Dinov/Desktop/02_Nof1_Data.csv")
```

Another amazing property of `data.table` is that we can use subscripts to access a specific location in the dataset just like `dataset[row, column]`. It also allows the selection of rows with Boolean expression and direct application of functions to those selected rows. Note that column names can be used to call the specific column in `data.table`, whereas with data frames, we have to use the `dataset$columnName` syntax.

```
nof1[ID==1, mean(PhyAct)]
## [1] 52.66667
```

This useful functionality can also help us run complex operations with only a few lines of code. One of the drawbacks of using `data.table` objects is that they are still limited by the available system memory.

16.4.3 *Creating Disk-Based Data Frames with ff*

The `ff` (fast-files) package allows us to overcome the RAM limitations of finite system memory. For example, it helps with operating datasets with billions of rows. `ff` creates objects in `ffdf` formats, which is like a map that points to a location of the data on a disk. However, this makes `ffdf` objects inapplicable for most R functions. The only way to address this problem is to break the huge dataset into small chunks. After processing a batch of these small chunks, we have to combine the results to reconstruct the complete output. This strategy is relevant in parallel computing, which will be discussed in detail in the next section. First, let's download one of the large datasets in our datasets archive, `UQ_VitalSignsData_Case04.csv`.

```
# install.packages("ff")
library(ff)
# vitalsigns<-read.csv.ffdf(file="UQ_VitalSignsData_Case04.csv", header=T)
vitalsigns<-
read.csv.ffdf(file="https://umich.instructure.com/files/366335/download?
download_frd=1", header=T)
```

As mentioned earlier, we cannot apply functions directly on this object.

```
mean(vitalsigns$Pulse)
## Warning in mean.default(vitalsigns$Pulse): argument is not numeric or
## Logical: returning NA
## [1] NA
```

For basic calculations on such large datasets, we can use another package, `ffbase`. It allows operations on `ffdf` objects using simple tasks like: mathematical operations, query functions, summary statistics and bigger regression models using packages like `biglm`, which will be mentioned later in this chapter.

```
# install.packages("ffbase")
library(ffbase)
mean(vitalsigns$Pulse)
## [1] 108.7185
```

16.4.4 Using Massive Matrices with `bigmemory`

The previously introduced packages include alternatives to `data.frames`. For instance, the `bigmemory` package creates alternative objects to 2D matrices (second-order tensors). It can store huge datasets and can be divided into small chunks that can be converted to data frames. However, we cannot directly apply machine-learning methods on this type of objects. More detailed information about the `bigmemory` package is available online.

16.5 Parallel Computing

In previous chapters, we saw various machine-learning techniques applied as serial computing tasks. The traditional protocol involves: First, applying *function 1* to our raw data. Then, using the output from *function 1* as an input to *function 2*. This process may be iterated over a series of functions. Finally, we have the terminal output generated by the last function. This serial or linear computing method is straightforward but time consuming and perhaps sub-optimal.

Now we introduce a more efficient way of computing - *parallel computing*, which provides a mechanism to deal with different tasks at the same time and combine the

outputs for all of processes to get the final answer faster. However, parallel algorithms may require special conditions and cannot be applied to all problems. If two tasks have to be run in a specific order, this problem cannot be parallelized.

16.5.1 Measuring Execution Time

To measure how much time can be saved for different methods, we can use function `system.time()`.

```
system.time(mean(vitalsigns$Pulse))
##    user  system elapsed
##      0      0      0
```

This means calculating the mean of `Pulse` column in the `vitalsigns` dataset takes less than 0.001 seconds. These values will vary between computers, operating systems, and states of operations.

16.5.2 Parallel Processing with Multiple Cores

We will introduce two packages for parallel computing `multicore` and `snow` (their core components are included in the package `parallel`). They both have a different way of multitasking. However, to run these packages, you need to have a relatively modern multicore computer. Let's check how many cores your computer has. This function `parallel::detectCores()` provides this functionality. `parallel` is a base package, so there is no need to install it prior to using it.

```
library(parallel); detectCores()
## [1] 8
```

So, there are eight (8) cores in my computer. I will be able to run up to 6-8 parallel jobs on this computer.

The `multicore` package simply uses the multitasking capabilities of the *kernel*, the computer's operating system, to "fork" additional R sessions that share the same memory. Imagine that we open several R sessions in parallel and let each of them do part of the work. Now, let's examine how this can save time when running complex protocols or dealing with large datasets. To start with, we can use the `mcapply()` function, which is similar to `lapply()`, which applies functions to a vector and returns a vector of lists. Instead of applying functions to vectors `mcapply()` divides the complete computational task and delegates portions of it to each available core. To demonstrate this procedure, we will construct a simple, yet time

consuming, task of generating random numbers. Also, we can use the `system.time()` function to track execution time.

```
set.seed(123)
system.time(c1<-rnorm(10000000))

##      user  system elapsed
##      0.64    0.00    0.64

# Note the multi core calls may not work on Windows, but will work on
# Linux/Mac.
# This shows a 2-core and 4-core invocations
# system.time(c2<-unlist(mclapply(1:2, function(x){rnorm(5000000)},
# mc.cores = 2)))
# system.time(c4<-unlist(mclapply(1:4, function(x){rnorm(2500000)},
# mc.cores = 4)))

# And here is a Windows (single core invocation)
system.time(c2<-unlist(mclapply(1:2, function(x){rnorm(5000000)},
mc.cores = 1)))

##      user  system elapsed
##      0.65    0.00    0.65
```

The `unlist()` is used at the end to combine results from different cores into a single vector. Each line of code creates 10,000,000 random numbers. The `c1` call took the longest time to complete. The `c2` call used two cores to finish the task (each core handled 5,000,000 numbers) and used less time than `c1`. Finally, `c4` used all four cores to finish the task and successfully reduced the overall time. We can see that when we use more cores the overall time is significantly reduced.

The `snow` package allows parallel computing on multicore multiprocessor machines or a network of multiple machines. It might be more difficult to use but it's also certainly more flexible. First we can set how many cores we want to use via `makeCluster()` function.

```
# install.packages("snow")
library(snow)

cl<-makeCluster(2)
```

This call might cause your computer to pop up a message warning about access though the firewall. To do the same task we can use `parLapply()` function in the `snow` package. Note that we have to call the object we created with the previous `makeCluster()` function.

```
system.time(c2<-unlist(parLapply(cl, c(5000000, 5000000), function(x) {
rnorm(x)})))

##      user  system elapsed
##      0.11    0.11    0.64
```

While using `parLapply()`, we have to specify the matrix and the function that will be applied to this matrix. Remember to stop the cluster we made after completing the task, to release back the system resources.

```
stopCluster(cl)
```

16.5.3 Parallelization Using `foreach` and `doParallel`

The `foreach` package provides another option of parallel computing. It relies on a loop-like process basically applying a specified function for each item in the set, which again is somewhat similar to `apply()`, `lapply()` and other regular functions. The interesting thing is that these loops can be computed in parallel saving substantial amounts of time. The `foreach` package alone cannot provide parallel computing. We have to combine it with other packages like `doParallel`. Let's reexamine the task of creating a vector of 10,000,000 random numbers. First, register the 4 compute cores using `registerDoParallel()`.

```
# install.packages("doParallel")
library(doParallel)

cl<-makeCluster(4)
registerDoParallel(cl)
```

Then we can examine the time saving `foreach` command.

```
#install.packages("foreach")
library(foreach)
system.time(c4<-foreach(i=1:4, .combine = 'c')
            %dopar% rnorm(2500000))

##    user  system elapsed
##    0.11    0.18    0.54
```

Here we used four items (each item runs on a separate core), `.combine=c` allows `foreach` to combine the results with the parameter `c()`, generating the aggregate result vector.

Also, don't forget to close the `doParallel` by registering the sequential backend.

```
unregister<-registerDoSEQ()
```

16.5.4 GPU Computing

Modern computers have graphics cards, GPUs (Graphical Processing Units), that consists of thousands of cores, however they are very specialized, unlike the standard CPU chip. If we can use this feature for parallel computing, we may reach amazing performance improvements, at the cost of complicating the processing algorithms and increasing the constraints on the data format. Specific disadvantages of GPU computing include reliance on proprietary manufacturer (e.g., NVidia) frameworks and Complete Unified Device Architecture (CUDA) programming language. CUDA allows programming of GPU instructions into a common computing language. This paper provides one example of using GPU computation to significantly improve the performance of advanced neuroimaging and brain mapping processing of multidimensional data.

The R package `gputools` is created for parallel computing using NVidia CUDA. Detailed GPU computing in R information is available online.

16.6 Deploying Optimized Learning Algorithms

As we mentioned earlier, some tasks can be parallelized easier than others. In real world situations, we can pick the algorithms that lend themselves well to parallelization. Some of the R packages that allow parallel computing using ML algorithms are listed below.

16.6.1 Building Bigger Regression Models with `biglm`

`biglm` allows training regression models with data from SQL databases or large data chunks obtained from the `ff` package. The output is similar to the standard `lm()` function that builds linear models. However, `biglm` operates efficiently on massive datasets.

16.6.2 Growing Bigger and Faster Random Forests with `bigrf`

The `bigrf` package can be used to train random forests combining the `foreach` and `doParallel` packages. In Chap. 15, we presented random forests as machine learners ensembling multiple tree learners. With parallel computing, we can split the task of creating thousands of trees into smaller tasks that can be outsourced to each

available compute core. We only need to combine the results at the end. Then, we will obtain the exact same output in a relatively shorter amount of time.

16.6.3 *Training and Evaluation Models in Parallel with caret*

Combining the `caret` package with `foreach`, we can obtain a powerful method to deal with time-consuming tasks like building a random forest learner. Utilizing the same example we presented in Chap. 15, we can see the time difference of utilizing the `foreach` package.

```
#library(caret)
system.time(m_rf <- train(CHARLSONSCORE ~ ., data = qol, method = "rf",
metric = "Kappa", trControl = ctrl, tuneGrid = grid_rf))
##    user  system elapsed
##  130.05    0.40 130.49
```

It took more than a minute to finish this task in standard execution model purely relying on the regular `caret` function. Below, this same model training completes much faster using parallelization (less than half the time) compared to the standard call above.

```
set.seed(123)
cl<-makeCluster(4)
registerDoParallel(cl)
getDoParWorkers()

## [1] 4

system.time(m_rf <- train(CHARLSONSCORE ~ ., data = qol, method = "rf",
metric = "Kappa", trControl = ctrl, tuneGrid = grid_rf))
##    user  system elapsed
##    4.61    0.02  47.70

unregister<-registerDoSEQ()
```

16.7 Practice Problem

Try to analyze the co-appearance network in the novel “Les Miserables”. The data contains the weighted network of co-appearances of characters in Victor Hugo’s novel “Les Miserables”. Nodes represent characters as indicated by the labels and edges connect any pair of characters that appear in the same chapter of the book. The values on the edges are the number of such co-appearances.

```
miserables<-read.table("https://umich.instructure.com/files/330389/download?download_frd=1", sep="", header=F) head(miserables)
```

Also, try to interrogate some of the larger datasets we have by using alternative parallel computing and big data analytics.

16.8 Assignment: 16. Specialized Machine Learning Topics

16.8.1 Working with Website Data

- Download the Main SOCR Wiki Page and compare `RCurl` and `httr`.
- Read and write XML code for the SOCR Main Page.
- Scrape the data from the SOCR Main Page.

16.8.2 Network Data and Visualization

- Download `03_les_miserablese_GraphData.txt`
- Visualize this undirected network.
- Summary the graph and explain the output.
- Calculate degree and the centrality of this graph.
- Find out some important characters.
- Will the result change or not if we assume the graph is directed?

16.8.3 Data Conversion and Parallel Computing

- Download `CaseStudy12_AdultsHeartAttack_Data.xlsx` or require online.
- load this data as data frame.
- Use `Export()` or `write.xlsx()` to renew the `.xlsx` file.
- Use `rio` package to convert this `".xlsx"` file to `".csv"`.
- Generate generalizing tabular data structures.
- Generate `data.table`.
- Create disk-based data frames and perform basic calculation.
- Perform basic calculation on the last 5 columns as a big matrix.
- Use `DIAGNOSIS`, `SEX`, `DRG`, `CHARGES`, `LOS` and `AGE` to predict `DIED` with `randomForest` setting `ntree=20000`. Notice: sample without replacement to get an as large as possible balanced dataset.
- Run `train()` in `caret` and detect the execute time.
- Detect cores and make proper number of clusters.

- Rerun `train()` parallelized and compare the execute time.
- Use `foreach` and `doMC` to design a parallelized random forest with `ntree=20000` totally and compare the execute time with sequential execution.

References

- Data Streams in R:<https://cran.r-project.org/web/packages/stream/vignettes/stream.pdf>
Dplyr:<https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html>
doParallel:<https://cran.rproject.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf>
Mailund, T. (2017) *Beginning Data Science in R: Data Analysis, Visualization, and Modelling for the Data Scientist*, Apress, ISBN 1484226712, 9781484226711

Chapter 17

Variable/Feature Selection



As we mentioned in Chap. 16, variable selection is very important when dealing with bioinformatics, healthcare, and biomedical data, where we may have more features than observations. Variable selection, or feature selection, can help us focus only on the core important information contained in the observations, instead of every piece of information. Due to presence of intrinsic and extrinsic noise, the volume and complexity of big health data, and different methodological and technological challenges, this process of identifying the salient features may resemble finding a needle in a haystack. Here, we will illustrate alternative strategies for feature selection using filtering (e.g., correlation-based feature selection), wrapping (e.g., recursive feature elimination), and embedding (e.g., variable importance via random forest classification) techniques.

The next Chap. 18, provides the details about another powerful technique for variable-selection using *decoy features* to control the false discovery rate of choosing inconsequential features.

17.1 Feature Selection Methods

There are three major classes of variable or feature selection techniques—filtering-based, wrapper-based, and embedded methods.

17.1.1 Filtering Techniques

- *Univariate*: Univariate filtering methods focus on selecting single features with high scores based on some statistics like χ^2 or Information Gain Ratio. Each

feature is viewed as independent of the others, effectively ignoring interactions between features.

- Examples: χ^2 , Euclidean distance, *t*-test, and Information gain.
- *Multivariate*: Multivariate filtering methods rely on various (multivariate) statistics to select the principal features. They typically account for between-feature interactions by using higher-order statistics like correlation. The basic idea is that we iteratively triage variables that have high correlations with other features.
- Examples: Correlation-based feature selection, Markov blanket filter, and fast correlation-based feature selection.

17.1.2 *Wrapper Methods*

- *Deterministic*: Deterministic wrapper feature selection methods either start with no features (forward-selection) or with all features included in the model (backward-selection) and iteratively refine the set of chosen features according to some model quality measures. The iterative process of adding or removing features may rely on statistics like the Jaccard similarity coefficient.
 - Examples: Sequential forward selection, Recursive Feature Elimination, Plus q take-away r , and Beam search.
- *Randomized*: Stochastic wrapper feature selection procedures utilize a binary feature-indexing vector indicating whether or not each variable should be included in the list of salient features. At each iteration, we *randomly* perturb the binary indicators vector and compare the combinations of features before and after the random inclusion-exclusion indexing change. Finally, we pick the indexing vector corresponding with the optimal performance based on some metric, like acceptance probability measures. The iterative process continues until no improvement of the objective function is observed.
 - Examples: Simulated annealing, genetic algorithms, distribution- and kernel-estimation algorithms.

17.1.3 *Embedded Techniques*

- Embedded-feature selection techniques are based on various classifiers, predictors, or clustering procedures. For instance, we can accomplish feature selection by using decision trees where the separation of the training data relies on features associated with the highest information gain. Further tree branching, separating the data deeper, may utilize *weaker* features. This process of choosing the vital features based on their separability characteristics continues until the classifier

generates group labels that are mostly homogeneous within clusters/classes and largely heterogeneous across groups, and when the information gain of further tree branching is marginal. The entire process may be iterated multiple times to select the features that appear most frequently.

- Examples: Decision trees, random forests, weighted naive Bayes, and feature selection using weighted-SVM.

The different types of feature selection methods have their own pros and cons. In this chapter, we are going to introduce the randomized wrapper method using the `Boruta` package, which utilizes the random forest classification method to output variable importance measures (VIMs). Then, we will compare its results with Recursive Feature Elimination, a classical deterministic wrapper method.

17.2 Case Study: ALS

17.2.1 Step 1: Collecting Data

First things first, let's explore the dataset we will be using. Case Study 15, Amyotrophic Lateral Sclerosis (ALS), examines the patterns, symmetries, associations and causality in a rare but devastating disease, amyotrophic lateral sclerosis (ALS), also known as *Lou Gehrig disease*. This ALS case-study reflects a large clinical trial including big, multi-source and heterogeneous datasets. It would be interesting to interrogate the data and attempt to derive potential biomarkers that can be used for detecting, prognosticating, and forecasting the progression of this neurodegenerative disorder. Overcoming many scientific, technical and infrastructure barriers is required to establish complete, efficient, and reproducible protocols for such complex data. These pipeline workflows start with ingesting the raw data, preprocessing, aggregating, harmonizing, analyzing, visualizing and interpreting the findings.

In this case-study, we use the training dataset that contains 2223 observations and 131 numeric variables. We select `ALSFRS slope` as our outcome variable, as it captures the patients' clinical decline over a year. Although we have more observations than features, this is one of the examples where multiple features are highly correlated. Therefore, we need to preprocess the variables, e.g., apply feature selection, before commencing with predictive analytics.

17.2.2 Step 2: Exploring and Preparing the Data

The dataset is located in our case-studies archive. We can use `read.csv()` to directly import the CSV dataset into R using the URL reference.

```
ALS.train<-read.csv("https://umich.instructure.com/files/1789624/download?do
wnload_frd=1")
summary(ALS.train)

##          ID          Age_mean      Albumin_max      Albumin_median
##  Min.   : 1.0   Min.   :18.00   Min.   :37.00   Min.   :34.50
##  1st Qu.: 614.5  1st Qu.:47.00  1st Qu.:45.00  1st Qu.:42.00
##  Median :1213.0  Median :55.00  Median :47.00  Median :44.00
##  Mean   :1214.9  Mean   :54.55  Mean   :47.01  Mean   :43.95
##  3rd Qu.:1815.5  3rd Qu.:63.00  3rd Qu.:49.00  3rd Qu.:46.00
##  Max.   :2424.0   Max.   :81.00  Max.   :70.30  Max.   :51.10
...
##  Urine.Ph_median  Urine.Ph_min
##  Min.   :5.000   Min.   :5.000
##  1st Qu.:5.000   1st Qu.:5.000
##  Median :6.000   Median :5.000
##  Mean   :5.711   Mean   :5.183
##  3rd Qu.:6.000   3rd Qu.:5.000
##  Max.   :9.000   Max.   :8.000
```

There are 131 features and some of variables represent statistics like *max*, *min* and *median* values of the same clinical measurements.

17.2.3 Step 3: Training a Model on the Data

Now let's explore the `Boruta()` function in the `Boruta` package to perform variables selection, based on random forest classification. `Boruta()` includes the following components:

```
vs<-Boruta(class~features, data=Mydata, pValue = 0.01, mcAdj =
TRUE, maxRuns = 100, doTrace=0, getImp = getImpRfZ, ...)
```

- `class`: variable for class labels.
- `features`: potential features to select from.
- `data`: dataset containing classes and features.
- `pValue`: confidence level. Default value is 0.01 (Notice we are applying multiple variable selection).
- `mcAdj`: Default TRUE to apply a multiple comparisons adjustment using the Bonferroni method.
- `maxRuns`: maximal number of importance source runs. You may increase it to resolve attributes left Tentative.
- `doTrace`: verbosity level. Default 0 means no tracing, 1 means reporting decision about each attribute as soon as it is justified, 2 means same as 1, plus at each importance source run reporting the number of attributes. The default is 0 where we don't do the reporting.

- `getImp`: function used to obtain attribute importance. The default is `getImpRfZ`, which runs random forest, from the ranger package, and gathers Z-scores of the mean decreased accuracy measure.

The resulting `vs` object is of class `Boruta` and contains two important components:

- `finalDecision`: a factor of three values: `Confirmed`, `Rejected` or `Tentative`, containing the final results of the feature selection process.
- `ImpHistory`: a data frame of importance of attributes gathered in each importance source run. Besides the predictors' importance, it contains maximal, mean and minimal importance of shadow attributes for each run. Rejected attributes get `-Inf` importance. This output is set to `NULL` if we specify `holdHistory=FALSE` in the `Boruta` call.

Note: Running the code below will take several minutes.

```
# install.packages("Boruta")
library(Boruta)
set.seed(123)
als<-Boruta(ALSFRS_slope~.-ID, data=ALS.train, doTrace=0)
print(als)

## Boruta performed 99 iterations in 4.683657 mins.
## 28 attributes confirmed important: ALSFRS_Total_max,
## ALSFRS_Total_median, ALSFRS_Total_min, ALSFRS_Total_range,
## Creatinine_median and 23 more;
## 59 attributes confirmed unimportant: Albumin_max, Albumin_median,
## Albumin_min, ALT.SGPT._max, ALT.SGPT._median and 54 more;
## 12 tentative attributes left: Age_mean, Albumin_range,
## Creatinine_max, Hematocrit_median, Hematocrit_range and 7 more;

als$ImpHistory[1:6, 1:10]

##          Age_mean Albumin_max Albumin_median Albumin_min Albumin_range
## [1,] 1.2031427 1.4969268 0.6976378 0.9385041 1.979510
## [2,] -0.1998469 0.7204092 -1.5626360 0.5777092 2.573882
## [3,] 1.9272058 -1.0274668 0.2216170 -1.2234402 1.843967
## [4,] 0.5763244 0.9097371 0.2960979 0.6137624 2.184383
## [5,] 3.3655147 1.9412326 0.3849548 1.7309793 1.134676
## [6,] 0.2603118 -0.0287943 1.4164860 2.3251879 2.259974
##          ALSFRS_Total_max ALSFRS_Total_median ALSFRS_Total_min
## [1,] 6.925233 9.551064 15.92924
## [2,] 8.124101 7.867399 14.94650
## [3,] 7.443326 8.735702 17.26469
## [4,] 7.578267 7.868885 16.95563
## [5,] 7.554582 7.248834 15.42697
## [6,] 7.516362 7.145460 14.94824
##          ALSFRS_Total_range ALT.SGPT._max
## [1,] 25.78135 4.1516252
## [2,] 26.11722 1.2187027
## [3,] 25.61523 2.1618804
## [4,] 28.19229 0.4305607
## [5,] 24.90620 1.2043325
## [6,] 26.57093 0.8463782
```

This is a fairly time-consuming computation. Boruta determines the *important* attributes from *unimportant* and *tentative* features. Here the importance is measured by the out-of-bag (OOB) error. The OOB estimates the prediction error of machine-learning methods (e.g., random forests and boosted decision trees) that utilize bootstrap aggregation to sub-sample training data. **OOB** represents the mean prediction error on each training sample x_i , using only the trees that did not include x_i in their bootstrap samples. Out-of-bag estimates provide *internal* assessment of the learning accuracy and avoid the need for an independent *external* validation dataset.

The importance scores for all features at every iteration are stored in the data frame `als$ImpHistory`. Let's plot a graph depicting the essential features.

Note: Again, running this code will take several minutes to complete (Fig. 17.1).

```
plot(als, xlab="", xaxt="n")
Lz<-lapply(1:ncol(als$ImpHistory), function(i)
als$ImpHistory[is.finite(als$ImpHistory[, i]), i])
names(Lz)<-colnames(als$ImpHistory)
Lb<-sort(sapply(Lz, median))
axis(side=1, las=2, labels=names(Lb), at=1:ncol(als$ImpHistory),
cex.axis=0.5, font = 4)
```

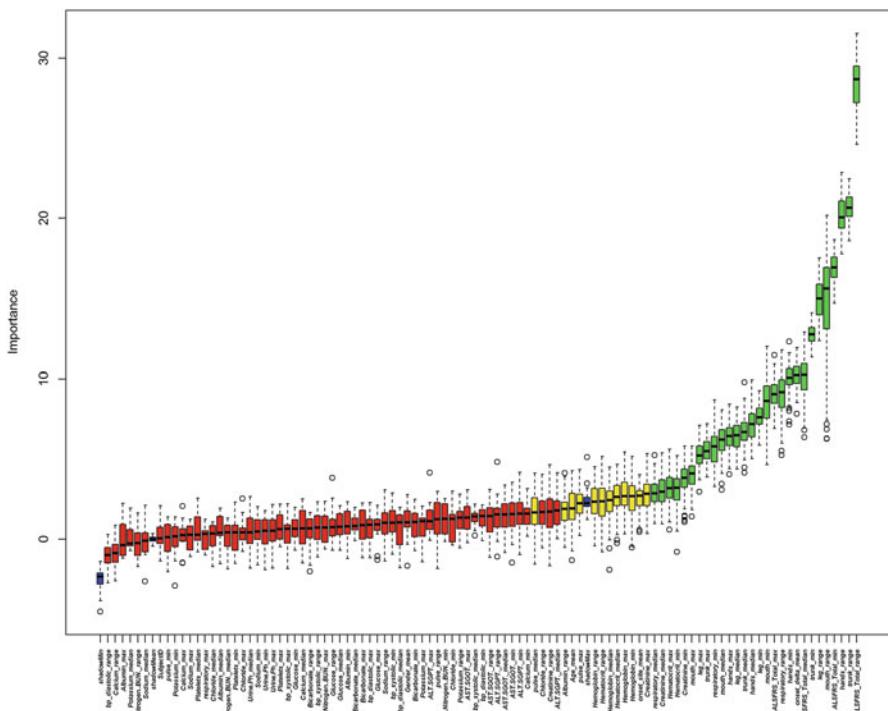


Fig. 17.1 Ranked variables importance using box and whisker plots for each feature

We can see that plotting the graph is easy but extracting matched feature names may require more work. The basic plot is done by this call `plot(als, xlab="", xaxt="n")`, where `xaxt="n"` means we suppress plotting of *x*-axis. The following lines in the script reconstruct the *x*-axis plot. `lz` is a list created by the `lapply()` function. Each element in `lz` contains all the important scores for a single feature in the original dataset. Also, we excluded all rejected features with infinite importance. Then, we sorted these non-rejected features according to their median importance and printed them on the *x*-axis by using `axis()`.

We have already seen similar groups of boxplots back in Chaps. 3 and 4. In this graph, variables with *green* boxes are more important than the ones represented with *red* boxes, and we can see the range of importance scores within a single variable in the graph.

It may be desirable to get rid of tentative features. Notice that this function should be used only when strict decision is highly desired, because this test is much weaker than Boruta and can lower the confidence of the final result.

```
final.als<-TentativeRoughFix(als)
print(final.als)

## Boruta performed 99 iterations in 4.683657 mins.
## Tentatives roughfixed over the last 99 iterations.
## 32 attributes confirmed important: ALSFRS_Total_max,
## ALSFRS_Total_median, ALSFRS_Total_min, ALSFRS_Total_range,
## Creatinine_median and 27 more;
## 67 attributes confirmed unimportant: Age_mean, Albumin_max,
## Albumin_median, Albumin_min, Albumin_range and 62 more;

final.als$finalDecision

##                               Age_mean           Albumin_max
##                               Rejected          Rejected
##                               Albumin_median       Albumin_min
##                               Rejected          Rejected
##                               Albumin_range       ALSFRS_Total_max
##                               Rejected          Confirmed
## ALSFRS_Total_median       ALSFRS_Total_min
##                               Confirmed          Confirmed
...
##                               Urine.Ph_max        Urine.Ph_median
##                               Rejected          Rejected
##                               Urine.Ph_min
##                               Rejected
## Levels: Tentative Confirmed Rejected

getConfirmedFormula(final.als)

## ALSFRS_slope ~ ALSFRS_Total_max+ALSFIRS_Total_median + ALSFRS_Total_min +
## ALSFRS_Total_range + Creatinine_median + Creatinine_min +
## hands_max + hands_median + hands_min + hands_range+Hematocrit_max+
## Hematocrit_min+Hematocrit_range+Hemoglobin_median+Hemoglobin_range +
## leg_max + leg_median + leg_min + leg_range + mouth_max +
## mouth_median + mouth_min + mouth_range + onset_delta_mean +
## pulse_max+respiratory_median + respiratory_min + respiratory_range+
## trunk_max + trunk_median + trunk_min + trunk_range
## <environment: 0x000000000989d6f8>
```

The report above shows the final features selection including only the “confirmed” and “Tentative” features.

17.2.4 Step 4: Evaluating Model Performance

Comparing with RFE

Let's compare the Boruta results against a classical variable selection method—*recursive feature elimination (RFE)*. First, we need to load two packages: `caret` and `randomForest`. Then, as we did in Chap. 15, we must specify a resampling method. Here we use *10-fold CV* to do the resampling.

```
library(caret)
library(randomForest)
set.seed(123)
control<-rfeControl(functions = rfFuncs, method = "cv", number=10)
```

Now, all preparations are complete and we are ready to do the RFE variable selection.

```

rf.train<-rfe(ALS.train[, -c(1, 7)], ALS.train[, 7], sizes=c(10,20,30,40),
rfeControl=control)
rf.train

## Recursive feature selection
## Outer resampling method: Cross-Validated (10 fold)
## Resampling performance over subset size:
## Variables   RMSE Rquared   RMSESD RquaredSD Selected
##          10 0.3500  0.6837  0.03451   0.03837
##          20 0.3471  0.6894  0.03230   0.03374
##          30 0.3468  0.6900  0.03135   0.02967      *
##          40 0.3473  0.6895  0.03061   0.02887
##          99 0.3503  0.6842  0.02995   0.02868

## The top 5 variables (out of 30):
## ALSFRS_Total_range, trunk_range, hands_range, mouth_range, ALSFRS_Total_min

```

This calculation may take a long time to complete. The RFE invocation is different from Boruta. Here we have to specify the feature data frame and the class labels separately. Also, the `sizes=` option allows us to specify the number of features we want to include in the model. Let's try `sizes=c(10, 20, 30, 40)` to compare the model performance for alternative numbers of features.

To visualize the results, we can plot the RMSE error for the five different feature size combinations listed in the summary. The one with 30 features has the lowest RMSE value. This result is similar to the Boruta output, which selected around 30 features (Fig. 17.2).

```
plot(rf.train, type=c("g", "o"), cex=1, col=1:5)
```

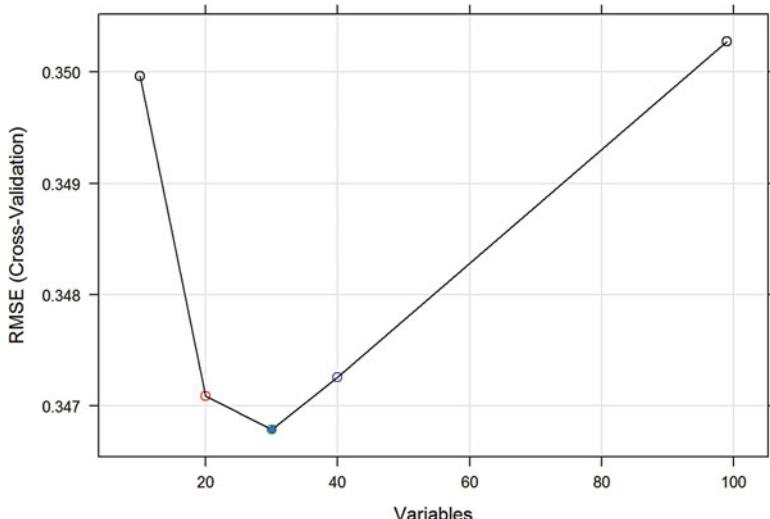


Fig. 17.2 Root-mean square cross-validation error rate for random forest classification of the ALS study against the number of features

Using the functions `predictors()` and `getSelectedAttributes()`, we can compare the final results of the two alternative feature selection methods.

```
predRFE <- predictors(rf.train)
predBoruta <- getSelectedAttributes(final.als, withTentative = F)
```

The Boruta and RFE feature-selection results are almost identical.

```
intersect(predBoruta, predRFE)
## [1] "ALSFRS_Total_max"    "ALSFRS_Total_median" "ALSFRS_Total_min"
## [4] "ALSFRS_Total_range"   "Creatinine_min"      "hands_max"
## [7] "hands_median"        "hands_min"        "hands_range"
## [10] "Hematocrit_max"     "Hemoglobin_median" "Leg_max"
## [13] "Leg_median"         "Leg_min"        "Leg_range"
## [16] "mouth_median"        "mouth_min"        "mouth_range"
## [19] "onset_delta_mean"    "respiratory_median" "respiratory_min"
## [22] "respiratory_range"   "trunk_max"        "trunk_median"
## [25] "trunk_min"          "trunk_range"
```

There are 26 common variables chosen by the two techniques, which suggests that both the Boruta and RFE methods are robust. Also, notice that the Boruta method can give similar results without utilizing the `size` option. If we want to consider ten or more different sizes, the procedure will be quite time consuming. Thus, the Boruta method is effective when dealing with complex real world problems.

Comparing with Stepwise Feature Selection

Next, we can contrast the Boruta feature selection results against another classical variable selection method – *stepwise model selection*. Let's start with fitting a bidirectional stepwise linear model-based feature selection.

```
data2 <- ALS.train[, -1]
# Define a base model - intercept only
base.mod <- Lm(ALSFRS_slope ~ 1, data= data2)
# Define the full model - including all predictors
all.mod <- Lm(ALSFRS_slope ~ ., data= data2)
# ols_step <- lm(ALSFRS_slope ~ ., data= data2)
ols_step <- step(base.mod, scope = List(Lower=base.mod, upper = all.mod),
direction = 'both', k=2, trace = F)
summary(ols_step); ols_step

## Call:
## Lm(formula = ALSFRS_slope ~ ALSFRS_Total_range + ALSFRS_Total_median +
##       ALSFRS_Total_min + Calcium_range + Calcium_max + bp_diastolic_min +
##       onset_delta_mean + Calcium_min + Albumin_range + Glucose_range +
##       ALT.SGPT._median + AST.SGOT._median + Glucose_max + Glucose_min +
##       Creatinine_range + Potassium_range + Chloride_range + Chloride_min +
##       Sodium_median + respiratory_min + respiratory_range + respiratory_max +
##       trunk_range + pulse_range + Bicarbonate_max + Bicarbonate_range +
##       Chloride_max + onset_site_mean + trunk_max + Gender_mean +
##       Creatinine_min, data = data2)
```


We can report the stepwise “Confirmed” (salient) features:

```
# get the shortlisted variable
stepwiseConfirmedVars <- names(unlist(ols_step[[1]]))
# remove the intercept
stepwiseConfirmedVars <- stepwiseConfirmedVars[!stepwiseConfirmedVars %in% "
(Intercept)"]
print(stepwiseConfirmedVars)

## [1] "ALSFRS_Total_range" "ALSFRS_Total_median" "ALSFRS_Total_min"
## [4] "Calcium_range" "Calcium_max" "bp_diastolic_min"
## [7] "onset_delta_mean" "Calcium_min" "Albumin_range"
## [10] "Glucose_range" "ALT.SGPT._median" "AST.SGOT._median"
## [13] "Glucose_max" "Glucose_min" "Creatinine_range"
## [16] "Potassium_range" "Chloride_range" "Chloride_min"
## [19] "Sodium_median" "respiratory_min" "respiratory_range"
## [22] "respiratory_max" "trunk_range" "pulse_range"
## [25] "Bicarbonate_max" "Bicarbonate_range" "Chloride_max"
## [28] "onset_site_mean" "trunk_max" "Gender_mean"
## [31] "Creatinine_min"
```

Again, the feature selection results of Boruta and step are similar.

```
library(mlbench)
library(caret)

# estimate variable importance
predStepwise <- varImp(ols_step, scale=FALSE)
# summarize importance
print(predStepwise)

##                                     Overall
## ALSFRS_Total_range 16.630592
## ALSFRS_Total_median 11.812263
## ALSFRS_Total_min    8.523606
## Calcium_range       5.754045
## Calcium_max         4.812942
## bp_diastolic_min   2.539766
## onset_delta_mean    2.758465
## Calcium_min          3.767450
## Albumin_range        2.812018
## Glucose_range         5.156259
## ALT.SGPT._median    2.876338
## AST.SGOT._median    2.641369
## Glucose_max          4.629759
## Glucose_min          4.022642
## Creatinine_range     2.293301
## Potassium_range       1.739268
## Chloride_range        4.474709
## Chloride_min          4.403551
## Sodium_median         2.118710
## respiratory_min       5.948488
## respiratory_range      5.756735
## respiratory_max        5.041816
## trunk_range            2.819029
## pulse_range             1.696811
```

```

## Bicarbonate_max      2.568068
## Bicarbonate_range   2.303757
## Chloride_max        1.750666
## onset_site_mean     1.663481
## trunk_max           2.706410
## Gender_mean          1.919380
## Creatinine_min       1.535642

# plot predStepwise
# plot(predStepwise)

# Boruta vs. Stepwise feature selection
intersect(predBoruta, stepwiseConfirmedVars)

## [1] "ALSFRS_Total_median" "ALSFRS_Total_min"      "ALSFRS_Total_range"
## [4] "Creatinine_min"       "onset_delta_mean"   "respiratory_min"
## [7] "respiratory_range"    "trunk_max"         "trunk_range"

```

There are about nine common variables chosen by the Boruta and Stepwise feature selection methods.

There is another more elaborate stepwise feature selection technique that is implemented in the function `MASS::stepAIC()` that is useful for a wider range of object classes.

17.3 Practice Problem

You can practice variable selection using the `SOCR_Data_AD_BiomedBigMetadata` on the SOCR website. This is a smaller dataset that has 744 observations and 63 variables. Here we utilize `DXCURREN` or current diagnostics as the class variable.

Let's import the dataset first.

```

library(rvest)

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_AD_Bio
medBigMetadata")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top
...
alzh <- html_table(html_nodes(wiki_url, "table"))[[1]]
summary(alzh)

##      SID          MMSCORE        FAQTOTAL        GDTOTAL
##  Min. : 2.0  Min. :18.00  Length:744  Min. :0.000
##  1st Qu.:355.5 1st Qu.:25.00  Class :character 1st Qu.:0.000
##  Median :697.5 Median :27.00  Mode :character Median :1.000
##  Mean   :707.5 Mean   :26.81                    Mean   :1.367
##  3rd Qu.:1063.0 3rd Qu.:29.00                    3rd Qu.:2.000
##  Max.  :1435.0  Max.  :30.00                    Max.  :6.000

```

```
##      CDHOME          CDCARE          CDGLOBAL
##  Min. :0.0000  Min. :0.0000  Min. :0.0000
##  1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000
##  Median :0.0000 Median :0.0000 Median :0.0000
##  Mean  :0.2513  Mean  :0.2849  Mean  :0.0672
##  3rd Qu.:0.5000 3rd Qu.:0.5000 3rd Qu.:0.0000
##  Max.  :2.0000  Max.  :2.0000  Max.  :2.0000
```

The data summary shows that we have several factor variables. After converting their type to numeric we find some missing data. We can manage this issue by selecting only the complete observation of the original dataset or by using multivariate imputation, see Chap. 3.

```
chrtofactor<-c(3, 5, 8, 10, 21:22, 51:54)
alzh[chrtofactor]<-data.frame(apply(alzh[chrtofactor], 2, as.numeric))
alzh<-alzh[complete.cases(alzh), ]
```

For simplicity, here we eliminated the missing data and are left with 408 complete observations. Now, we can apply the Boruta method for feature selection.

```
## Boruta performed 99 iterations in 9.413648 secs.
## 12 attributes confirmed important: adascog, BCBREATH, CDCARE,
## CDCOMMUN, CDGLOBAL and 7 more;
## 47 attributes confirmed unimportant: Age, BC.USSEA, BCABDOMN,
## BCANKLE, BCCHEST and 42 more;
## 2 tentative attributes left: ApoEGeneAllele1, ApoEGeneAllele2;
```

You might get a result that is a little bit different. We can plot the variable importance graph using some previous knowledge (Fig. 17.3).

The final step is to get rid of the tentative features.

```
## Boruta performed 99 iterations in 9.413648 secs.
## Tentatives roughfixed over the last 99 iterations.
## 14 attributes confirmed important: adascog, ApoEGeneAllele1,
## ApoEGeneAllele2, BCBREATH, CDCARE and 9 more;
## 47 attributes confirmed unimportant: Age, BC.USSEA, BCABDOMN,
## BCANKLE, BCCHEST and 42 more;
## [1] "MMSCORE"          "FAQTOTAL"        "adascog"
## [4] "sobcdr"            "DX_Confidence"   "BCBREATH"
## [7] "ApoEGeneAllele1"  "ApoEGeneAllele2" "CDORIENT"
## [10] "CDJUDGE"           "CDCOMMUN"        "CDHOME"
## [13] "CDCARE"            "CDGLOBAL"
```

Can you reproduce these results? Also try to apply some of these techniques to other data from the list of our Case-Studies.

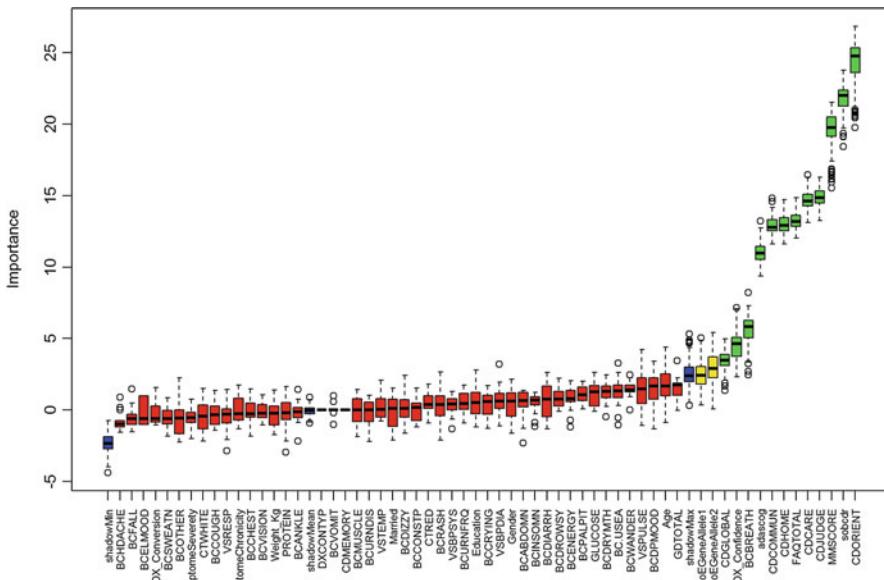


Fig. 17.3 Variable importance plot of predicting diagnosis for the Alzheimer's disease case-study

17.4 Assignment: 17. Variable/Feature Selection

17.4.1 Wrapper Feature Selection

- Explain the three major types of feature selection methods
 - Filter,
 - Wrapper, and
 - Embedded.

17.4.2 Use the PPMI Dataset

Use the 06_PPMI_ClassificationValidationData_Short dataset setting ResearchGroup as class variable.

- Delete irrelevant columns (e.g. X, FID_IID) and select only the *PD* and *Control* cases.
 - Properly convert the variable types.
 - Apply Boruta to train a model, try different parameters (e.g., try different *pValue*, *maxRuns*). What are the differences?
 - Summarize and visualize the results.

- Apply *Random Feature Elimination* (RFE) and tune the model size.
- Evaluate the Boruta model performance by comparing with REF.
- Output and compare the variables selected by both methods. How much overlap is there in the selected variables?

References

- Guyon, E, Gunn, S, Nikravesh, M, Zadeh, LA (eds.) (2008) *Feature Extraction: Foundations and Applications*, Springer, ISBN 3540354883, 9783540354888
- Liu, H and Motoda, H (eds.) (2007) *Computational Methods of Feature Selection*, Chapman & Hall/CRC, ISBN 1584888792, 9781584888796
- Pacheco, ER (2015) *Unsupervised Learning with R*, Packt Publishing, ISBN 1785885812, 9781785885815

Chapter 18

Regularized Linear Modeling and Controlled Variable Selection



Many biomedical and biosocial studies involve large amounts of complex data, including cases where the number of features (k) is large and may exceed the number of cases (n). In such situations, parameter estimates are difficult to compute or may be unreliable as the system is underdetermined. Regularization provides one approach to improve model reliability, prediction accuracy, and result interpretability. It is based on augmenting the primary fidelity term of the objective function used in the model-fitting process with a dual regularization term that provides restrictions on the parameter space.

Classical techniques for choosing *important* covariates to include in a model of complex multivariate data rely on various types of stepwise variable selection processes, see Chap. 17. These tend to improve prediction accuracy in certain situations, e.g., when a small number of features are strongly predictive, or associated, with the clinical outcome or biosocial trait. However, the prediction error may be large when the model relies purely on a fidelity term. Including a regularization term in the optimization of the cost function improves the prediction accuracy. For example, below we show that by shrinking large regression coefficients, ridge regularization reduces overfitting and decreases the prediction error. Similarly, the Least Absolute Shrinkage and Selection Operator (LASSO) employs regularization to perform simultaneous parameter estimation and variable selection. LASSO enhances the prediction accuracy and provides a natural interpretation of the resulting model. *Regularization* refers to forcing certain characteristics of model-based scientific inference, e.g., discouraging complex models or extreme explanations, even if they fit the data well, by enforcing model generalizability to prospective data, or restricting model overfitting of accidental samples.

In this chapter, we extend the mathematical foundation we presented in Chap. 5 and (1) discuss computational protocols for handling complex high-dimensional data, (2) illustrate model estimation by controlling the false-positive rate of selection of salient features, and (3) derive effective forecasting models.

18.1 Questions

- How to deal with extremely high-dimensional data (hundreds or thousands of features)?
- Why mix fidelity (model fit) and regularization (model interpretability) terms in objective function optimization?
- How to reduce the false-positive rate, increase scientific validation, and improve result reproducibility (e.g., Knockoff filtering)?

18.2 Matrix Notation

We should review the basics of matrix notation, linear algebra, and matrix computing we covered in Chap. 5. At the core of matrix manipulations are scalars, vectors and matrices.

- y_i : output or response variable, $i = 1, \dots, n$ (cases/subjects).
- x_{ij} : input, predictor, or feature variable, $1 \leq j \leq k$, $1 \leq i \leq n$.

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

and

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} \end{pmatrix}.$$

18.3 Regularized Linear Modeling

If we assume that the covariates are orthonormal, i.e., we have a special kind of a *design matrix* $X^T X = I$, then:

- The ordinary least squares (OLS) estimates minimize

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 \right\},$$

and are defined by

$$\hat{\beta}^{OLS} = (X^T X)^{-1} X^T y = X^T y,$$

- LASSO estimates minimize

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\},$$

and are defined as a soft-threshold function of the OLS estimates:

$$\hat{\beta}_j = S_{N\lambda}(\hat{\beta}_j^{OLS}) = \hat{\beta}_j^{OLS} \max \left(0, 1 - \frac{N\lambda}{|\hat{\beta}_j^{OLS}|} \right),$$

where $S_{N\lambda}$ is a soft thresholding operator translating values towards zero, instead of setting smaller values to zero and leaving larger ones untouched as the hard thresholding operator would.

- Ridge regression estimates minimize the following objective function:

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2 \right\},$$

which yields estimates $\hat{\beta}_j = (1 + N\lambda)^{-1} \hat{\beta}_j^{OLS}$. Thus, ridge regression shrinks all coefficients by a uniform factor, $(1 + N\lambda)^{-1}$, and does not set any coefficients to zero.

- Best subset selection regression, also called orthogonal matching pursuit (OMP), minimizes:

$$\min_{\beta \in \mathbb{R}^k} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \lambda \|\beta\|_0 \right\},$$

where $\|\cdot\|_0$ is the “ ℓ^0 norm”, defined as $|z| = m$ if exactly m components of z are nonzero. In this case, the estimates are:

$$\hat{\beta}_j = H_{\sqrt{N\lambda}}(\hat{\beta}_j^{OLS}) = \hat{\beta}_j^{OLS} I(|\hat{\beta}_j^{OLS}| \geq \sqrt{N\lambda}),$$

where H_α is a hard-thresholding function and I is an indicator function (it is 1 if its argument is true, and 0 otherwise).

The LASSO estimates share features of the estimates from both ridge and best subset selection regression since they both shrink the magnitude of all the coefficients, like ridge regression. However, LASSO, also sets some of them to zero, as in the best subset selection does. Ridge regression scales all of the coefficients by a constant factor, whereas LASSO translates the coefficients towards zero by a constant value and sets some of them to zero.

18.3.1 Ridge Regression

Ridge regression relies on L^2 regularization to improve the model prediction accuracy. It improves prediction error by shrinking large regression coefficients to reduce overfitting. By itself, ridge regularization does not perform variable selection and does not really help with model interpretation.

Let's look at one example using one of our datasets 01a_data.txt (Figs. 18.1, 18.2 and 18.3).

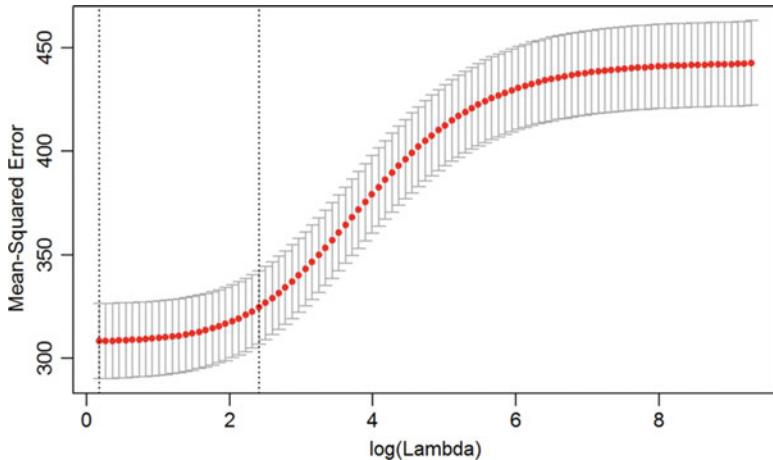


Fig. 18.1 Plot of the MSE rate of the ridge-regularized linear model of MLB player's weight against the regularization weight parameter λ (log scale on the x-axis)

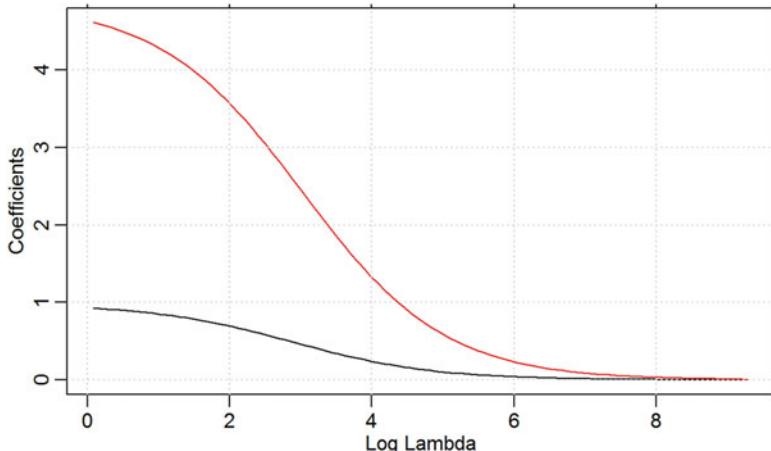


Fig. 18.2 Plot of the effect-size coefficients (Age and Height) of the ridge-regularized linear model of MLB player's weight against the regularization weight parameter λ

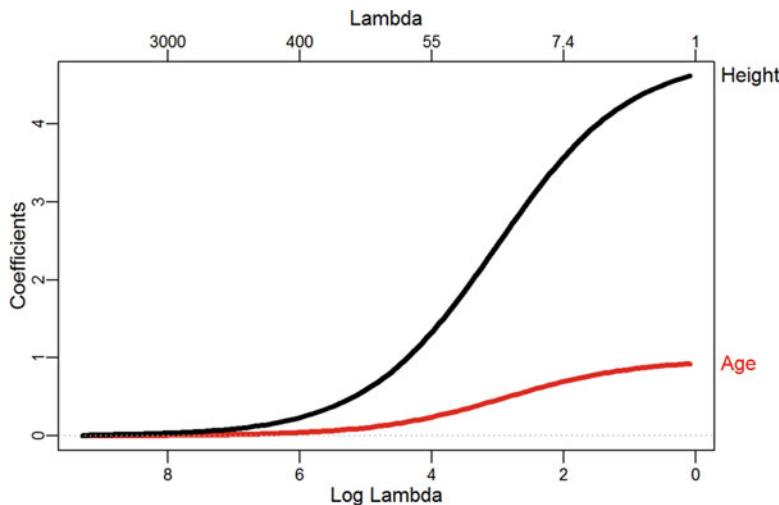


Fig. 18.3 Effect of the regularization weight parameter λ on the model coefficients (Age and Height) of the ridge-regularized linear model of MLB player's weight

```

# Data: https://umich.instructure.com/courses/38100/files/folder/data/01a\_data.txt
data <- read.table('https://umich.instructure.com/files/330381/download?downLoad_frd=1', as.is=T, header=T)
attach(data); str(data)

## 'data.frame': 1034 obs. of 6 variables:
## $ Name   : chr "Adam_Donachie" "Paul_Bako" "Ramon_Hernandez" "Kevin_Millar" ...
## $ Team   : chr "BAL" "BAL" "BAL" "BAL" ...
## $ Position: chr "Catcher" "Catcher" "Catcher" "First_Baseman"
## $ Height : int 74 74 72 72 73 69 69 71 76 71 ...
## $ Weight : int 180 215 210 210 188 176 209 200 231 180 ...
## $ Age    : num 23 34.7 30.8 35.4 35.7 ...

# Training Data
# Full Model: x <- model.matrix(Weight ~ ., data = data[1:900, ])
# Reduced Model
x <- model.matrix(Weight ~ Age + Height, data = data[1:900, ])
# creates a design (or model) matrix, and adds 1 column for outcome
# according to the formula.
y <- data[1:900, ]$Weight

# Testing Data
x.test <- model.matrix(Weight ~ Age + Height, data = data[901:1034, ])
y.test <- data[901:1034, ]$Weight

# install.packages("glmnet")
library("glmnet")

cv.ridge <- cv.glmnet(x, y, type.measure = "mse", alpha = 0)
## alpha = 1 for Lasso only, alpha = 0 for ridge only, and 0 < alpha < 1 to
# blend ridge & Lasso penalty !!!!
plot(cv.ridge)

```

```

coef(cv.ridge)

## 4 x 1 sparse Matrix of class "dgCMatrix"
##                1
## (Intercept) -55.7491733
## (Intercept) .
## Age          0.6264096
## Height       3.2485564

sqrt(cv.ridge$cvm[cv.ridge$lambda == cv.ridge$lambda.1se])

## [1] 17.94358

#plot variable feature coefficients against the shrinkage parameter lambda.
glmmmod <- glmnet(x, y, alpha = 0)
plot(glmmmod, xvar="Lambda")
grid()

# for plot_glmnet with ridge/lasso coefficient path labels
# install.packages("plotmo")
library(plotmo)

plot_glmnet(glmmmod, lwd=4)           #default colors

# More elaborate plots can be generated using:
# plot_glmnet(glmmmod,label=2,lwd=4) #label the 2 biggest final coefs
# specify color of each line
# g <- "blue"
# plot_glmnet(glmmmod, lwd=4, col=c(2,g))

# report the model coefficient estimates
coef(glmmmod)[, 1]

## (Intercept) (Intercept)      Age      Height
## 2.016556e+02 0.000000e+00 8.327372e-37 4.789383e-36

cv.glmmmod <- cv.glmnet(x, y, alpha=0)

mod.ridge <- cv.glmnet(x, y, alpha = 0, thresh = 1e-12)
Lambda.best <- mod.ridge$lambda.min
Lambda.best

## [1] 1.192177

ridge.pred <- predict(mod.ridge, newx = x.test, s = Lambda.best)
ridge.RMS <- mean((y.test - ridge.pred)^2); ridge.RMS

## [1] 264.083

ridge.test.r2 <- 1 - mean((y.test - ridge.pred)^2)/mean((y.test -
mean(y.test))^2)
# plot(cv.glmmmod)
best_lambda <- cv.glmmmod$lambda.min
best_lambda

## [1] 1.192177

```

In the plots above, different colors represents the vector of features, and the corresponding coefficients are displayed as a function of the regularization parameter, λ . The top axis indicates the number of nonzero coefficients at the current value of λ . For LASSO regularization, this top-axis corresponds to the effective degrees of freedom (df) for the model.

Notice the usefulness of Ridge regularization for model estimation in highly ill-conditioned problems ($n \ll k$) where slight feature perturbations may cause disproportionate alterations of the corresponding weight calculations. When λ is very large, the regularization effect dominates the optimization of the objective function and the coefficients tend to zero. At the other extreme, as $\lambda \rightarrow 0$, the resulting model solution tends towards the ordinary least squares (OLS) and the coefficients exhibit large oscillations. In practice, we often need to tune λ to balance this tradeoff.

Also note that in the `cv.glmnet` call, `alpha = 0` (ridge) and `alpha = 1` (LASSO) correspond to different types of regularization, and $0 < \text{alpha} < 1$ corresponds to *elastic net* blended regularization.

18.3.2 Least Absolute Shrinkage and Selection Operator (LASSO) Regression

Estimating the linear regression coefficients in a linear regression model using LASSO involves minimizing an objective function that includes an L^1 regularization term which tends to shrink the number of features. A descriptive representation of the fidelity (left) and regularization (right) terms of the objective function are shown below:

$$\underbrace{\sum_{i=1}^n \left[y_i - \beta_0 - \sum_{j=1}^k \beta_j x_{ij} \right]^2}_{\text{fidelity term}} + \underbrace{\underbrace{\lambda}_{\text{reg. weight}} \times \sum_{j=1}^p |\beta_j|}_{\text{regularization term}}.$$

LASSO jointly achieves model quality, reliability and variable selection by penalizing the sum of the absolute values of the regression coefficients. This forces the shrinkage of certain coefficients effectively acting as a variable selection process. This is similar to ridge regression's penalty on the sum of the squares of the regression coefficients, although ridge regression only shrinks the magnitude of the coefficients without truncating them to 0.

Let's show how to select the regularization weight parameter λ using training data and report the error (e.g., MSE) using testing data.

```

mod.Lasso <- cv.glmnet(x, y, alpha = 1, thresh = 1e-12)
## alpha =1 for Lasso only, alpha = 0 for ridge only, and 0<alpha<1 for elastic net, a blend ridge & Lasso penalty !!!!
Lambda.best <- mod.Lasso$Lambda.min
Lambda.best

## [1] 0.05933494

Lasso.pred <- predict(mod.Lasso, newx = x.test, s = Lambda.best)
LASSO.MSE <- mean((y.test - Lasso.pred)^2); LASSO.MSE
## [1] 261.8194

```

Let's retrieve the estimates of the model coefficients.

```

mod.Lasso <- glmnet(x, y, alpha = 1)
predict(mod.Lasso, s = Lambda.best, type = "coefficients")

## 4 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept) -181.9254079
## (Intercept) .
## Age          0.9654354
## Height       4.8284803

Lasso.test.r2 <- 1 - mean((y.test - Lasso.pred)^2)/mean((y.test -
mean(y.test))^2)

```

Perhaps obtain a classical OLS linear model, as well.

```

lm.fit <- Lm(Weight ~ Age + Height, data = data[1:900, ])
summary(lm.fit)

##
## Call:
## Lm(formula = Weight ~ Age + Height, data = data[1:900, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -50.602  -12.399   -0.718   10.913   74.446
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -184.3736   19.4232 -9.492 < 2e-16 ***
## Age          0.9799    0.1335  7.341 4.74e-13 ***
## Height       4.8561    0.2551 19.037 < 2e-16 ***
## ---
## Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.5 on 897 degrees of freedom
## Multiple R-squared:  0.3088, Adjusted R-squared:  0.3072 
## F-statistic: 200.3 on 2 and 897 DF,  p-value: < 2.2e-16

```

The OLS linear (unregularized) model has slightly larger coefficients and greater MSE than LASSO, which attests to the shrinkage of LASSO (Fig. 18.4).

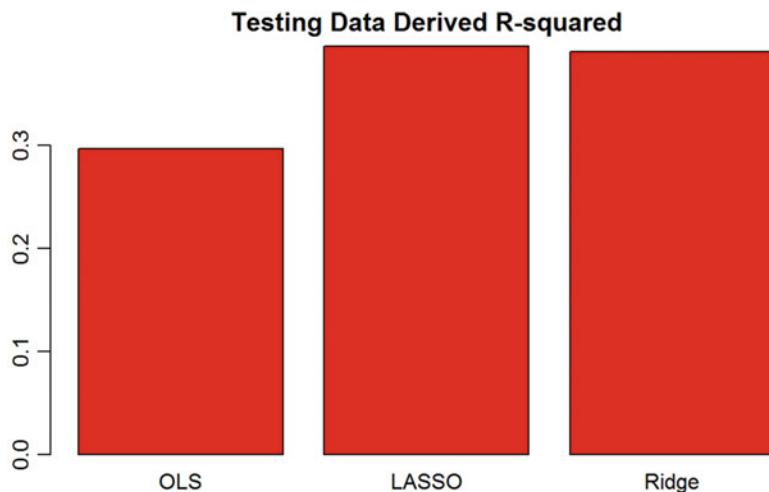


Fig. 18.4 Comparison of the coefficients of determination (R^2) for three alternative models

Table 18.1 Results of the test dataset errors (MSE) for the three methods

LM	LASSO	Ridge
305.1995	261.8194	264.083

```

Lm.pred <- predict(Lm.fit, newx = x.test)
LM.MSE <- mean((y - Lm.pred)^2); LM.MSE
## [1] 305.1995

Lm.test.r2 <- 1 - mean((y - Lm.pred)^2) / mean((y.test - mean(y.test))^2)

barplot(c(Lm.test.r2, Lasso.test.r2, ridge.test.r2), col = "red", names.arg
= c("OLS", "LASSO", "Ridge"), main = "Testing Data Derived R-squared")

```

Compare the results of the three alternative models (LM, LASSO and Ridge) for these data and contrast the derived MSE results (Table. 18.1).

```

library(knitr) # kable function to convert tabular R-results into Rmd tables
# create table as data frame
MSE_Table = data.frame(LM=LM.MSE, LASSO=LASSO.MSE, Ridge=ridge.MSE)

# convert to markdown
kable(MSE_Table, format="pandoc", caption="Results of test dataset errors",
align=c("c", "c", "c"))

```

As both the *inputs* (features or predictors) and the *output* (response) are observed for the testing data, we can *learn* the relationship between the two types of features (controlled covariates and observable responses). Most often, we are interested in forecasting or predicting of responses using prospective (new, testing, or validation) data.

18.3.3 Predictor Standardization

Prior to fitting regularized linear modeling and estimating the effects, covariates may be standardized. This can be accomplished by using the classic “z-score” formula. This puts each predictor on the same scale (unitless quantities) - the mean is 0 and the variance is 1. We use $\hat{\beta}_0 = \bar{y}$, for the mean intercept parameter, and estimate the coefficients or the remaining predictors. To facilitate interpretation of the model or results in the context of the specific case-study, we can transform the results back to the original scale/units after the model is estimated.

18.3.4 Estimation Goals

The basic setting here is: given a set of predictors X , find a function, $f(X)$, to model or predict the outcome Y .

Let's denote the objective (loss or cost) function by $L(y, f(X))$. It determines adequacy of the fit and allows us to estimate the squared error loss:

$$L(y, f(X)) = (y - f(X))^2.$$

We are looking to find f that minimizes the expected loss:

$$E[(Y - f(X))^2] \Rightarrow f = E[Y|X = x].$$

18.4 Linear Regression

Let's assume that:

$$Y_i = \beta_0 + x_{i1}\beta_1 + x_{i2}\beta_2 + \dots + x_{ip}\beta_p + \epsilon.$$

- In shorthand matrix notation, that is: $Y = X\beta + \epsilon$.
- And the expectation of the observed outcome given the data, $E[Y|X = x]$, is a linear function, which in certain situations can be expressed as:

$$\operatorname{argmin}_{\beta} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij}\beta_j \right)^2 = \operatorname{argmin}_{\beta} \sum_{i=1}^n \left(y_i - x_i^T \beta \right)^2.$$

Remember that matrix multiplication is not always commutative. Multiplying on the left both hand sides by $X^T = X'$, the transpose of the design matrix X , yields:

$$X^T Y = X^T (X\beta) = (X^T X)\beta.$$

To solve for the effect-sizes β , we can multiply both sides of the equation by the inverse of its (right hand side) multiplier:

$$(X^T X)^{-1} (X^T Y) = (X^T X)^{-1} (X^T X) \beta = \beta.$$

The *ordinary least squares (OLS)* estimate of β is given by:

$$\begin{aligned}\hat{\beta} &= \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 = \underset{\beta}{\operatorname{argmin}} \| y - X\beta \|_2^2 \Rightarrow \\ \hat{\beta}^{OLS} &= (X^T X)^{-1} X^T y \Rightarrow \hat{f}(x_i) = x_i' \hat{\beta}.\end{aligned}$$

18.4.1 Drawbacks of Linear Regression

Despite its wide use and elegant theory, linear regression has some shortcomings.

- Prediction accuracy – Often can be improved upon;
- Model interpretability – Linear model does not automatically do variable selection.

18.4.2 Assessing Prediction Accuracy

Given a new input, x_0 , how do we assess our prediction $\hat{f}(x_0)$?

The *Expected Prediction Error (EPE)* is:

$$\begin{aligned}EPE(x_0) &= E \left[(Y_0 - \hat{f}(x_0))^2 \right] \\ &= \text{Var}(\epsilon) + \text{Var}(\hat{f}(x_0)) + \text{Bias}(\hat{f}(x_0))^2 \\ &= \text{Var}(\epsilon) + MSE(\hat{f}(x_0))\end{aligned}$$

where

- $\text{Var}(\epsilon)$: irreducible error variance
- $\text{Var}(\hat{f}(x_0))$: sample-to-sample variability of $\hat{f}(x_0)$, and
- $\text{Bias}(\hat{f}(x_0))$: average difference of $\hat{f}(x_0)$ & $f(x_0)$.

18.4.3 Estimating the Prediction Error

Common approaches to estimating prediction errors include:

- Randomly splitting the data into “training” and “testing” sets, where the testing data has m observations that will be used to independently validate the model quality. We estimate/calculate \hat{f} using training data;
- Estimating prediction error using the *testing set MSE*

$$\hat{MSE}(\hat{f}) = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{f}(x_i))^2.$$

Ideally, we want our model/predictions to perform well with new or prospective data.

18.4.4 Improving the Prediction Accuracy

If $f(x) \approx$ linear, \hat{f} will have low bias but possibly high variance, e.g., in high-dimensional setting due to correlated predictors, over (k features $\ll n$ cases), or underdetermination ($k > n$). The goal is to minimize total error by trading off bias and precision:

$$MSE(\hat{f}(x)) = \text{Var}(\hat{f}(x)) + \text{Bias}(\hat{f}(x))^2.$$

We can sacrifice bias to reduce variance, which may lead to a decrease in *MSE*. So, regularization allows us to tune this tradeoff.

We aim to predict the outcome variable, $Y_{n \times 1}$, in terms of other features $X_{n \times k}$. Assume a first-order relationship relating Y and X is of the form $Y = f(X) + \epsilon$, where the error term $\epsilon \sim N(0, \sigma)$. An estimate model $\hat{f}(X)$ can be computed in many different ways (e.g., using least squares calculations for linear regressions, Newton-Raphson, steepest decent and other methods). Then, we can decompose the expected squared prediction error at x as:

$$E(x) = E[(Y - \hat{f}(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{precision (variance)}} + \underbrace{\sigma^2}_{\text{irreducible error (noise)}}.$$

When the true Y vs. X relation is not known, infinite data may be necessary to calibrate the model \hat{f} and it may be impractical to jointly reduce both the model *bias* and *variance*. In general, minimizing the *bias* at the same time as minimizing the *variance* may not be possible.

Figure 18.5 illustrates diagrammatically the dichotomy between *bias* (accuracy) and *precision* (variability). Additional information is available in the SOCR SMHS EBook.

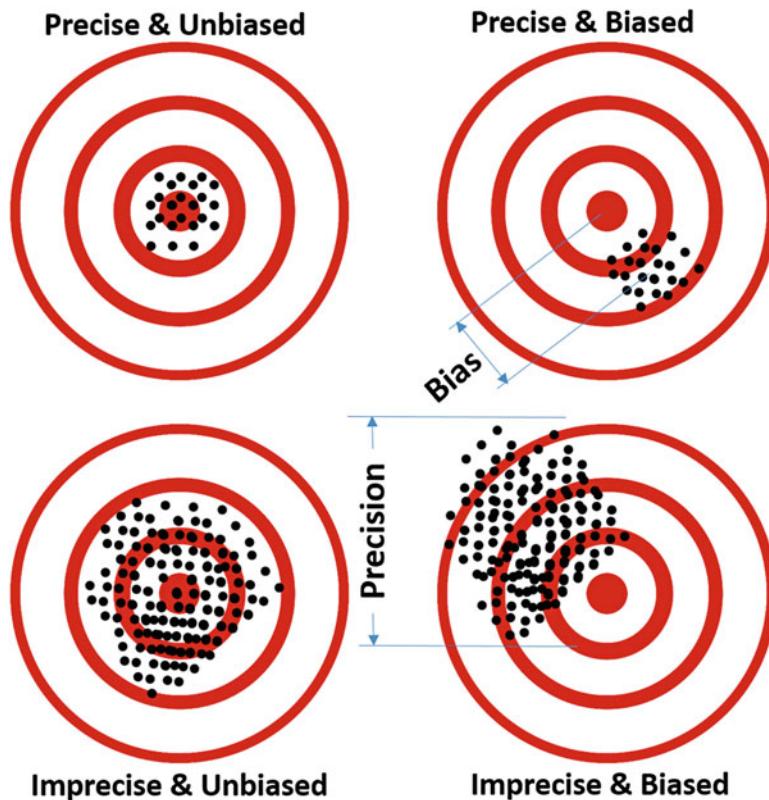


Fig. 18.5 Graphical representation of the four extreme scenarios for bias and precision

18.4.5 Variable Selection

Oftentimes, we are only interested in using a subset of the original features as model predictors. Thus, we need to identify the most relevant predictors, which usually capture the big picture of the process. This helps us avoid overly complex models that may be difficult to interpret. Typically, when considering several models that achieve similar results, it's natural to select the simplest of them.

Linear regression does not directly determine the importance of features to predict a specific outcome. The problem of selecting critical predictors is therefore very important.

Automatic feature subset selection methods should directly determine an optimal subset of variables. Forward or backward stepwise variable selection and forward stagewise are examples of classical methods for choosing the best subset by assessing various metrics like MSE , C_p , AIC, or BIC, see Chap. 17.

18.5 Regularization Framework

As before, we start with a given X and look for a (linear) function, $f(X)$, to model or predict y subject to certain objective cost function, e.g., squared error loss. Adding a second term to the cost function minimization process yields (model parameter) estimates expressed as:

$$\hat{\beta}(\lambda) = \operatorname{argmin}_{\beta} \left\{ \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij}\beta_j \right)^2 + \lambda J(\beta) \right\}.$$

In the above expression, $\lambda \geq 0$ is the regularization (tuning or penalty) parameter, $J(\beta)$ is a user-defined penalty function - typically, the intercept is not penalized.

18.5.1 Role of the Penalty Term

Consider $J(\beta) = \sum_{j=1}^k \beta_j^2 = \|\beta\|_2^2$ (Ridge Regression, *RR*). Then, the formulation of the regularization framework is:

$$\hat{\beta}(\lambda)^{RR} = \operatorname{argmin}_{\beta} \left\{ \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^k \beta_j^2 \right\}.$$

Or, alternatively:

$$\hat{\beta}(t)^{RR} = \operatorname{argmin}_{\beta} \sum_{i=1}^n \left(y_i - \sum_{j=1}^p x_{ij}\beta_j \right)^2,$$

subject to

$$\sum_{j=1}^k \beta_j^2 \leq t.$$

18.5.2 Role of the Regularization Parameter

The regularization parameter $\lambda \geq 0$ directly controls the bias-variance trade-off:

- $\lambda = 0$ corresponds to OLS, and
- $\lambda \rightarrow \infty$ puts more weight on the penalty function and results in more shrinkage of the coefficients, i.e., we introduce bias at the sake of reducing the variance.

The choice of λ is crucial and will be discussed below as each λ results in a different solution $\hat{\beta}(\lambda)$.

18.5.3 LASSO

The LASSO (Least Absolute Shrinkage and Selection Operator) regularization relies on:

$$J(\beta) = \sum_{j=1}^k |\beta_j| = \|\beta\|_1,$$

which leads to the following objective function:

$$\hat{\beta}(\lambda)^L = \operatorname{argmin}_{\beta} \left\{ \sum_{i=1}^n \left(y_i - \sum_{j=1}^k x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^k |\beta_j| \right\}.$$

In practice, subtle changes in the penalty terms frequently lead to big differences in the results. Not only does the regularization term shrink coefficients towards zero, but it sets some of them to be exactly zero. Thus, it performs continuous variable selection, hence the name, Least Absolute Shrinkage and Selection Operator (LASSO).

For further details, see “Tibshirani’s LASSO Page”.

18.5.4 General Regularization Framework

The general regularization framework involves optimization of a more general objective function:

$$\min_{f \in \mathcal{H}} \sum_{i=1}^n \{L(y_i, f(x_i)) + \lambda J(f)\},$$

where \mathcal{H} is a space of possible functions, L is the *fidelity term*, e.g., squared error, absolute error, zero-one, negative log-likelihood (GLM), hinge loss (support vector machines), and J is the *regularizer*, e.g., ridge regression, LASSO, adaptive LASSO, group LASSO, fused LASSO, thresholded LASSO, generalized LASSO, constrained LASSO, elastic-net, Dantzig selector, SCAD, MCP, smoothing splines, etc.

This represents a very general and flexible framework that allows us to incorporate prior knowledge (sparsity, structure, etc.) into the model estimation.

18.6 Implementation of Regularization

18.6.1 Example: Neuroimaging-Genetics Study of Parkinson's Disease Dataset

More information about this specific study and the included derived neuroimaging biomarkers is available [online](#). A link to the data and a brief summary of the features are included below:

- 05_PPMI_top_UPDRS_Integrated_LongFormat1.csv.
- Data elements include: FID_IID, L_insular_cortex_ComputeArea, L_insular_cortex_Volume, R_insular_cortex_ComputeArea, R_insular_cortex_Volume, L_cingulate_gyrus_ComputeArea, L_cingulate_gyrus_Volume, R_cingulate_gyrus_ComputeArea, R_cingulate_gyrus_Volume, L_caudate_ComputeArea, L_caudate_Volume, R_caudate_ComputeArea, R_caudate_Volume, L_putamen_ComputeArea, L_putamen_Volume, R_putamen_ComputeArea, R_putamen_Volume, Sex, Weight, ResearchGroup, Age, chr12_rs34637584_GT, chr17_rs11868035_GT, chr17_rs11012_GT, chr17_rs393152_GT, chr17_rs12185268_GT, chr17_rs199533_GT, UPDRS_part_I, UPDRS_part_II, UPDRS_part_III, time_visit.

Note that the dataset includes missing values and repeated measures.

The *goal* of this demonstration is to use OLS, ridge regression, and the LASSO to **find the best predictive model for the clinical outcomes** – UPDRS score (vector) and Research Group (factor variable), in terms of demographic, genetics, and neuroimaging biomarkers.

We can utilize the `glmnet` package in R for most calculations.

```
##### Initial Stuff #####
# clean up
rm(list=ls())
# load required packages
# install.packages("arm")
library(glmnet)
library(arm)
library(knitr) # kable function to convert tabular R-results into Rmd tables
# pick a random seed, but set.seed(seed) only effects next block of code!
seed = 1234

##### Organize Data #####
# load dataset
# Data: https://umich.instructure.com/courses/38100/files/folder/data
# (05_PPMI_top_UPDRS_Integrated_LongFormat1.csv)
data1 <- read.table('https://umich.instructure.com/files/330397/download?downLoad_frd=1', sep=",", header=T)
# we will deal with missing values using multiple imputation later. For now,
# let's just ignore incomplete cases
data1.completeRowIndexes <- complete.cases(data1);

table(data1.completeRowIndexes)
```

```

## data1.completeRowIndexes
## FALSE TRUE
## 609 1155

prop.table(table(data1.completeRowIndexes))

## data1.completeRowIndexes
## FALSE TRUE
## 0.3452381 0.6547619

attach(data1)
# View(data1[data1.completeRowIndexes, ])
# define response and predictors
y <- data1$UPDRS_part_I + data1$UPDRS_part_II + data1$UPDRS_part_III
table(y) # Show Clinically relevant classification

## y
##0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
###54 20 25 12 8 7 11 16 16 9 21 16 13 13 22 25 21 31 25 29 29 28 20 25 28
###25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
###26 35 41 23 34 32 31 37 34 28 36 29 27 22 19 17 18 18 19 16 9 10 12 9 11
###50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 66 68 69 71 75 80 81 82
###7 10 11 5 7 4 1 5 9 4 3 2 1 6 1 2 1 2 1 1 2 3 1

y <- y[data1.completeRowIndexes]

# X = scale(data1[,]) # Explicit Scaling is not needed, as glmnet auto
# standardizes predictors
# X = as.matrix(data1[, c("R_caudate_Volume", "R_putamen_Volume", "Weight",
# "Age", "chr17_rs12185268_GT")]) # X needs to be a matrix, not a data frame
drop_features <- c("FID_IID", "ResearchGroup", "PDRS_part_I",
"UPDRS_part_II", "UPDRS_part_III")
X <- data1[, !(names(data1) %in% drop_features)]
X = as.matrix(X) # remove columns: index, ResearchGroup, and
y=(PDRS_part_I + UPDRS_part_II + UPDRS_part_III)
X <- X[data1.completeRowIndexes, ]
summary(X)

## L_insular_cortex_ComputeArea L_insular_cortex_Volume
## Min. : 50.03             Min. : 22.63
## 1st Qu.:2174.57           1st Qu.: 5867.23
## Median :2522.52           Median : 7362.90
## Mean   :2306.89           Mean   : 6710.18
## 3rd Qu.:2752.17           3rd Qu.: 8483.80
## Max.  :3650.81           Max.  :13499.92
...
## chr17_rs393152_GT chr17_rs12185268_GT chr17_rs199533_GT UPDRS_part_I
## Min. :0.0000  Min. :0.0000  Min. :0.0000  Min. : 0.000
## 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.:0.0000 1st Qu.: 0.000
## Median :0.0000  Median :0.0000  Median :0.0000  Median : 1.000
## Mean   :0.4468  Mean   :0.4268  Mean   :0.4052  Mean   : 1.306
## 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.:1.0000 3rd Qu.: 2.000
## Max.  :2.0000  Max.  :2.0000  Max.  :2.0000  Max.  :13.000
## time_visit
## Min. : 0.00
## 1st Qu.: 9.00
## Median :24.00
## Mean   :23.83
## 3rd Qu.:36.00
## Max.  :54.00

```

```

# randomly split data into training (80%) and test (20%) sets
set.seed(seed)
train = sample(1 : nrow(X), round((4/5) * nrow(X)))
test = -train

# subset training data
yTrain = y[train]
XTrain = X[train, ]
XTrainOLS = cbind(rep(1, nrow(XTrain)), XTrain)

# subset test data
yTest = y[test]
XTest = X[test, ]

##### Model Estimation & Selection #####
# Estimate models
fitOLS = lm(yTrain ~ XTrain) # Ordinary Least Squares
# glmnet automatically standardizes the predictors
fitRidge = glmnet(XTrain, yTrain, alpha = 0) # Ridge Regression
fitLASSO = glmnet(XTrain, yTrain, alpha = 1) # The LASSO

```

Readers are encouraged to compare the two models, *ridge* and *LASSO*.

18.6.2 Computational Complexity

Recall that the regularized regression estimates depend on the regularization parameter λ . Fortunately, efficient algorithms for choosing optimal λ parameters do exist. Examples of solution path algorithms include:

- LARS Algorithm for the LASSO (Efron et al. 2004)
- Piecewise linearity (Rosset and Zhu 2007)
- Generic path algorithm (Zhou and Wu 2013)
- Pathwise coordinate descent (Friedman et al. 2007)
- Alternating Direction Method of Multipliers (ADMM) (Boyd et al. 2011)

We will show how to visualize the relations between the regularization parameter ($\ln(\lambda)$) and the number and magnitude of the corresponding coefficients for each specific regularized regression method.

18.6.3 LASSO and Ridge Solution Paths

Figures 18.6 and 18.7 show plots of the *LASSO* results and are obtained using the R script below. Note that the top-horizontal axis labels indicate the number of non-trivial parameters in the resulting model corresponding to the $\log(\lambda)$, which is labeled on the bottom-horizontal axis.

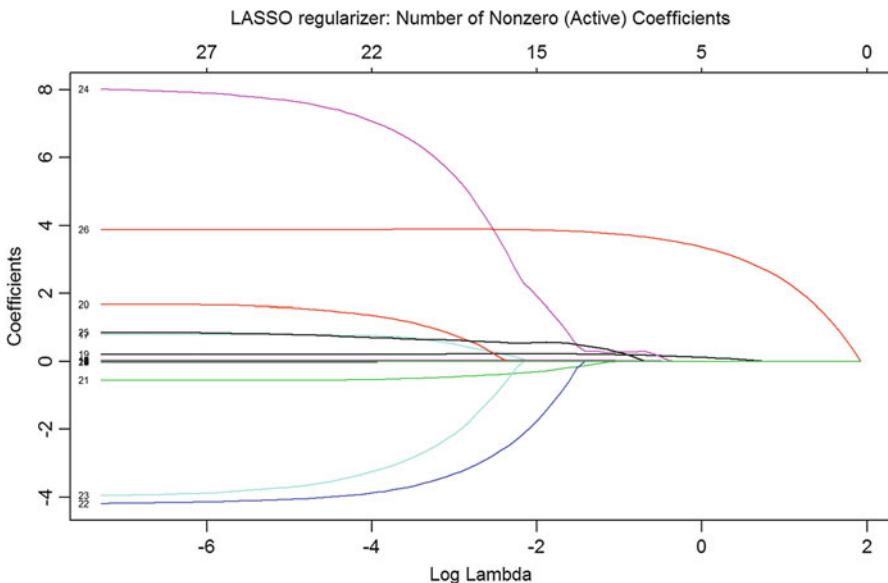


Fig. 18.6 Relations between LASSO-regularized model coefficient sizes (y-axis), magnitude of the regularization parameter (bottom axis), and the efficacy of the model selection, i.e., number of non-trivial coefficients (bottom axis)

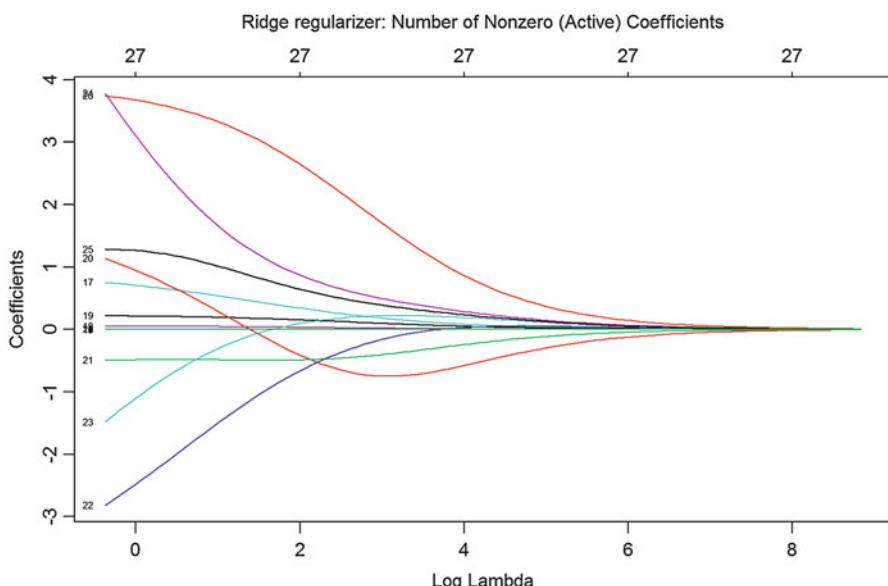


Fig. 18.7 Relations between Ridge-regularized model coefficient sizes (y-axis), magnitude of the regularization parameter (bottom axis), and the efficacy of the model selection, i.e., number of non-trivial coefficients (bottom axis)

```
### Plot Solution Path ###
# LASSO
plot(fitLASSO, xvar="Lambda", Label="TRUE")
# add label to upper x-axis
mtext("LASSO regularizer: Number of Nonzero (Active) Coefficients",
side=3, line=2.5)
```

Similarly, the plot for the *Ridge* regularization can be obtained by:

```
### Plot Solution Path ###
# Ridge
plot(fitRidge, xvar="Lambda", Label="TRUE")
# add label to upper x-axis
mtext("Ridge regularizer: Number of Nonzero (Active) Coefficients",
side=3, line=2.5)
```

Let's try to compare the paths of the *LASSO* and *Ridge* regression solutions. Below, you will see that the curves of *LASSO* are steeper and non-differentiable at some points, which is the result of using the L_1 norm. On the other hand, the *Ridge* path is smoother and asymptotically tends to 0 as λ increases.

Let's start by examining the joint objective function (including *LASSO* and *Ridge* terms):

$$\min_{\beta} \left(\sum_i (y_i - x_i \beta)^2 + \frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \right),$$

where $\|\beta\|_1 = \sum_{j=1}^p |\beta_j|$ and $\|\beta\|_2 = \sqrt{\sum_{j=1}^p \|\beta_j\|^2}$ are the norms of β corresponding to

the L_1 and L_2 distance measures, respectively. When $\alpha = 0$ and $\alpha = 1$ correspond to *Ridge* and *LASSO* regularization. The following two natural questions raise:

- What if $0 < \alpha < 1$?
- How does the regularization penalty term affect the optimal solution?

In Chap. 10, we explored the minimal SSE (Sum of Square Error) for the OLS (without penalty) where the feasible parameter (β) spans the entire real solution space. In penalized optimization problems, the best solution may actually be unachievable. Therefore, we look for solutions that are “closest”, within the feasible region, to the enigmatic best solution.

The effect of the penalty term on the objective function is separate from the *fidelity term* (OLS solution). Thus, the effect of $0 \leq \alpha \leq 1$ is limited to the **size and shape of the penalty region**. Let's try to visualize the feasible region as:

- centrosymmetric, when $\alpha = 0$, and
- super diamond, then $\alpha = 1$.

Here is a hands-on example:

```

require(needs)

# Constructing Quadratic Formula
result <- function(a,b,c){
  if(delta(a,b,c) > 0){ # first case D>0
    x_1 = (-b+sqrt(delta(a,b,c)))/(2*a)
    x_2 = (-b-sqrt(delta(a,b,c)))/(2*a)
    result = c(x_1,x_2)
  }
  else if(delta(a,b,c) == 0){ # second case D=0
    x = -b/(2*a)
  }
  else {"There are no real roots."} # third case D<0
}
# Constructing delta
delta<-function(a,b,c){
  b^2-4*a*c
}

```

To make this realistic, we will use the MLB dataset to first fit an OLS model. The dataset contains 1,034 records of *heights and weights* for some current and recent Major League Baseball (MLB) Players.

- *Height*: Player height in inches,
- *Weight*: Player weight in pounds,
- *Age*: Player age at time of record.

Then, we can obtain the SSE for any $\|\beta\|$:

$$SSE = \|Y - \hat{Y}\|^2 = (Y - \hat{Y})^T (Y - \hat{Y}) = Y^T Y - 2\beta^T X^T Y + \beta^T X^T X \beta.$$

Next, we will compute the SSE contours in several situations.

```

library("ggplot2")

# load data
mlb<- read.table('https://umich.instructure.com/files/330381/download?downlo
ad_frd=1', as.is=T, header=T)
str(mlb)

## 'data.frame': 1034 obs. of 6 variables:
## $ Name   : chr "Adam_Donachie" "Paul_Bako" "Ramon_Hernandez"
## $ Kevin_Millar" ...
## $ Team   : chr "BAL" "BAL" "BAL" ...
## $ Position: chr "Catcher" "Catcher" "Catcher" "First_Baseman" ...
## $ Height  : int 74 74 72 72 73 69 69 71 76 71 ...
## $ Weight  : int 180 215 210 210 188 176 209 200 231 180 ...
## $ Age     : num 23 34.7 30.8 35.4 35.7 ...

fit<-lm(Height~Weight+Age-1, data = as.data.frame(scale(mlb[,4:6])))
points = data.frame(x=c(0,fit$coefficients[1]),y=c(0,fit$coefficients[2]),
z=c("(0,0)", "OLS Coef"))

Y=scale(mlb$Height)
X = scale(mlb[,c(5,6)])
beta1=seq(-0.556, 1.556, length.out = 100)
beta2=seq(-0.661, 0.3386, length.out = 100)
df <- expand.grid(beta1 = beta1, beta2 = beta2)

```

```

b = as.matrix(df)
df$sse <- rep(t(Y)%%Y, 100*100)-2*b%%t(X)%%Y + diag(b%%t(X)%%X%%t(b))

base <- ggplot(df) +
  stat_contour(aes(beta1, beta2, z = sse), breaks = round(quantile(df$sse,
  seq(0, 0.2, 0.03)), 0),
  size = 0.5, color="darkorchid2", alpha=0.8) +
  scale_x_continuous(limits = c(-0.4,1)) +
  scale_y_continuous(limits = c(-0.55,0.4)) +
  coord_fixed(ratio=1) +
  geom_point(data = points, aes(x,y)) +
  geom_text(data = points, aes(x,y, label=z), vjust = 2, size=3.5) +
  geom_segment(aes(x = -0.4, y = 0, xend = 1, yend = 0), colour = "grey46",
  arrow = arrow(length=unit(0.30, "cm")), size=0.5, alpha=0.8) +
  geom_segment(aes(x = 0, y = -0.55, xend = 0, yend = 0.4), colour="grey46",
  arrow = arrow(length=unit(0.30, "cm")), size=0.5, alpha=0.8)

plot_alpha = function(alpha=0, restrict=0.2, beta1_range=0.2,
annot=c(0.15, -0.25, 0.205, -0.05)){
  a=alpha; t=restrict; k=beta1_range; pos=data.frame(V1=annot[1:4])
  tex=paste("(", as.character(annot[3]), ", ", as.character(annot[4]), ")",
  sep = ""))
  K = seq(0,k, length.out = 50)
  y = unlist(lapply((1-a)*K^2/2+a*K-t, result, a=(1-a)/2, b=a))[seq(1,99, by=2)]
  fills = data.frame(x=c(rev(-K),K), y1=c(rev(y),y), y2=c(-rev(y),-y))
  p<-base+geom_line(data=fills, aes(x = x, y = y1), colour = "salmon1",
  alpha=0.6, size=0.7) +
  geom_line(data=fills, aes(x = x, y = y2), colour = "salmon1", alpha=0.6,
  size=0.7) +
  geom_polygon(data = fills, aes(x, y1), fill = "red", alpha = 0.2) +
  geom_polygon(data = fills, aes(x, y2), fill = "red", alpha = 0.2) +
  geom_segment(data=pos, aes(x = V1[1] , y = V1[2], xend = V1[3],
  yend = V1[4]),
  arrow = arrow(length=unit(0.30, "cm")), alpha=0.8,
  colour = "magenta") +
  ggplot2::annotate("text", x = pos$V1[1]-0.01, y = pos$V1[2]-0.11,
  label = paste(tex, "\n", "Point of Contact \n i.e.,",
  Coef of", "alpha =", fractions(a)), size=3) +
  xlab(expression(beta[1])) +
  ylab(expression(beta[2])) +
  ggtitle(paste("alpha =", as.character(fractions(a)))) +
  theme(Legend.position="none")
}

# $alpha=0$ - Ridge
p1 <- plot_alpha(alpha=0, restrict=(0.21^2)/2, beta1_range=0.21,
annot=c(0.15, -0.25, 0.205, -0.05))
p1 <- p1 + ggtitle(expression(paste(alpha, "=0 (Ridge)")))
# $alpha=1/9$
p2 <- plot_alpha(alpha=1/9, restrict=0.046, beta1_range=0.22,
annot =c(0.15, -0.25, 0.212, -0.02))
p2 <- p2 + ggtitle(expression(paste(alpha, "=1/9")))
# $alpha=1/5$
p3 <- plot_alpha(alpha=1/5, restrict=0.063, beta1_range=0.22,
annot=c(0.13, -0.25, 0.22, 0))
p3 <- p3 + ggtitle(expression(paste(alpha, "=1/5")))
# $alpha=1/2$
p4 <- plot_alpha(alpha=1/2, restrict=0.123, beta1_range=0.22,
annot=c(0.12, -0.25, 0.22, 0))

```

```

p4 <- p4 + ggtitle(expression(paste(alpha, "=1/2")))
# $alpha=3/4
p5 <- plot_alpha(alpha=3/4, restrict=0.17, beta1_range=0.22,
annote=c(0.12, -0.25, 0.22, 0))
p5 <- p5 + ggtitle(expression(paste(alpha, "=3/4")))

# $alpha=1$ - LASSO
t=0.22
K = seq(0,t,Length.out = 50)
fills = data.frame(x=c(-rev(K),K),y1=c(rev(t-K),c(t-K)),
y2=c(-rev(t-K),-c(t-K)))
p6 <- base +
  geom_segment(aes(x = 0, y = t, xend = t, yend = 0), colour = "salmon1",
alpha=0.1, size=0.2) +
  geom_segment(aes(x = 0, y = t, xend = -t, yend = 0), colour = "salmon1",
alpha=0.1, size=0.2) +
  geom_segment(aes(x = 0, y = -t, xend = t, yend = 0), colour = "salmon1",
alpha=0.1, size=0.2) +
  geom_segment(aes(x = 0, y = -t, xend = -t, yend = 0), colour = "salmon1",
alpha=0.1, size=0.2) +
  geom_polygon(data = fills, aes(x, y1), fill = "red", alpha = 0.2) +
  geom_polygon(data = fills, aes(x, y2), fill = "red", alpha = 0.2) +
  geom_segment(aes(x = 0.12, y = -0.25, xend = 0.22, yend = 0),
colour = "magenta",
arrow = arrow(Length=unit(0.30, "cm")), alpha=0.8) +
ggplot2::annotate("text", x = 0.11, y = -0.36,
label = "(0.22,0)\n Point of Contact \n i.e Coef of LASSO", size=3) +
xlab( expression(beta[1])) +
ylab( expression(beta[2])) +
theme(Legend.position="none") +
ggtitle(expression(paste(alpha, "=1 (LASSO)")))

```

Then, let's add the six feasible regions corresponding to $\alpha = 0$ (Ridge), $\alpha = \frac{1}{9}$, $\alpha = \frac{1}{5}$, $\alpha = \frac{1}{2}$, $\alpha = \frac{3}{4}$ and $\alpha = 1$ (LASSO).

Figures 18.8, 18.9 and 18.10 provide some intuition into the continuum from Ridge to LASSO regularization. The feasible regions are drawn as ellipse contours of the SSE in red. Curves around the corresponding feasible regions represent the boundary of the constraint function $\frac{1-\alpha}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1 \leq t$.

In this example, β_2 shrinks to 0 for $\alpha = \frac{1}{5}$, $\alpha = \frac{1}{2}$, $\alpha = \frac{3}{4}$ and $\alpha = 1$.

We observe that it is almost impossible for the contours of Ridge regression to touch the circle at any of the coordinate axes. This is also true in higher dimensions (nD), where the L_1 and L_2 metrics are unchanged and the 2D ellipse representations of the feasibility regions become hyper-ellipsoidal shapes.

Generally, as α goes from 0 to 1, the coefficients of more features tend to shrink towards 0. This specific property makes LASSO useful for variable selection.

Let's compare the feasibility regions corresponding to *Ridge* (top, $p1$) and *LASSO* (bottom, $p6$) regularization.

```
plot(p1)
```

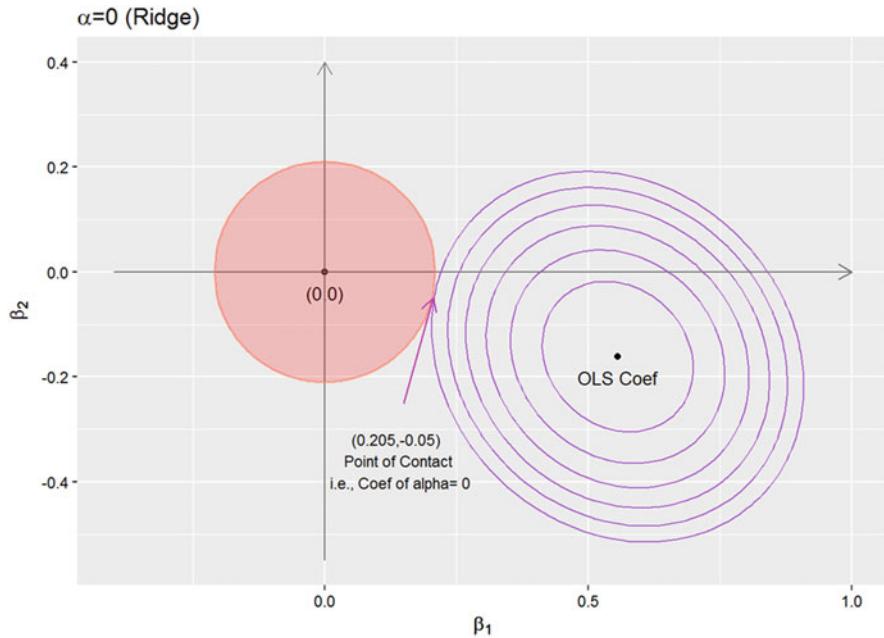


Fig. 18.8 Ridge-regularization SSE contour and penalty region

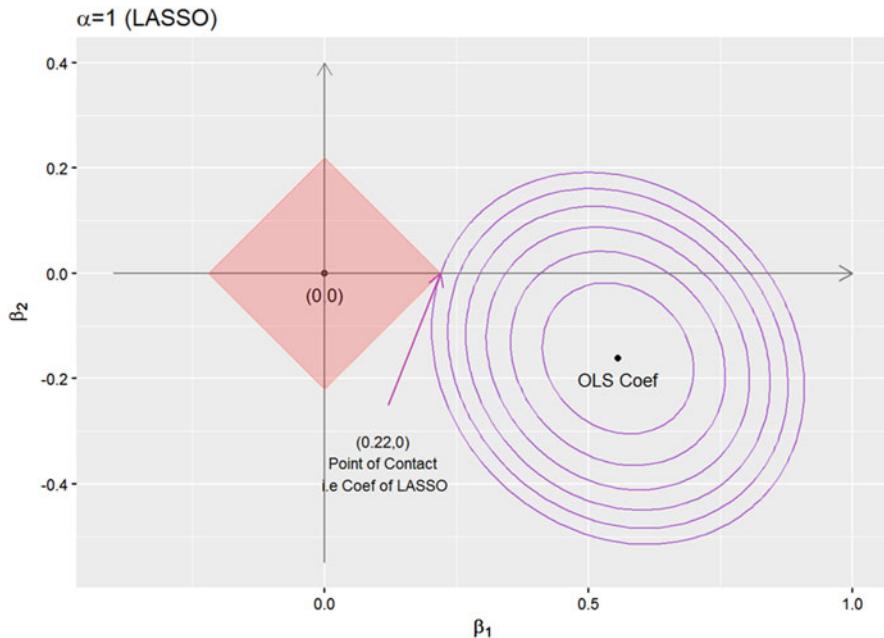


Fig. 18.9 LASSO-regularization SSE contour and penalty region

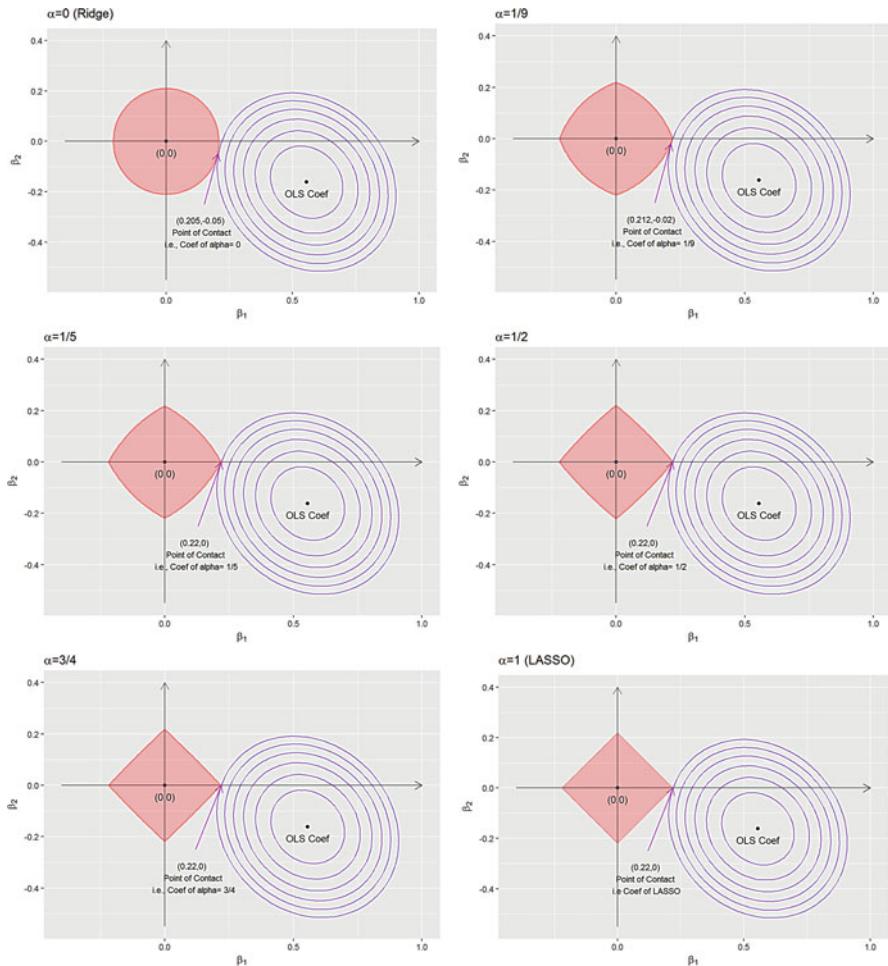


Fig. 18.10 SSE contour and penalty region for six continuous values of the alpha parameter illustrating the smooth transition from Ridge ($\alpha = 0$) to LASSO ($\alpha = 1$) regularization

plot(p6)

Then, we can plot the *progression* from Ridge to LASSO. This composite *plot* is *intense* and may take several minutes to render, Fig. 18.10! Finally, Fig. 18.11 depicts the MSE of the cross-validated LASSO-regularized model against the magnitude of number of non-trivial coefficients (top axis). The dashed vertical lines suggest an optimal range [3:9] for number of features to include in the model.

```
library("gridExtra")
grid.arrange(p1, p2, p3, p4, p5, p6, nrow=3)
```

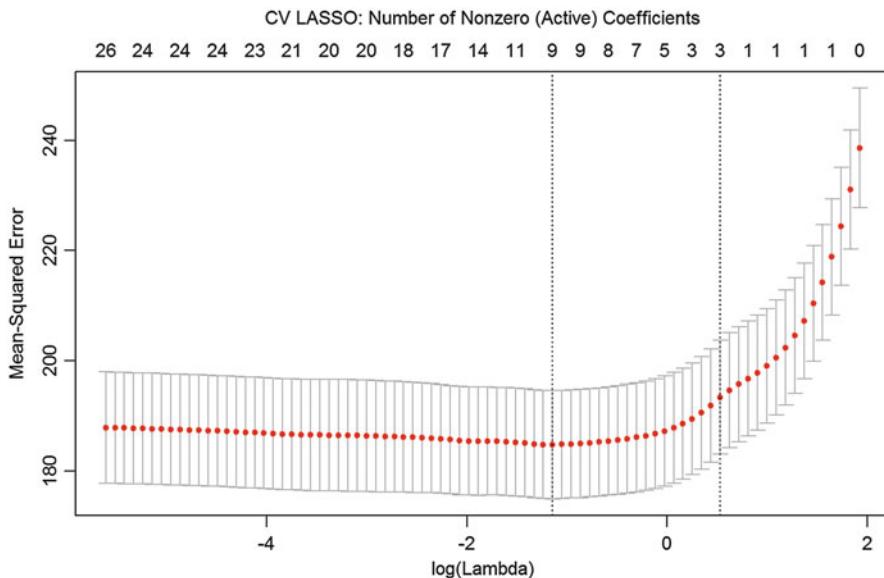


Fig. 18.11 MSE of the cross-validated LASSO-regularized model against the magnitude of the regularization parameter (bottom axis), and the efficacy of the model selection, i.e., number of non-trivial coefficients (top axis). The dashed vertical lines suggest an optimal range for the penalty term and the number of features

18.6.4 *Choice of the Regularization Parameter*

Efficiently obtaining the entire solution path is nice, but we still have to choose a specific λ regularization parameter. This is critical as λ controls the bias-variance tradeoff. Traditional model selection methods rely on various metrics like Mallows' C_p , AIC, BIC, and adjusted R^2 .

Internal statistical validation (Cross validation) is a popular modern alternative, which offers some of these benefits:

- Choice is based on predictive performance,
- Makes fewer model assumptions,
- More widely applicable.

18.6.5 Cross Validation Motivation

Ideally, we would like a separate validation set for choosing λ for a given method. Reusing training sets may encourage overfitting and using testing data to pick λ may underestimate the true error rate. Often, when we do not have enough data for a separate validation set, cross-validation provides an alternative strategy.

18.6.6 n-Fold Cross Validation

We have already seen examples of using cross-validation, e.g., Chap. 14, and Chap. 21 provides additional details about this internal statistical assessment strategy.

We can use either automated or manual cross-validation. In either case, the protocol involves the following iterative steps:

1. Randomly split the training data into n parts (“folds”).
2. Fit a model using data in $n - 1$ folds for multiple λ s.
3. Calculate some prediction quality metrics (e.g., MSE, accuracy) on the last remaining fold, see Chap. 14.
4. Repeat the process and average the prediction metrics across iterations.

Common choices of n are 5, 10, and n (which corresponds to leave-one-out CV). One standard error rule is to choose λ corresponding to the smallest model with MSE within one standard error of the minimum MSE.

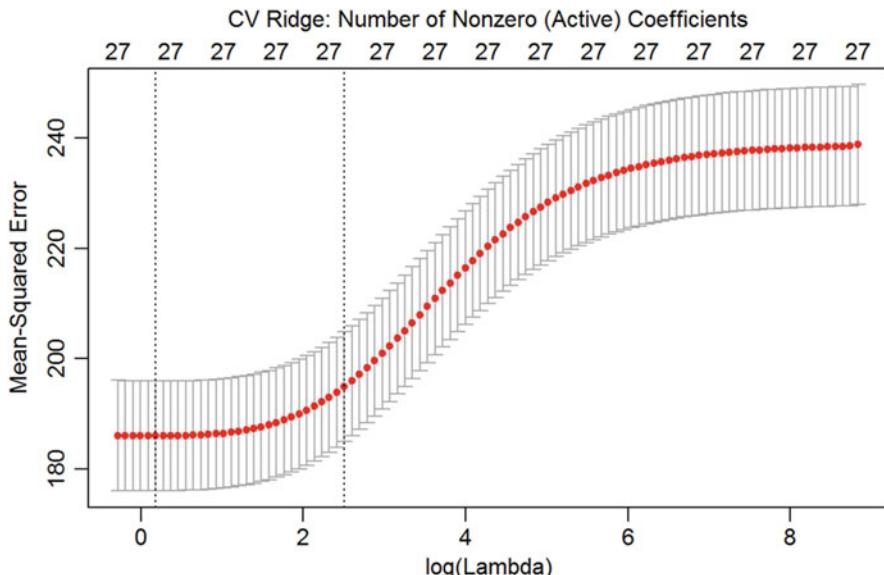


Fig. 18.12 Ridge-regularization, similar to Fig. 18.11

18.6.7 LASSO 10-Fold Cross Validation

Now, let's apply an internal statistical cross-validation to assess the quality of the LASSO and Ridge models, based on our Parkinson's disease case-study. Recall our split of the PD data into training (yTrain, XTrain) and testing (yTest, XTest) sets (Fig. 18.12).

```
##### 10-fold cross validation #####
# LASSO
library("glmnet")
set.seed(seed) # set seed
# (10-fold) cross validation for the LASSO
cvLASSO = cv.glmnet(XTrain, yTrain, alpha = 1)
plot(cvLASSO)
mtext("CV LASSO: Number of Nonzero (Active) Coefficients", side=3, line=2.5)

# Report MSE LASSO
predLASSO <- predict(cvLASSO, s = cvLASSO$Lambda.1se, newx = XTest)
testMSE_LASSO <- mean((predLASSO - yTest)^2); testMSE_LASSO
## [1] 200.5609

##### 10-fold cross validation #####
# Ridge Regression
set.seed(seed) # set seed
# (10-fold) cross validation for Ridge Regression
cvRidge = cv.glmnet(XTrain, yTrain, alpha = 0)
plot(cvRidge)
mtext("CV Ridge: Number of Nonzero (Active) Coefficients", side=3, line=2.5)

# Report MSE Ridge
predRidge <- predict(cvRidge, s = cvRidge$Lambda.1se, newx = XTest)
testMSE_Ridge <- mean((predRidge - yTest)^2); testMSE_Ridge
## [1] 195.7406
```

Note that the `predict()` method, applied to `cv.glmnet` or `glmnet` forecasting models, is effectively a function wrapper to `predict.glmnet()`. According to what you would like to get as a **prediction output**, you can use `type="..."` to specify one of the following types of prediction outputs:

- `type = "link"`, reports the linear predictors for “binomial”, “multinomial”, “poisson” or “cox” models; for “gaussian” models it gives the fitted values.
- `type = "response"`, reports the fitted probabilities for “binomial” or “multinomial”, fitted mean for “poisson” and the fitted relative-risk for “cox”; for “gaussian” type “response” is equivalent to type “link”.
- `type = "coefficients"`, reports the coefficients at the requested values for `s`. Note that for “binomial” models, results are returned only for the class corresponding to the second level of the factor response.

- type = "class", applies only to "binomial" or "multinomial" models, and produces the class label corresponding to the maximum probability.
- type = "nonzero", returns a list of the indices of the nonzero coefficients for each value of `s`.

18.6.8 Stepwise OLS (Ordinary Least Squares)

For a fair comparison, let's also obtain an OLS stepwise model selection, see Chap. 17.

```
dt = as.data.frame(cbind(yTrain, XTrain))
ols_step <- lm(yTrain ~., data = dt)
ols_step <- step(ols_step, direction = 'both', k=2, trace = F)
summary(ols_step)

## Call:
## lm(formula = yTrain ~ L_cingulate_gyrus_ComputeArea +
##     R_cingulate_gyrus_Volume +
##     L_caudate_Volume + L_putamen_ComputeArea + L_putamen_Volume +
##     R_putamen_ComputeArea + Weight + Age + chr17_rs11012_GT +
##     chr17_rs393152_GT + chr17_rs12185268_GT + UPDRS_part_I, data = dt)
##
## Residuals:
##   Min     1Q Median     3Q    Max
## -29.990 -9.098 -0.310  8.373 49.027
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                 -2.8179771  4.5458868 -0.620  0.53548    
## L_cingulate_gyrus_ComputeArea 0.0045203  0.0013422  3.368  0.00079 ***  
## R_cingulate_gyrus_Volume     -0.0010036  0.0003461 -2.900  0.00382 **   
## L_caudate_Volume              -0.0021999  0.0011054 -1.990  0.04686 *    
## L_putamen_ComputeArea        -0.0087295  0.0045925 -1.901  0.05764 .    
## L_putamen_Volume              0.0035419  0.0017969  1.971  0.04902 *    
## R_putamen_ComputeArea        0.0029862  0.0019036  1.569  0.11706    
## Weight                       0.0424646  0.0268088  1.584  0.11355    
## Age                          0.2198283  0.0522490  4.207 2.84e-05 ***  
## chr17_rs11012_GT             -4.2408237  1.8122682 -2.340  0.01950 *    
## chr17_rs393152_GT             -3.5818432  2.2619779 -1.584  0.11365    
## chr17_rs12185268_GT           8.2990131  2.7356037  3.034  0.00248 **  
## UPDRS_part_I                  3.8780897  0.2541024 15.262 < 2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.41 on 911 degrees of freedom
## Multiple R-squared:  0.2556, Adjusted R-squared:  0.2457 
## F-statistic: 26.06 on 12 and 911 DF,  p-value: < 2.2e-16
```

We use direction=both for both *forward* and *backward* selection and choose the optimal one. k=2 specifies AIC and BIC criteria, and you can choose $k \sim \log(n)$.

Then, we use the `ols_step` model to predict the outcome Y for some new test data.

```
betaHatOLS_step = ols_step$coefficients
var_step <- colnames(ols_step$model)[-1]
XTestOLS_step = cbind(rep(1, nrow(XTest)), XTest[,var_step])
predOLS_step = XTestOLS_step%*%betaHatOLS_step
testMSEOLS_step = mean((predOLS_step - yTest)^2)
# Report MSE OLS Stepwise feature selection
testMSEOLS_step
## [1] 186.3043
```

Alternatively, we can predict the outcomes directly using the `predict()` function, and the results should be identical:

```
pred2 <- predict(ols_step, as.data.frame(XTest))
any(pred2 == predOLS_step)
## [1] TRUE
```

18.6.9 Final Models

Let's identify the most important (predictive) features, which can then be interpreted in the context of the specific data.

```
# Determine final models
# Extract Coefficients
# OLS coefficient estimates
betaHatOLS = fitOLS$coefficients
# LASSO coefficient estimates
betaHatLASSO = as.double(coef(fitLASSO, s = cvLASSO$Lambda.1se)) # s is lam
bda
# Ridge coefficient estimates
betaHatRidge = as.double(coef(fitRidge, s = cvRidge$Lambda.1se))

# Test Set MSE
# calculate predicted values

XTestOLS = cbind(rep(1, nrow(XTest)), XTest) # add intercept to test data
predOLS = XTestOLS%*%betaHatOLS
predLASSO = predict(fitLASSO, s = cvLASSO$Lambda.1se, newx = XTest)
predRidge = predict(fitRidge, s = cvRidge$Lambda.1se, newx = XTest)

# calculate test set MSE
testMSEOLS = mean((predOLS - yTest)^2)
testMSELASSO = mean((predLASSO - yTest)^2)
testMSERidge = mean((predRidge - yTest)^2)
```

Figure 18.13 shows a rank-ordered list of the key predictors of the clinical outcome variable (total UPDRS, $y <- \text{data1\$UPDRS_part_I} + \text{data1\$UPDRS_part_II} + \text{data1\$UPDRS_part_III}$).

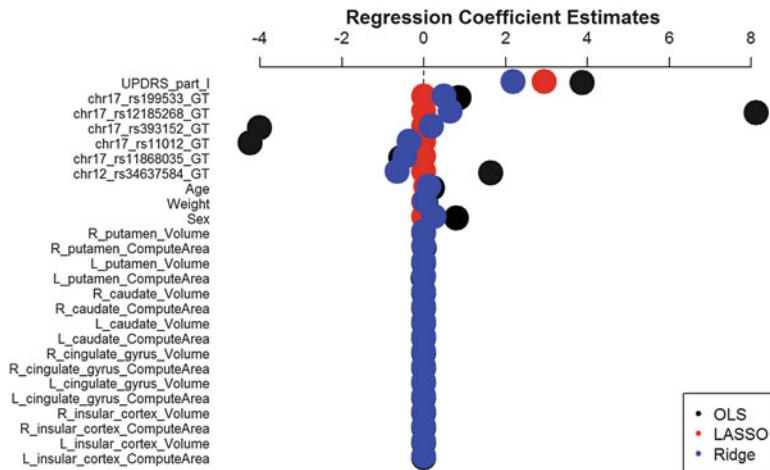


Fig. 18.13 Variables importance plot for the three alternative models

```

# Plot Regression Coefficients
# create variable names for plotting
library("arm")
par(mar=c(2, 13, 1, 1))  # extra large left margin
varNames <- colnames(data1[, !(names(data1) %in% drop_features)])

varNames; Length(varNames)

## [1] "L_insular_cortex_ComputeArea"  "L_insular_cortex_Volume"
## [3] "R_insular_cortex_ComputeArea"  "R_insular_cortex_Volume"
## [5] "L_cingulate_gyrus_ComputeArea" "L_cingulate_gyrus_Volume"
## [7] "R_cingulate_gyrus_ComputeArea" "R_cingulate_gyrus_Volume"
## [9] "L_caudate_ComputeArea"         "L_caudate_Volume"
## [11] "R_caudate_ComputeArea"         "R_caudate_Volume"
## [13] "L_putamen_ComputeArea"        "L_putamen_Volume"
## [15] "R_putamen_ComputeArea"        "R_putamen_Volume"
## [17] "Sex"                          "Weight"
## [19] "Age"                          "chr12_rs34637584_GT"
## [21] "chr17_rs11868035_GT"         "chr17_rs11012_GT"
## [23] "chr17_rs393152_GT"           "chr17_rs12185268_GT"
## [25] "chr17_rs199533_GT"           "UPDRS_part_I"
## [27] "time_visit"

## [1] 27

# Graph 27 regression coefficients (exclude intercept [1], betaHat
indices 2:27)
coefplot(betaHatOLS[2:27], sd = rep(0, 26), cex.pts = 5,
main = "Regression Coefficient Estimates", varnames = varNames)
coefplot(betaHatLASSO[2:27], sd = rep(0, 26), add = TRUE, col.pts = "red",
cex.pts = 5)
coefplot(betaHatRidge[2:27], sd = rep(0, 26), add = TRUE, col.pts = "blue",
cex.pts = 5)
Legend("bottomright", c("OLS", "LASSO", "Ridge"), col = c("black", "red",
"blue"), pch = c(20, 20, 20), bty = "o")

```

Table 18.2 Testing data MSE

OLS	OLS_step	LASSO	Ridge
183.3239	186.3043	200.5609	195.7406

18.6.10 Model Performance

Table 18.2 quantifies the performance of the four models.

```
# Test Set MSE Table
# create table as data frame
MSETable = data.frame(OLS=testMSEOLS, OLS_step=testMSEOLS_step,
LASSO=testMSELASSO, Ridge=testMSERidge)

# convert to markdown
kable(MSETable, format="pandoc", caption="Test Set MSE", align=c("c", "c",
"c", "c"))
```

18.6.11 Comparing Selected Features

```
var_step = names(ols_step$coefficients)[-1]
var_Lasso=colnames(XTrain)[which(coef(fitLASSO,s=cvLASSO$Lambda.min)!=0)-1]

intersect(var_step,var_Lasso)

## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Weight"                      "Age"
## [5] "chr17_rs12185268_GT"          "UPDRS_part_I"

coef(fitLASSO, s = cvLASSO$Lambda.min)
## 28 x 1 sparse Matrix of class "dgCMatrix"
##                               1
## (Intercept)           1.7142107049
## L_insular_cortex_ComputeArea .
## L_insular_cortex_Volume   .
## R_insular_cortex_ComputeArea .
## R_insular_cortex_Volume   .
## L_cingulate_gyrus_ComputeArea 0.0003399436
## L_cingulate_gyrus_Volume    0.0002099980
## R_cingulate_gyrus_ComputeArea .
## R_cingulate_gyrus_Volume    .
## L_caudate_ComputeArea      .
## L_caudate_Volume          .
## R_caudate_ComputeArea      .
## R_caudate_Volume          .
## L_putamen_ComputeArea     .
## L_putamen_Volume          .
## R_putamen_ComputeArea     0.0010417502
## R_putamen_Volume          .
## Sex                         .
## Weight                      0.0336216322
## Age                         0.2097678904
## chr12_rs34637584_GT       .
## chr17_rs11868035_GT       -0.0094055047
## chr17_rs11012_GT           .
## chr17_rs393152_GT           .
## chr17_rs12185268_GT       0.2688574886
## chr17_rs199533_GT           .
## UPDRS_part_I                3.7697168303
## time_visit                  .
```

Stepwise variable selection for OLS selects 12 variables, whereas LASSO selects 9 variables with the best λ . There are 6 common variables identified as salient features by both OLS and LASSO.

18.6.12 Summary

Traditional linear models are useful but also have their shortcomings:

- Prediction accuracy may be sub-optimal.
- Model interpretability may be challenging (especially when a large number of features are used as regressors).
- Stepwise model selection may improve the model performance and add some interpretations, but still may not be optimal.

Regularization adds a penalty term to the estimation:

- Enables exploitation of the *bias-variance* tradeoff.
- Provides flexibility on specifying penalties to allow for continuous variable selection.
- Allows incorporation of prior knowledge.

18.7 Knock-off Filtering: Simulated Example

Variable selection that controls the false discovery rate (FDR) of *salient features* can be accomplished in different ways. Knockoff filtering represents one strategy for controlled variable selection. To show the usage of `knockoff.filter` we start with a synthetic dataset constructed so that the true coefficient vector β has only a few nonzero entries.

The essence of knockoff filtering is based on the following three-step process:

- Construct the decoy features (knockoff variables), one for each real observed feature. These act as controls for assessing the importance of the real variables.
- For each feature, X_i , compute the knockoff statistic, W_j , which measures the importance of the variable, relative to its decoy counterpart, \tilde{X}_i .
- Determine the overall knockoff threshold. This is computed by rank-ordering the W_j statistics (from large to small), walking down the list of W_j 's, selecting variables X_j corresponding to positive W_j 's, and terminating this search the last time the ratio of negative to positive W_j 's is below the default FDR q value, e.g., $q = 0.10$.

Mathematically, we consider X_j to be *unimportant* (i.e., peripheral or extraneous) if the conditional distribution of Y given X_1, \dots, X_p does not depend on X_j . Formally, X_j is unimportant if it is conditionally independent of Y given all other features, X_{-j} :

$$Y \perp X_j \mid X_{-j}.$$

We want to generate a Markov Blanket of Y , such that the smallest set of features J satisfies this condition. Further, to make sure we do not make too many mistakes, we search for a set \hat{S} controlling the false discovery rate (FDR):

$$FDR(\hat{S}) = E\left(\frac{\#\{j \in \hat{S} : x_j \text{ unimportant}\}}{\#\{j \in \hat{S}\}}\right) \leq q \text{ (e.g. 10\%).}$$

Let's look at one simulation example.

```
# Problem parameters
n = 1000          # number of observations
p = 300           # number of variables
k = 30            # number of variables with nonzero coefficients
amplitude = 3.5  # signal amplitude (for noise level = 1)

# Problem data
X = matrix(rnorm(n*p), nrow=n, ncol=p)
nonzero = sample(p, k)
beta = amplitude * (1:p %in% nonzero)
y.sample <- function() X %*% beta + rnorm(n)
```

To begin with, we will invoke the `knockoff.filter` using the default settings.

```
# install.packages("knockoff")
library(knockoff)
y = y.sample()
result = knockoff.filter(X, y)
print(result)

## Call:
## knockoff.filter(X = X, y = y)
##
## Selected variables:
##  [1] 6 29 30 42 52 54 63 68 70 83 88 96 102 113 115 135 138
## [18] 139 167 176 179 194 212 220 225 228 241 248 265 273 287 288 295
```

The false discovery proportion (fdp) is:

```
fdp <- function(selected) sum(beta[selected] == 0) / max(1, length(selected))
)
fdp(result$selected)
## [1] 0.09090909
```

This yields an approximate FDR of 0.10.

The default settings of the knockoff filter use a test statistic based on LASSO -- `knockoff.stat.lasso_signed_max`, which computes the W_j statistics that quantify the discrepancy between a real (X_j) and a decoy, knockoff (\tilde{X}_j), feature:

$$W_j = \max(X_j, \tilde{X}_j) \times \text{sgn}(X_j - \tilde{X}_j).$$

Effectively, the W_j statistics measure how much more important the variable X_j is relative to its decoy counterpart \tilde{X}_j . The strength of the importance of X_j relative to \tilde{X}_j is measured by the magnitude of W_j .

The `knockoff` package includes several other test statistics, with appropriate names prefixed by `knockoff.stat`. For instance, we can use a statistic based on forward selection (`fs`) and a lower target FDR of 0.10.

```
result = knockoff.filter(X, y, fdr = 0.10, statistic = knockoff.stat.fs)
fdp(result$selected)
## [1] 0.1428571
```

One can also define additional test statistics, complementing the ones included in the package already. For instance, if we want to implement the following test-statistics:

$$W_j = \|X_j^t \cdot y\| - \|\tilde{X}^t \cdot y\|.$$

We can code it as:

```
new_knockoff_stat <- function(X, X_ko, y) {
  abs(t(X) %*% y) - abs(t(X_ko) %*% y)
}
result = knockoff.filter(X, y, statistic = new_knockoff_stat)
fdp(result$selected)
## [1] 0.3333333
```

18.7.1 Notes

The `knockoff.filter` function is a wrapper around several simpler functions that (1) construct knockoff variables (`knockoff.create`); (2) compute the test statistic W (various functions with prefix `knockoff.stat`); and (3) compute the threshold for variable selection (`knockoff.threshold`).

The high-level function `knockoff.filter` will automatically normalize the columns of the input matrix (unless this behavior is explicitly disabled). However,

all other functions in this package assume that the columns of the input matrix have unitary Euclidean norm.

18.8 PD Neuroimaging-Genetics Case-Study

Let's illustrate controlled variable selection via knockoff filtering using the real PD dataset.

The goal is to determine which imaging, genetics and phenotypic covariates are associated with the clinical diagnosis of PD. The dataset is publicly available online.

18.8.1 Fetching, Cleaning and Preparing the Data

The data set consists of clinical, genetics, and demographic measurements. To evaluate our results, we will compare diagnostic predictions created by the model for the *UPDRS scores* and the *ResearchGroup* factor variable. First, we download the data and read it into data frames.

```
data1 <- read.table('https://umich.instructure.com/files/330397/download?download_frd=1', sep=",", header=T)
# we will deal with missing values using multiple imputation later. For now,
let's just ignore incomplete cases
data1.completeRowIndexes <- complete.cases(data1)

# table(data1.completeRowIndexes)
prop.table(table(data1.completeRowIndexes))

## data1.completeRowIndexes
##      FALSE      TRUE
## 0.3452381 0.6547619

# attach(data1)
# View(data1[data1.completeRowIndexes, ])
data2 <- data1[data1.completeRowIndexes, ]
Dx_Label <- data2$ResearchGroup; table(Dx_Label)

## Dx_Label
## Control      PD      SWEDD
##      121      897      137
```

We now construct the design matrix X and the response vector Y . The features (columns of X) represent covariates that will be used to explain the response Y .

```

# Construct preliminary design matrix.
# Define response and predictors
Y <- data1$UPDRS_part_I + data1$UPDRS_part_II + data1$UPDRS_part_III
table(Y) # Show Clinically relevant classification

## Y
### 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
### 54 20 25 12 8 7 11 16 16 9 21 16 13 13 22 25 21 31 25 29 29 28 20 25 25
### 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
### 26 35 41 23 34 32 31 37 34 28 36 29 27 22 19 17 18 18 19 16 9 10 12 9 11
### 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 66 68 69 71 75 80 81 82
### 7 10 11 5 7 4 1 5 9 4 3 2 1 6 1 2 1 2 1 1 2 3 1

Y <- Y[data1.completeRowIndexes]

# X = scale(ncaaData[, -20]) # Explicit Scaling is not needed, as
# glmnet auto standardizes predictors
# X = as.matrix(data1[, c("R_caudate_Volume", "R_putamen_Volume", "Weight",
# "Age", "chr17_rs12185268_GT")]) # X needs to be a matrix, not a data frame
drop_features <- c("FID_IID", "ResearchGroup", "UPDRS_part_I",
"UPDRS_part_II", "UPDRS_part_III")
X <- data1[, !(names(data1) %in% drop_features)]
X = as.matrix(X) # remove columns: index, ResearchGroup, and
y=(PDERS_part_I + UPDRS_part_II + UPDRS_part_III)
X <- X[data1.completeRowIndexes, ]; dim(X)

## [1] 1155 26

summary(X)

##  L_insular_cortex_ComputeArea L_insular_cortex_Volume
##  Min.   : 50.03               Min.   : 22.63
##  1st Qu.:2174.57              1st Qu.: 5867.23
##  Median :2522.52              Median : 7362.90
##  Mean   :2306.89              Mean   : 6710.18
##  3rd Qu.:2752.17              3rd Qu.: 8483.80
##  Max.   :3650.81              Max.   :13499.92
...
##  chr17_rs393152_GT chr17_rs12185268_GT chr17_rs199533_GT  time_visit
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.00
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 9.00
##  Median :0.0000   Median :0.0000   Median :0.0000   Median :24.00
##  Mean   :0.4468   Mean   :0.4268   Mean   :0.4052   Mean   :23.83
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:36.00
##  Max.   :2.0000   Max.   :2.0000   Max.   :2.0000   Max.   :54.00

mode(X) <- 'numeric'

Dx_Label <- Dx_Label[data1.completeRowIndexes]; Length(Dx_Label)

## [1] 1155

```

18.8.2 Preparing the Response Vector

The knockoff filter is designed to control the FDR under Gaussian noise. A quick inspection of the response vector shows that it is highly non-Gaussian (Figs. 18.14 and 18.5).

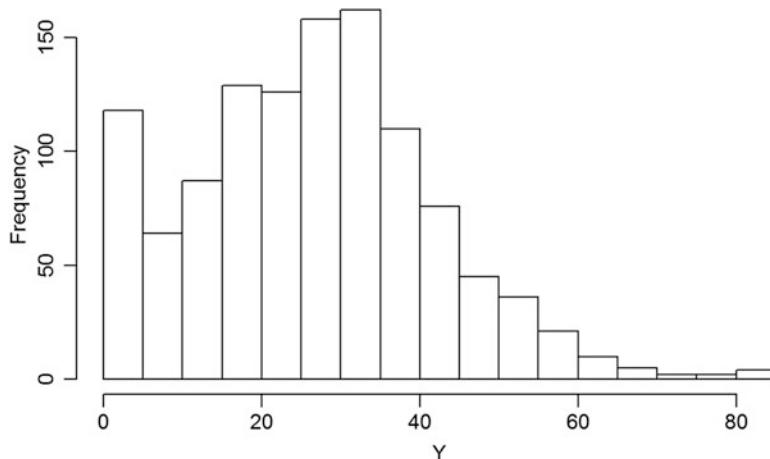


Fig. 18.14 Histogram of the outcome clinical diagnostic variable (Y) for the Parkinson's disease case-study

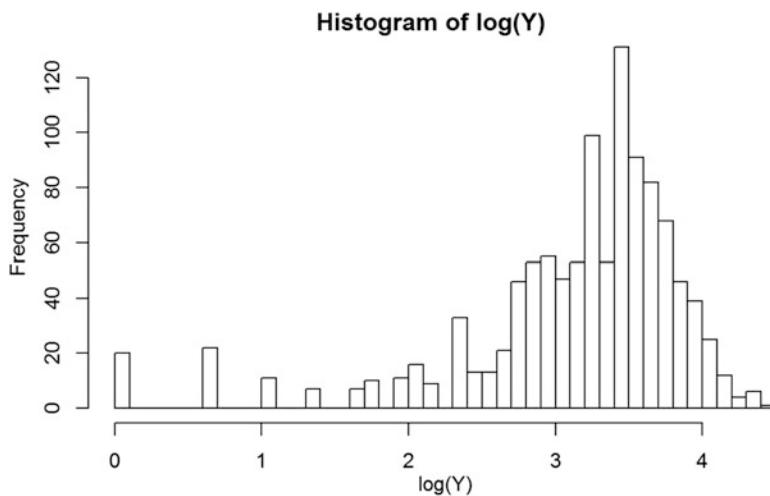


Fig. 18.15 Log-transformed histogram of the outcome clinical diagnostic variable (Y)

```
hist(Y, breaks='FD')
```

A log-transform may help to stabilize the clinical response measurements:

```
hist(log(Y), breaks='FD')
```

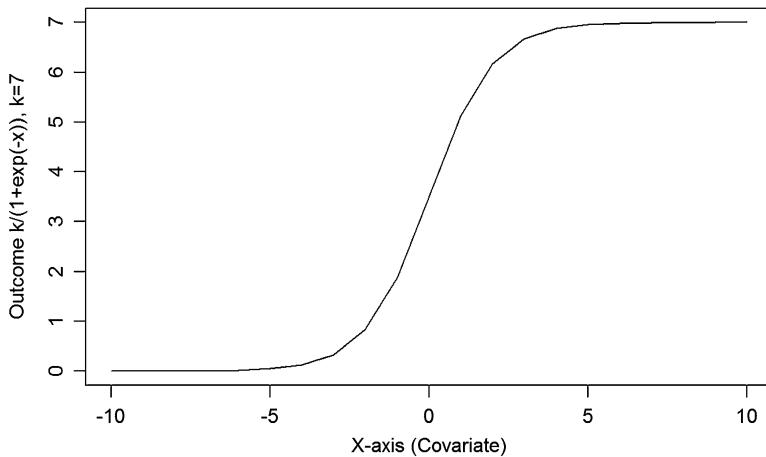


Fig. 18.16 Logistic curve transforming a continuous variable into a probability value

For **binary outcome variables**, or **ordinal categorical variables**, we can employ the logistic curve to transform the polytomous outcomes into real values (Fig. 18.16).

The Logistic curve is $y = f(x) = \frac{1}{1+e^{-x}}$, where y and x represent probability and quantitative-predictor values, respectively. A slightly more general form is: $y = f(x) = \frac{K}{1+e^{-x}}$, where the covariate $x \in (-\infty, \infty)$ and the response $y \in [0, K]$. For example,

```
library("ggplot2")
k=7
x <- seq(-10, 10, 1)
plot(x, k/(1+exp(-x)), xlab="X-axis (Covariate)", ylab="Outcome k/(1+exp(-x))",
, k=7", type="l")
```

The point of this logistic transformation is that:

$$y = \frac{1}{1 + e^{-x}} \Leftrightarrow x = \ln \frac{y}{1 - y},$$

which represents the **log-odds** (where y is the probability of an event of interest)!

We use the logistic regression equation model to estimate the probability of specific outcomes: (Estimate of) $P(Y = 1|x_1, x_2, \dots, x_l) = \frac{1}{1 + e^{-(a_0 + \sum_{k=1}^l a_k x_k)}}$, where the coefficients a_0 (intercept) and effects a_k , $k = 1, 2, \dots, l$, are estimated using GLM according to a maximum likelihood approach. Using this model allows us to estimate the probability of the dependent (clinical outcome) variable $Y = 1$ (CO), i.e., surviving surgery, given the observed values of the predictors X_k , $k = 1, 2, \dots, l$.

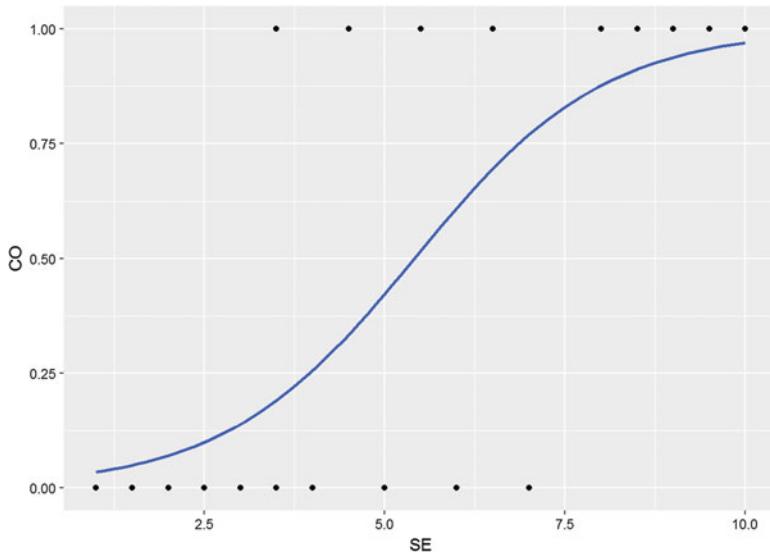


Fig. 18.17 Estimate of the logistic function for the clinical outcome (CO) probability based on the surgeon's experience (SE)

Table 18.3 Survival outcomes of a hypothetical surgical transplant treatment based on surgeon's experience

Surgeon's experience (SE)	1	1.5	2	2.5	3	3.5	3.5	4	4.5	5	5.5	6	6.5	7	8	8.5	9	9.5	10	10
Clinical outcome (CO)	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	

Probability of surviving a heart transplant based on surgeon's experience. A group of 20 patients undergo heart transplantation with different surgeons having experience in the range {0(least), 2, ..., 10(most)}, representing 100's of operating/surgery hours. How does the surgeon's experience affect the probability of the patient survival?

The data below shows the outcome of a surgery (1 = survival) or (0 = death) according to the surgeons' experience in 100's of hours of practice (Fig. 18.17 and Table 18.3).

```
mydata <- read.csv("https://umich.instructure.com/files/405273/download?download_frd=1") # 01_HeartSurgerySurvivalData.csv
# estimates a logistic regression model for the clinical outcome (CO),
# survival, using the glm
# (generalized linear model) function.
# convert Surgeon's Experience (SE) to a factor to indicate it should be
# treated as a categorical variable.
# mydata$rank <- factor(mydata$SE)
mylogit <- glm(CO ~ SE, data = mydata, family = "binomial")

# library(ggplot2)
ggplot(mydata, aes(x=SE, y=CO)) + geom_point() +
  stat_smooth(method="glm",method.args=list(family="binomial"),se=FALSE)
```

Graph of a logistic regression curve showing probability of surviving the surgery versus surgeon's experience, Fig. 18.17.

The graph shows the probability of the clinical outcome, survival, (Y-axis) versus the surgeon's experience (X-axis), with the logistic regression curve fitted to the data.

```
myLogit <- glm(CO ~ SE, data = mydata, family = "binomial")
summary(myLogit)

## Call:
## glm(formula = CO ~ SE, family = "binomial", data = mydata)
##
## Deviance Residuals:
##      Min      1Q  Median      3Q      Max
## -1.7131  -0.5719  -0.0085   0.4493   1.8220
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -4.1030     1.7629  -2.327   0.0199 *
## SE           0.7583     0.3139   2.416   0.0157 *
## ---
## Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 27.726 on 19 degrees of freedom
## Residual deviance: 16.092 on 18 degrees of freedom
## AIC: 20.092
##
## Number of Fisher Scoring iterations: 5
```

The output indicates that surgeon's experience (SE) is significantly associated with the probability of surviving the surgery (0.0157, Wald test). The output also provides the coefficients for:

- Intercept = -4.1030 , and
- SE = 0.7583 .

These coefficients can then be used in the logistic regression equation model to estimate the probability of surviving the heart surgery:

$$\text{Probability of surviving heart surgery } CO = \frac{1}{1+exp(-(-4.1030+0.7583\times SE))}.$$

For example, for a patient who is operated on by a surgeon with 200 h of operating experience (SE = 2), we plug in the value 2 in the equation to get an estimated probability of survival, $p = 0.07$:

```
SE=2
CO =1/(1+exp(-(-4.1030+0.7583*SE)))
CO
## [1] 0.07001884
```

Similarly, a patient undergoing heart surgery with a doctor who has 400 operating hours experience (SE = 4), the estimated probability of survival is $p = 0.26$:

```
SE=4; CO =1/(1+exp(-(-4.1030+0.7583*SE))); CO
## [1] 0.2554411
CO
## [1] 0.2554411
for (SE in c(1:5)) {
  CO <- 1/(1+exp(-(-4.1030+0.7583*SE)));
  print(c(SE, CO))
}
## [1] 1.0000000 0.03406915
## [1] 2.0000000 0.07001884
## [1] 3.0000000 0.1384648
## [1] 4.0000000 0.2554411
## [1] 5.0000000 0.4227486
```

[1] 0.2554411

The table below shows the probability of surviving surgery for several values of surgeons' experience (Table. 18.4).

The output from the logistic regression analysis gives a p-value of $p = 0.0157$, which is based on the Wald z-score. In addition to the Wald method, we can calculate the p-value for logistic regression using the Likelihood Ratio Test (LRT), which for these data yields 0.0006476922 (Table 18.5).

Table 18.4 Estimates of the likelihood of transplant surgery patient survival based on SE

Surgeon's experience (SE)	Probability of patient survival (Clinical outcome)
1	0.034
2	0.07
3	0.14
4	0.26
5	0.423

Table 18.5 Estimates of the effect-size, standard error and p-value quantifying the significance of SE on CO

.	Estimate	Std. error	z value	Pr(>z) Wald
SE	0.7583	0.3139	2.416	0.0157 *

```

mylogit <- glm(CO ~ SE, data = mydata, family = "binomial")
summary(mylogit)

## Call:
## glm(formula = CO ~ SE, family = "binomial", data = mydata)
##
## Deviance Residuals:
##       Min      1Q  Median      3Q      Max
## -1.7131 -0.5719 -0.0085  0.4493  1.8220
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -4.1030    1.7629 -2.327   0.0199 *  
## SE          0.7583    0.3139  2.416   0.0157 *  
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 27.726 on 19 degrees of freedom
## Residual deviance: 16.092 on 18 degrees of freedom
## AIC: 20.092
## Number of Fisher Scoring iterations: 5

```

The *logit* of a number $0 \leq p \leq 1$ is given by the formula: $\text{logit}(p) = \log \frac{p}{1-p}$, and represents the log-odds ratio (of survival in this case) (Table. 18.6).

```

confint(myLogit)

##           2.5 %    97.5 %
## (Intercept) -8.6083535 -1.282692
## SE          0.2687893  1.576912

```

So, why do we need to exponentiate the coefficients? Because,

$$\text{logit}(p) = \log \frac{p}{1-p} \rightarrow e^{\text{logit}(p)} = e^{\log \frac{p}{1-p}} \rightarrow \text{RHS} = \frac{p}{1-p}, \text{(odds - ratio, OR)}.$$

```

exp(coef(myLogit))    # exponentiated logit model coefficients

## (Intercept)      SE
## 0.01652254  2.13474149

```

- (Intercept) SE
- $0.01652254 \ 2.13474149 == \exp(0.7583456)$
- **coef**(mylogit) # raw logit model coefficients
- (Intercept) SE
- $-4.1030298 \ 0.7583456$

Table 18.6 Point and interval estimates of the odds ratio of survival

.	OR	2.5%	97.5%
(Intercept)	0.01652254	0.0001825743	0.277290
SE	2.13474149	1.3083794719	4.839986

```
exp(cbind(OR = coef(mylogit), confint(mylogit)))
##                                OR      2.5 %    97.5 %
## (Intercept) 0.01652254 0.0001825743 0.277290
## SE          2.13474149 1.3083794719 4.839986
```

We can compute the LRT and report its p-values (0.0006476922) by using the `with()` function:

```
with(mylogit, df.null - df.residual)
with(mylogit, pchisq(null.deviance - deviance, df.null - df.residual,
lower.tail = FALSE))
## [1] 0.0006476922
```

LRT p-value <0.001 tells us that our model as a whole fits significantly better than an empty model. The deviance residual is $-2 \log \text{likelihood}$, and we can report the model's log likelihood by:

```
LogLik(mylogit)
## 'Log Lik.' -8.046117 (df=2)
```

The LRT compares the data fit of two models. For instance, removing predictor variables from a model may reduce the model quality (i.e., a model will have a lower log likelihood). To statistically assess whether the observed difference in model fit is significant, the LRT compares the difference of the log likelihoods of the two models. When this difference is statistically significant, the full model (the one with more variables) represents a better fit to the data, compared to the reduced model. LRT is computed using the log likelihoods (ll) of the two models:

$$LRT = -2 \ln \left(\frac{L(m_1)}{L(m_2)} \right) = 2(ll(m_2) - ll(m_1)),$$

where:

- m_1 and m_2 are the reduced and the full models, respectively,
- $L(m_1)$ and $L(m_2)$ denote the likelihoods of the 2 models, and
- $ll(m_1)$ and $ll(m_2)$ represent the *log likelihood* (natural log of the model likelihood function).

As $n \rightarrow \infty$, the distribution of the LRT is asymptotically chi-squared with degrees of freedom equal to the number of parameters that are reduced (i.e., the number of variables removed from the model). In our case, $LRT \sim \chi^2_{df=2}$, as we have an intercept and one predictor (SE), and the null model is empty (no parameters).

18.8.3 False Discovery Rate (FDR)

The FDR provides one measure of test or classifier performance:

$$\underbrace{\text{FDR}}_{\text{False Discovery Rate}} = \underbrace{E}_{\text{expectation}} \left(\underbrace{\frac{\# \text{FalsePositives}}{\text{total number of selected features}}}_{\text{False Discovery Proportion}} \right).$$

The Benjamini-Hochberg (BH) FDR procedure involves ordering the p-values, specifying a target FDR, calculating and applying the threshold. Below we show how this is accomplished in R.

```
#p-values entered from smallest to largest
pvals <- c(0.9, 0.35, 0.01, 0.013, 0.014, 0.19, 0.35, 0.5, 0.63, 0.67, 0.75,
0.81, 0.01, 0.051)
length(pvals)

## [1] 14

#enter the target FDR
alpha.star <- 0.05

# order the p-values small to large
pvals <- sort(pvals); pvals

## [1] 0.010 0.010 0.013 0.014 0.051 0.190 0.350 0.350 0.500 0.630 0.670
## [12] 0.750 0.810 0.900

#calculate the threshold for each p-value
threshold<-alpha.star*(1:length(pvals))/length(pvals)

#compare the p-value against its threshold and display results
cbind(pvals, threshold, pvals<=threshold)

##      pvals    threshold
##  [1,] 0.010 0.003571429  0
##  [2,] 0.010 0.007142857  0
##  [3,] 0.013 0.010714286  0
...
## [12,] 0.750 0.042857143  0
## [13,] 0.810 0.046428571  0
## [14,] 0.900 0.050000000  0
```

Start with the smallest p-value and move up we find that the largest k for which the p-value is less than its threshold, α^* , which is $\hat{k} = 4$.

Next, the algorithm rejects the null hypotheses for the tests that correspond to the p-values $p_{(1)}, p_{(2)}, p_{(3)}, p_{(4)}$.

Note: that since we controlled FDR at $\alpha^* = 0.05$, we are guaranteed that on average only 5% of the tests that we rejected are spurious. Since $\alpha^* = 0.05$ of 4 is

quite small and less than 1, we are confident that none of our rejections are expected to be spurious.

The Bonferroni corrected α for these data is $\frac{0.05}{14} = 0.0036$. If we had used this family-wise error rate in our individual hypothesis tests, then we would have concluded that none of our 14 results were significant!

Graphical Interpretation of the Benjamini-Hochberg (BH) Method

There's a graphical interpretation of the BH calculations.

- Sort the p-values from largest to smallest.
- Plot the ordered p-values $p_{(k)}$ on the y-axis versus their indexes on the x-axis.
- Superimpose on this plot a line that passes through the origin and has slope α^* .

Any p-value that falls on or below this line corresponds to a significant result (Fig. 18.18).

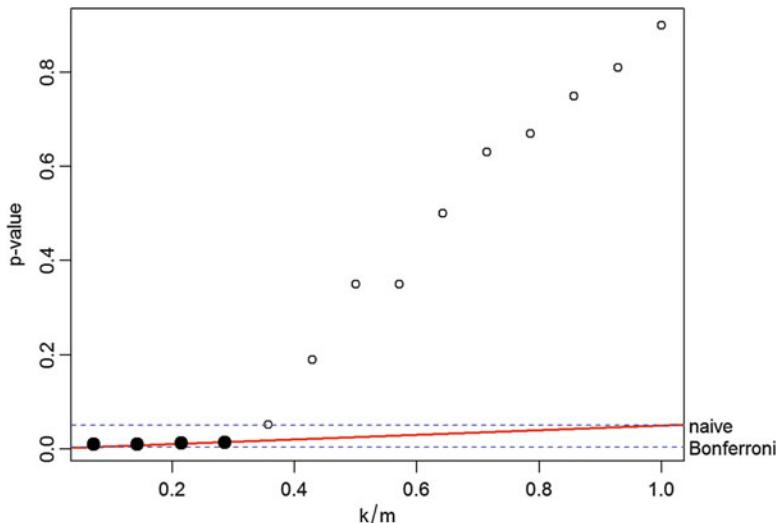


Fig. 18.18 Graphical representation of the naïve, conservative Bonferroni, and FDR critical p-values

```

#generate the values to be plotted on x-axis
x.values<- (1:Length(pvals))/Length(pvals)
#widen right margin to make room for labels
par(mar=c(4.1, 4.1, 1.1, 4.1))

#plot points
plot(x.values, pvals, xLab=expression(k/m), yLab="p-value")
#, ylim=c(0.0, 0.4))

#add FDR line
abline(0, .05, col=2, lwd=2)

#add naive threshold line
abline(h=.05, col=4, lty=2)

#add Bonferroni-corrected threshold line
abline(h=.05/Length(pvals), col=4, lty=2)

#label lines
mtext(c('naive', 'Bonferroni'), side=4, at=c(.05, .05/Length(pvals)),
las=1, line=0.2)
#select observations that are less than threshold
for.test <- cbind(1:Length(pvals), pvals)
pass.test <- for.test[pvals <= 0.05*x.values, ]
pass.test
##      pvals
## 4.000 0.014
#use the largest k to color points that meet Benjamini-Hochberg FDR test
Last<-ifelse(is.vector(pass.test), pass.test[1],
pass.test[nrow(pass.test), 1])
points(x.values[1:last], pvals[1:last], pch=19, cex=1.5)

```

FDR Adjusting the p-Values

R can automatically performs the Benjamini-Hochberg procedure. The adjusted p-values are obtained by

```
pvals.adjusted <- p.adjust(pvals, "BH")
```

The adjusted p-values indicate the corresponding null hypothesis we need to reject to preserve the initial α^* false-positive rate. We can also compute the adjusted p-values as follows:

```
#calculate the term that appears in the innermost minimum function
test.p <- length(pvals)/(1:length(pvals))*pvals
test.p

## [1] 0.14000000 0.07000000 0.06066667 0.04900000 0.14280000 0.44333333
## [7] 0.70000000 0.61250000 0.77777778 0.88200000 0.85272727 0.87500000
## [13] 0.87230769 0.90000000

#use a loop to run through each p-value and carry out the adjustment
adj.p <- numeric(14)
for(i in 1:14) {
  adj.p[i]<-min(test.p[i:length(test.p)])
  ifelse(adj.p[i]>1, 1, adj.p[i])
}
adj.p

## [1] 0.0490000 0.0490000 0.0490000 0.0490000 0.1428000 0.4433333 0.6125000
## [8] 0.6125000 0.7777778 0.8527273 0.8527273 0.8723077 0.8723077 0.9000000
```

Note that the manually computed (`adj.p`) and the automatically computed (`pvals.adjusted`) adjusted-p-values are the same.

18.8.4 Running the Knockoff Filter

We now run the knockoff filter along with the Benjamini-Hochberg (BH) procedure for controlling the false-positive rate of feature selection. More details about the knock-off filtering methods are available online.

Before running either selection procedure, remove rows with missing values, reduce the design matrix by removing predictor columns that do not appear frequently (e.g., at least three times in the sample), and remove any columns that are duplicates.

```
library(knockoff)

# Direct call to knockoff filtering looks like this:
fdr <- 0.1
result = knockoff.filter(X, Y, fdr = fdr, knockoffs = 'equicorrelated')

names(result$selected)

## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Sex"                            "Weight"
## [5] "Age"                            "chr12_rs34637584_GT"
## [7] "chr17_rs11012_GT"              "chr17_rs199533_GT"

knockoff_selected <- names(result$selected)

# Run BH (Benjamini-Hochberg)
k = ncol(X)
Lm.fit = Lm(Y ~ X - 1) # no intercept
# Alternatively: dat = as.data.frame(cbind(Y,X))
# lm.fit = lm(Y ~ . -1,data=dat ) # no intercept
```

```

p.values = coef(summary(lm.fit))[, 4]
cutoff = max(c(0, which(sort(p.values) <= fdr * (1:k) / k)))
BH_selected = names(which(p.values <= fdr * cutoff / k))

knockoff_selected; BH_selected

## [1] "L_cingulate_gyrus_ComputeArea" "R_putamen_ComputeArea"
## [3] "Sex"                            "Weight"
## [5] "Age"                            "chr12_rs34637584_GT"
## [7] "chr17_rs11012_GT"              "chr17_rs199533_GT"

## [1] "XL_putamen_ComputeArea" "XL_putamen_Volume"
## [3] "XSex"                  "XWeight"
## [5] "XAge"                  "Xchr17_rs11868035_GT"
## [7] "Xchr17_rs11012_GT"      "Xchr17_rs12185268_GT"

# Housekeeping: remove the "X" prefixes in the BH_selected list of features
for(i in 1:length(BH_selected)){
  BH_selected[i] <- substring(BH_selected[i], 2)
}

intersect(BH_selected, knockoff_selected)

## [1] "Sex"          "Weight"        "Age"
## [4] "chr17_rs11012_GT"

```

We see that there are some features that are selected by both methods suggesting they may be indeed salient.

Try to apply some of these techniques to other data from the list of our Case-Studies.

18.9 Assignment: 18. Regularized Linear Modeling and Knockoff Filtering

Use the ALS (Case Study 15) data to:

- Detect and impute missing value if any.
- Use the ALSFRS_slope as a clinically relevant outcome variable.
- Randomly split data into training (70%) and testing (30%) datasets.
- Use the LASSO to fit a model with cross validation (with optimized regularization parameter) and visualize the result.
- Similarly, train a ridge regression model.
- Train OLS model and improve it with stepwise variable selection.
- Report the coefficient estimates for OLS, Stepwise OLS with AIC, Ridge and LASSO.
- Calculate the predicted values for all 4 models and report the models performance metrics (RMSE and R^2).
- Apply knockoff filtering for variable selection, controlling the false discovery rate.
- Compare the variables selected by Stepwise OLS, LASSO and knockoff.

References

- Barber RF, Candès EJ. Controlling the false discovery rate via knockoffs. *The Annals of Statistics*. 2015;43(5):2055-85.
<https://web.stanford.edu/~candes/Knockoffs/>
<https://cran.r-project.org/web/packages/knockoff/vignettes/>
Liu, H and Motoda, H (eds.) (2007) *Computational Methods of Feature Selection*, Chapman & Hall/CRC, ISBN 1584888792.
http://wiki.socr.umich.edu/index.php/SOCR_Data_PD_BiomedBigMetadata
https://umich.instructure.com/files/330397/download?download_frd=1

Chapter 19

Big Longitudinal Data Analysis



The time-varying (longitudinal) characteristics of large information flows represent a special case of the complexity and the dynamic multi-scale nature of big biomedical data that we discussed in the DSPA Motivation section. Previously, in Chap. 4, we saw space-time (4D) functional magnetic resonance imaging (fMRI) data, and in Chap. 16 we discussed streaming data, which also has a natural temporal dimension. Now we will go deeper into managing, modeling and analyzing big longitudinal data.

In this Chapter, we will expand our predictive data analytic strategies specifically for analyzing big longitudinal data. We will interrogate datasets that track the same type of information, for the same subjects, units or locations, over a period of time. Specifically, we will present time series analysis, forecasting using autoregressive integrated moving average (ARIMA) models, structural equation models (SEM), and longitudinal data analysis via linear mixed models.

19.1 Time Series Analysis

Time series analysis relies on models like ARIMA (Autoregressive integrated moving average) that utilize past longitudinal information to predict near future outcomes. Times series data tend to track univariate, sometimes multivariate, processes over a continuous time interval. The stock market, e.g., daily closing value of the Dow Jones Industrial Average index, electroencephalography (EEG) data, and functional magnetic resonance imaging provide examples of such longitudinal datasets (timeseries).

The basic concepts in time series analysis include:

- The characteristics of (*second-order*) *stationary time series* (e.g., first two moments are stable over time) do not depend on the time at which the series process is observed.
- *Differencing* – a transformation applied to time-series data to make it stationary; Differences between consecutive time-observations may be computed by y_t

$= y_t - y_{t-1}$. Differencing removes the level changes in the time series, eliminates trend, reduces seasonality, and stabilizes the mean of the time series. Differencing the time series repeatedly may yield a stationary time series. For example, a second order differencing:

$$\begin{aligned} y_t'' &= y_t' - y_{t-1}' \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned}$$

- *Seasonal differencing* is computed as a difference between one observation and its corresponding observation in the previous epoch, or season (e.g., annually, there are $m = 4$ seasons), like in this example:

$$y_t''' = y_t - y_{t-m} \quad \text{where } m = \text{number of seasons.}$$

- The differenced data may then be used to estimate an ARMA model.

We will use the Beijing air quality PM2.5 dataset as an example to demonstrate the analysis process. This dataset measures air pollutants - PM2.5 particles in micrograms per cubic meter over a period of 8 years (2008–2016). It measures the *hourly average of the number of particles that are of size 2.5 microns (PM2.5)* once per hour in Beijing, China.

Let's first import the dataset into R.

```
beijing.pm25<-read.csv("https://umich.instructure.com/files/1823138/downLoad
?download_frd=1")
summary(beijing.pm25)
##      Index           Site      Parameter      Date..LST.
##  Min.   : 1  Beijing:69335  PM2.5:69335  3/13/2011 3:00:   2
##  1st Qu.:17335                    3/13/2016 3:00:   2
##  Median :34668                    3/14/2010 3:00:   2
##  Mean   :34668                    3/8/2009  3:00 :   2
##  3rd Qu.:52002                    3/8/2015  3:00 :   2
##  Max.   :69335                    3/9/2014  3:00 :   2
##                                         (Other)      :69323
##      Year        Month        Day        Hour
##  Min.   :2008  Min.   : 1.000  Min.   : 1.00  Min.   : 0.0
##  1st Qu.:2010  1st Qu.: 4.000  1st Qu.: 8.00  1st Qu.: 5.5
##  Median :2012  Median : 6.000  Median :16.00  Median :11.0
##  Mean   :2012  Mean   : 6.407  Mean   :15.73  Mean   :11.5
##  3rd Qu.:2014  3rd Qu.: 9.000  3rd Qu.:23.00  3rd Qu.:17.5
##  Max.   :2016  Max.   :12.000  Max.   :31.00  Max.   :23.0
##
##      Value        Duration      QC.Name
##  Min.   :-999.00  1 Hr:69335  Missing: 4408
##  1st Qu.: 22.00                    Valid  :64927
##  Median : 63.00
##  Mean   : 24.99
##  3rd Qu.:125.00
##  Max.   : 994.00
```

The `Value` column records PM2.5 AQI (Air Quality Index) for 8 years. We observe that there are some missing data in the `Value` column. By looking at the `QC.Name` column, we only have about 6.5% (4408 observations) missing values. One way of solving data-missingness problems, where incomplete observations are recorded, is to replace the absent elements by the corresponding variable mean.

```
beijing.pm25[beijing.pm25$value == -999, 9] <- NA
beijing.pm25[is.na(beijing.pm25$value), 9] <- floor(mean(beijing.pm25$value,
na.rm = T))
```

Here we first reassign the missing values into `NA` labels. Then we replace all `NA` labels with the `mean` computed using all non-missing observations. Note that the `floor()` function casts the arithmetic averages as integer numbers, which is needed as AQI values are expected to be whole numbers.

Now, let's observe the trend of hourly average PM2.5 across 1 day. You can see a significant pattern: The PM2.5 level peaks in the afternoons and is the lowest in the early mornings. It exhibits approximate periodic boundary conditions (these patterns oscillate daily) (Fig. 19.1).

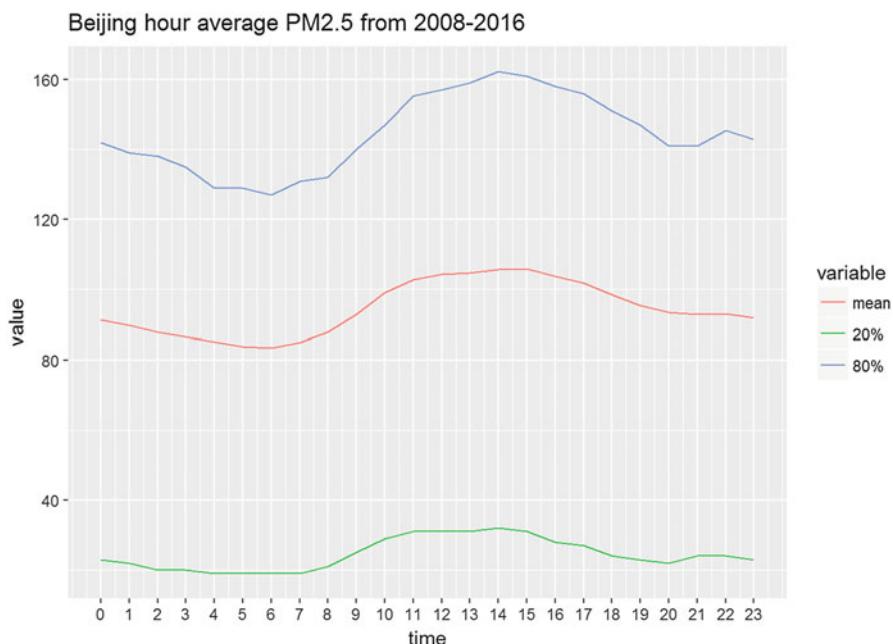


Fig. 19.1 Time course of the mean, top-20%, and bottom-20% air quality in Beijing (PPM2.5)

```

require(ggplot2)
id = 1:nrow(beijing.pm25)
mat = matrix(0, nrow=24, ncol=3)
stat = function(x){
  c(mean(beijing.pm25[iid, "Value"]), quantile(beijing.pm25[iid, "Value"], c(0.2, 0.8)))
}
for (i in 1:24){
  iid = which(id%%24==i-1)
  mat[i,] = stat(iid)
}

mat <- as.data.frame(mat)
colnames(mat) <- c("mean", "20%", "80%")
mat$time = c(15:23, 0:14)
require(reshape2)

## Loading required package: reshape2

dt <- melt(mat, id="time")
colnames(dt)

## [1] "time"      "variable"   "value"

ggplot(data = dt, mapping = aes(x=time, y=value, color=variable)) + geom_line() +
  scale_x_continuous(breaks = 0:23) + ggtitle("Beijing hour average PM2.5 from 2008-2016")

```

Are there any daily or monthly trends? We can start the data interrogation by building an ARIMA model and examining detailed patterns in the data.

19.1.1 Step 1: Plot Time Series

To begin with, we can visualize the overall trend by plotting PM2.5 values against time. This can be achieved using the `plyr` package.

```

library(plyr)
ts<-ts(beijing.pm25$value, start=1, end=69335, frequency=1)
ts.plot(ts)

```

The dataset is recorded hourly, and the 8-year time interval includes about 69,335 h of records. Therefore, we start at the first hour and end with 69,335th h. Each hour has a univariate PM2.5 AQI value measurement, so `frequency=1`.

From this time series plot, Fig. 19.2, we observe that the data has some peaks but most of the AQIs stay under 300 (which is considered hazardous).

The original plot seems have no trend at all. Remember we have our measurements in hours. Will there be any difference if we use monthly average instead of hourly reported values? In this case, we can use Simple Moving Average (SMA) technique to smooth the original graph.

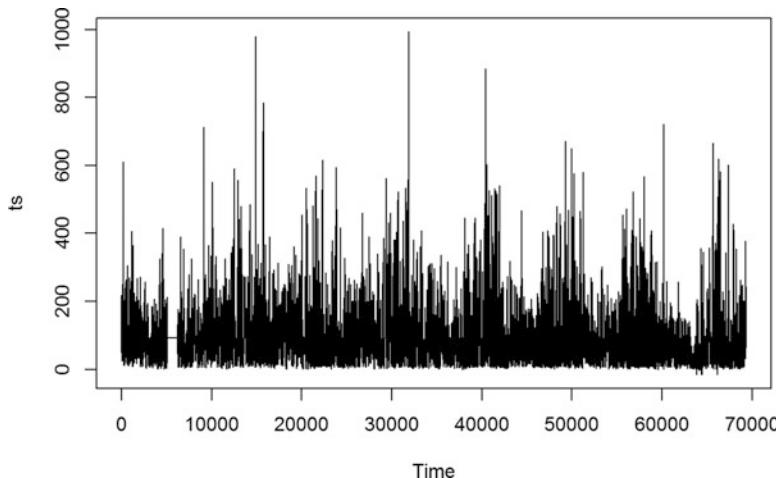


Fig. 19.2 Raw time-series plot of the Beijing air quality measures (2008–2016)

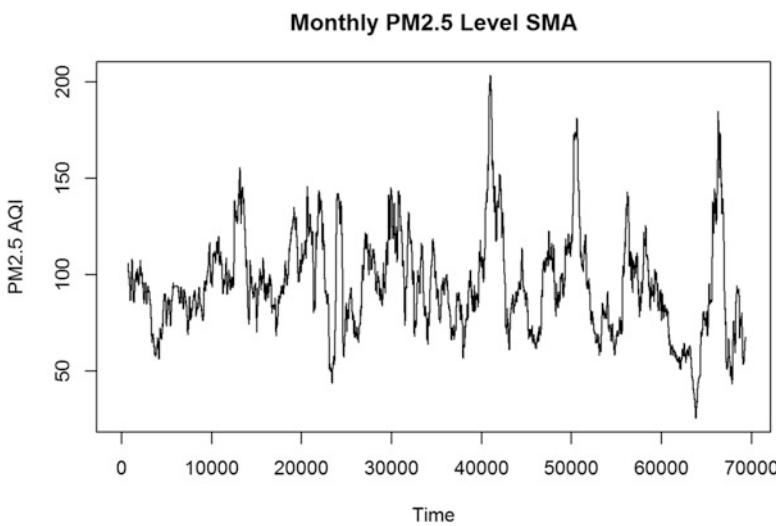


Fig. 19.3 Simple moving monthly average PM2.5 air quality index values

To accomplish this, we need to install the `TTR` package and utilize the `SMA()` method (Fig. 19.3).

```
#install.packages("TTR")
library(TTR)
bj.month<-SMA(ts, n=720)
plot.ts(bj.month, main="Monthly PM2.5 Level SMA", yLab="PM2.5 AQI")
```

Here we chose `n` to be $24 * 30 = 720$, and we can see some pattern. It seems that for the first 4 years (or approximately 35,040 h), the AQI fluctuates less than the last

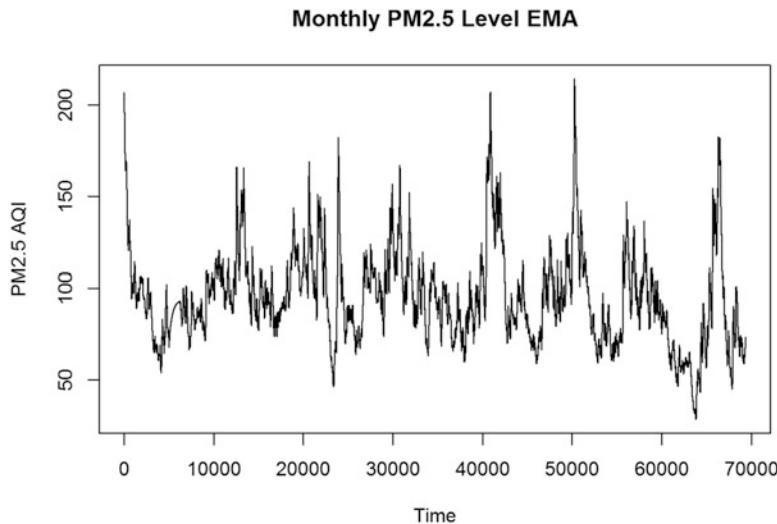


Fig. 19.4 Exponentially-weighted monthly mean of PM2.5 air quality

5 years. Let's see what happens if we use *exponentially-weighted mean*, instead of *arithmetic mean*.

```
bj.month<-EMA(ts, n=1, ratio = 2/(720+1))
plot.ts(bj.month, main="Monthly PM2.5 Level EMA", ylab="PM2.5 AQI")
```

The pattern seems less obvious in this graph, Fig. 19.4. Here we used exponential smoothing ratio of $2/(n + 1)$.

19.1.2 Step 2: Find Proper Parameter Values for ARIMA Model

ARIMA models have 2 components: autoregressive (AR) part and moving average (MA) part. An $ARMA(p, d, q)$ model is a model with p terms in AR, q terms in MA, and d representing the order difference. Differencing is used to make the original dataset approximately stationary. $ARMA(p, d, q)$ has the following analytical form:

$$\left(1 - \sum_{i=1}^p \phi_i L^i\right) (1 - L)^d X_t = \left(1 + \sum_{i=1}^q \theta_i L^i\right) \epsilon_t.$$

19.1.3 Check the Differencing Parameter

First, let's try to determine the parameter d . To make the data stationary on the mean (remove any trend), we can use first differencing or second order differencing. Mathematically, first differencing is taking the difference between two adjacent data points:

$$y_t' = y_t - y_{t-1}.$$

While second order differencing is differencing the data twice:

$$y_t^* = y_t' - y_{t-1}' = y_t - 2y_{t-1} + y_{t-2}.$$

Let's see which differencing method is proper for the Beijing PM2.5 dataset. Function `diff()` in R base can be used to calculate differencing. We can plot the differences by `plot.ts()` (Fig. 19.5).

```
par(mfrow= c(2, 1))
bj.diff2<-diff(ts, differences=2)
plot.ts(bj.diff2, main="2nd differencing")
bj.diff<-diff(ts, differences=1)
plot.ts(bj.diff, main="1st differencing")
```

Neither of them appears quite stationary. In this case, we can consider using some smoothing techniques on the data like we just did above (`bj.months<-SMA(ts, n=720)`). Let's see if smoothing by exponentially-weighted mean (EMA) can help making the data approximately stationary (Fig. 19.6).

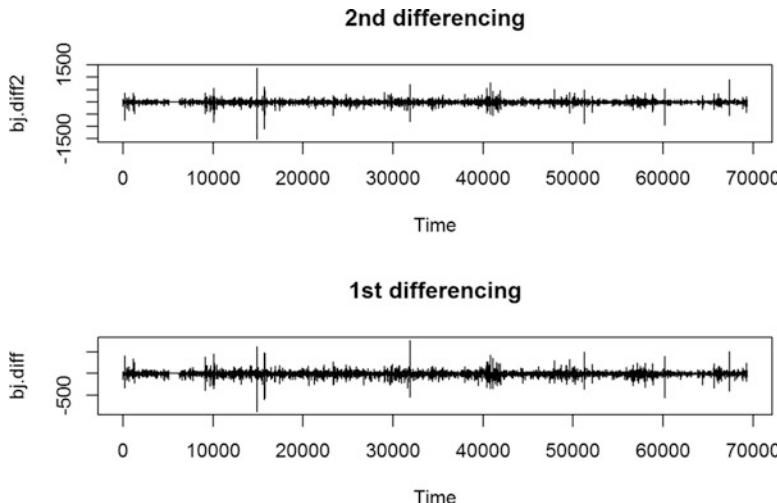


Fig. 19.5 First- and second-order differencing of the AQI data

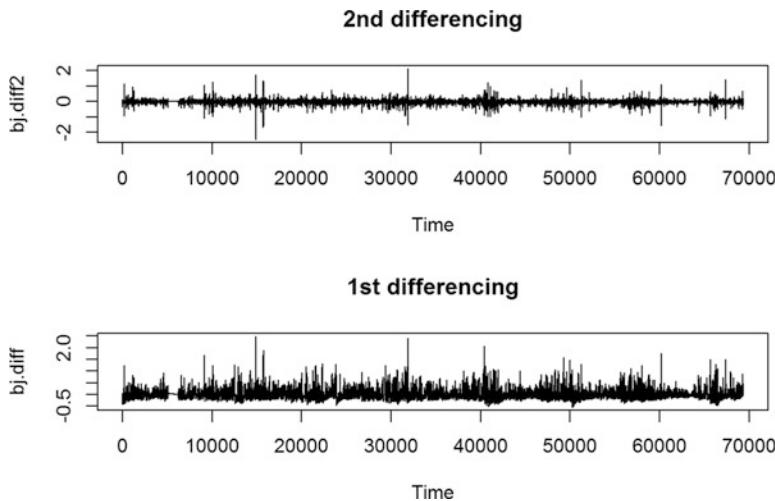


Fig. 19.6 Monthly-smoothed first- and second-order differencing of the AQI data

```
par(mfrow=c(2, 1))
bj.diff2<-diff(bj.month, differences=2)
plot.ts(bj.diff2, main="2nd differencing")
bj.diff<-diff(bj.month, differences=1)
plot.ts(bj.diff, main="1st differencing")
```

Both of these EMA-filtered graphs have tempered variance and appear pretty stationary with respect to the first two moments, mean and variance.

19.1.4 Identifying the AR and MA Parameters

To decide the auto-regressive (AR) and moving average (MA) parameters in the model we need to create **autocorrelation factor (ACF)** and **partial autocorrelation factor (PACF) plots**. PACF may suggest a value for the AR-term parameter q , and ACF may help us determine the MA-term parameter p . We plot the ACF and PACF using the approximately stationary time series, `bj.diff` object (Fig. 19.7).

```
par(mfrow=c(1, 2))
acf(ts(bj.diff), Lag.max = 20, main="ACF")
pacf(ts(bj.diff), Lag.max = 20, main="PACF")
```

- Pure AR model, ($q = 0$), will have a cut off at lag p in the PACF.
- Pure MA model, ($p = 0$), will have a cut off at lag q in the ACF.
- ARIMA(p, q) will (eventually) have a decay in both.

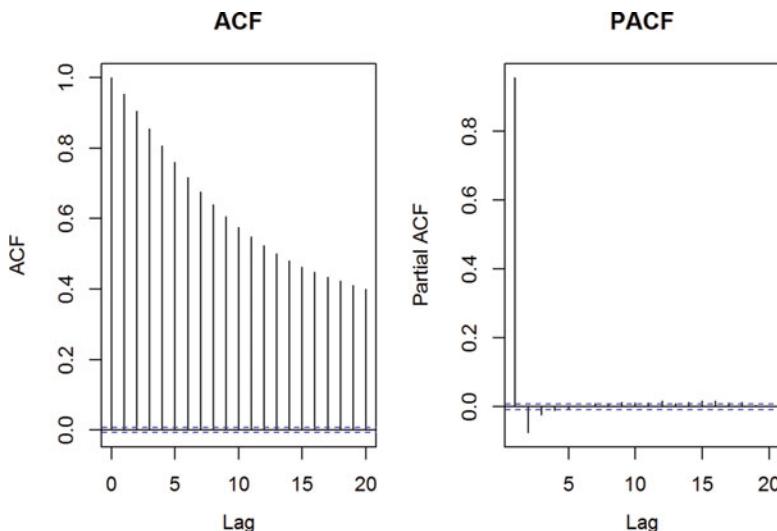


Fig. 19.7 Autocorrelation factor (ACF) and partial autocorrelation factor (PACF) plots of `bj.diff`

All spikes in the plots are outside of the (normal) insignificant zone in the ACF plot while two of them are significant in the PACF plot. In this case, the best ARIMA model is likely to have both AR and MA parts.

We can examine for seasonal effects in the data using `stats::stl()`, a flexible function for decomposing and forecasting the series, which uses averaging to calculate the seasonal component of the series and then subtracts the seasonality. Decomposing the series and removing the seasonality can be done by subtracting the seasonal component from the original series using `forecast::seasadj()`. The frequency parameter in the `ts()` object specifies the periodicity of the data or the number of observations per period, e.g., 30, for monthly smoothed daily data (Fig. 19.8).

```
count_ma = ts(bj.month, frequency=30)
decomp = stl(count_ma, s.window="periodic")
deseasonal_count <- forecast::seasadj(decomp)
plot(decomp)
```

The augmented Dickey-Fuller (ADF) test, `tseries::adf.test` can be used to examine the timeseries stationarity. The *null hypothesis is that the series is non-stationary*. The ADF test quantifies if the change in the series can be explained by a lagged value and a linear trend. Non-stationary series can be *corrected* by differencing to remove trends or cycles.

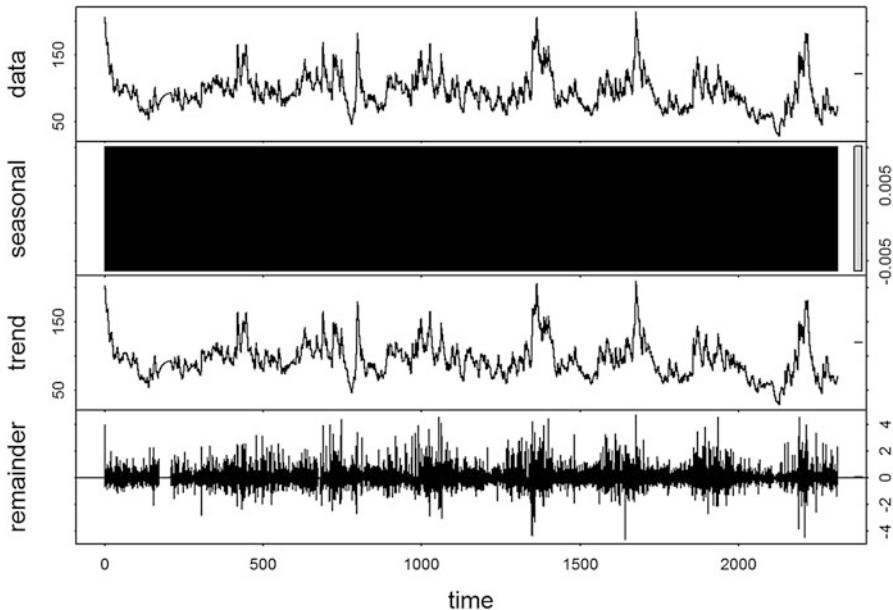


Fig. 19.8 Trend and seasonal decomposition of the time-series

```
tseries::adf.test(count_ma, alternative = "stationary")
##  Augmented Dickey-Fuller Test
##
## data:  count_ma
## Dickey-Fuller = -8.0313, Lag order = 41, p-value = 0.01
## alternative hypothesis: stationary

tseries::adf.test(bj.diff, alternative = "stationary")
##  Augmented Dickey-Fuller Test
##
## data:  bj.diff
## Dickey-Fuller = -29.188, Lag order = 41, p-value = 0.01
## alternative hypothesis: stationary
```

We see that we can reject the null and therefore, there is no statistically significant non-stationarity in the `bj.diff` timeseries.

19.1.5 Step 3: Build an ARIMA Model

As we have some evidence suggesting $d = 1$, the `auto.arima()` function in the `forecast` package can help us to find the optimal estimates for the remaining pair parameters of the ARIMA model, p and q .

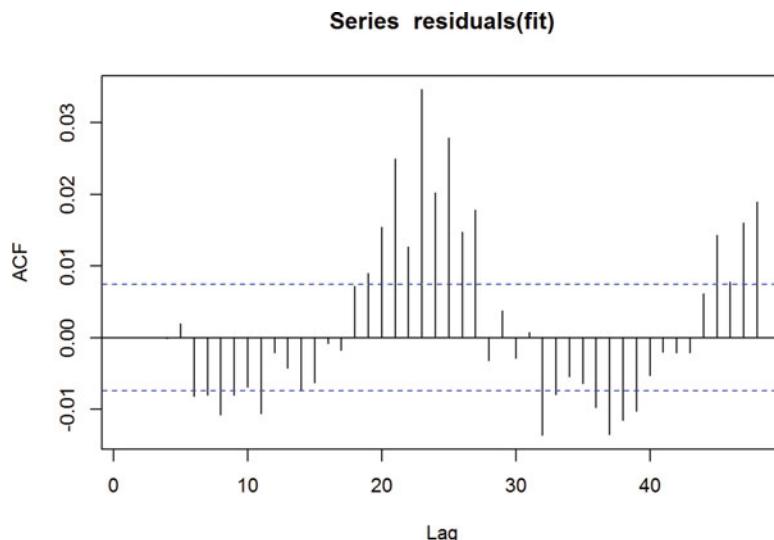


Fig. 19.9 ACF of the time-series residuals

```

# install.packages("forecast")
library(forecast)
fit<-auto.arima(bj.month, approx=F, trace = F)
fit

## Series: bj.month
## ARIMA(1,1,4)
##
## Coefficients:
##             ar1      ma1      ma2      ma3      ma4
##             0.9426  0.0813  0.0323  0.0156  0.0074
## s.e.    0.0016  0.0041  0.0041  0.0041  0.0041
##
## sigma^2 estimated as 0.004604:  Log Likelihood=88161.91
## AIC=-176311.8  AICc=-176311.8  BIC=-176257

Acf(residuals(fit))

```

Finally, the optimal model determined by the step-wise selection is ARIMA (1, 1, 4). The residual plot is show on Fig. 19.9.

We can also use external information to fit ARIMA models. For example, if we want to add the month information, in case we suspect a seasonal change in PM2.5 AQI, we can use the following script.

```

fit1<-auto.arima(bj.month, xreg=beijing.pm25$Month, approx=F, trace = F)
fit1

## Series: bj.month
## Regression with ARIMA(1,1,4) errors
##
## Coefficients:
##             ar1      ma1      ma2      ma3      ma4  beijing.pm25$Month
##             0.9427  0.0813  0.0322  0.0156  0.0075          -0.0021
## s.e.    0.0016  0.0041  0.0041  0.0041  0.0041          0.0015
##
## sigma^2 estimated as 0.004604:  Log Likelihood=88162.9
## AIC=-176311.8  AICc=-176311.8  BIC=-176247.8

fit3<-arima(bj.month, order = c(2, 1, 0))
fit3

## Call:
## arima(x = bj.month, order = c(2, 1, 0))
##
## Coefficients:
##             ar1      ar2
##             1.0260 -0.0747
## s.e.    0.0038  0.0038
##
## sigma^2 estimated as 0.004606:  Log Likelihood = 88138.32, aic=-176270.6

```

We want the model AIC and BIC to be as small as possible. In terms of AIC and BIC, this model is not drastically different compared to the last model without Month predictor. Also, the coefficient of Month is very small and not significant (according to the t-test) and thus can be removed.

We can examine further the ACF and the PACF plots and the residuals to determine the model quality. When the model order parameters and structure are correctly specified, we expect no significant autocorrelations present in the model residual plots.

```
tsdisplay(residuals(fit), Lag.max=45, main='(1,1,4) Model Residuals')
```

There is a clear pattern present in ACF/PACF plots, Fig. 19.10, suggesting that the model residuals repeat with an approximate lag of 12 or 24 months. We may try a modified model with a different parameters, e.g., $p = 24$ or $q = 24$. We can define a new `displayForecastErrors()` function to show a histogram of the forecasted errors (Figs. 19.11 and 19.12).

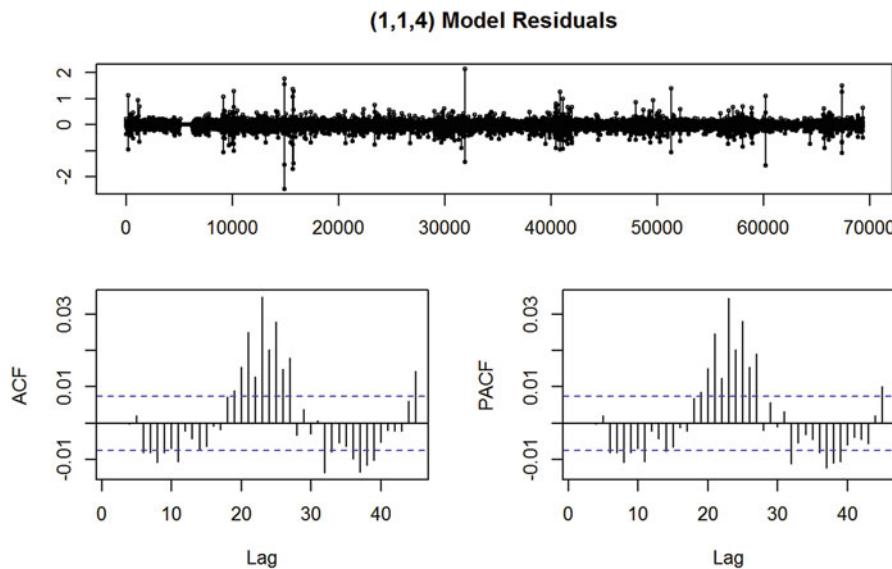


Fig. 19.10 ARIMA(1,1,4) model plot, ACF and PACF plots of the residuals for `bj.month`

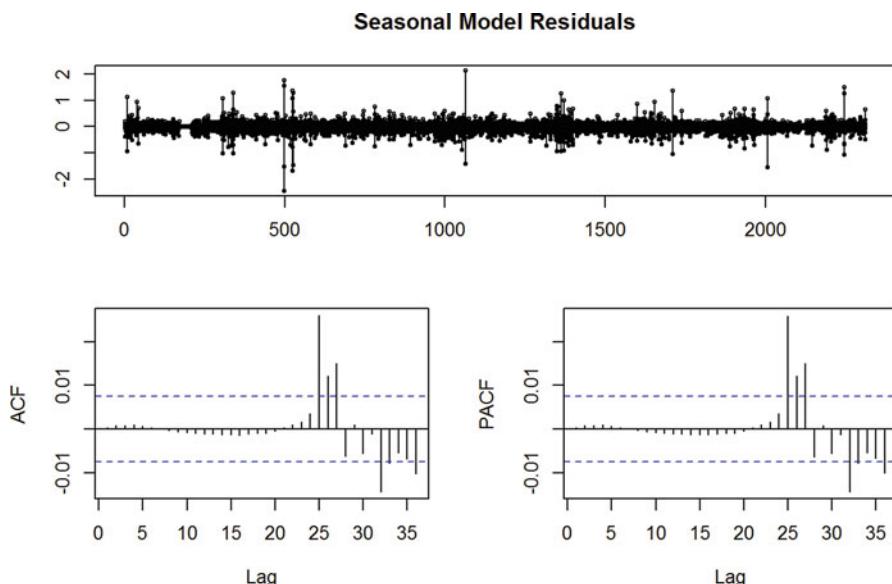


Fig. 19.11 An improved ARIMA(1,1,24) model plot, ACF and PACF plots of the residuals for `bj.month`

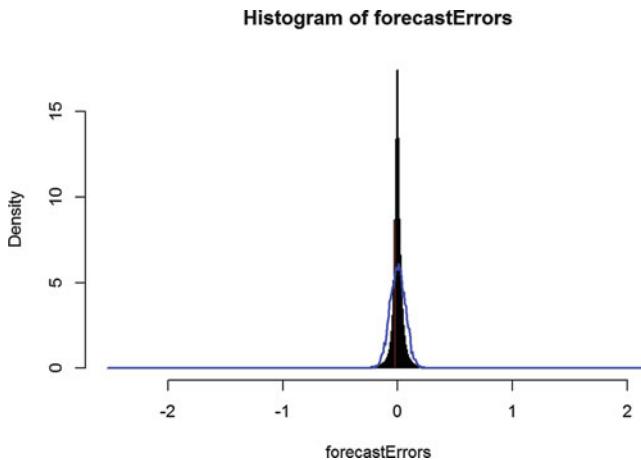


Fig. 19.12 Diagnostic plot of the residuals of the ARIMA(1,1,24) time-series model for `bj.month`

```

fit24 <- arima(deseasonal_count, order=c(1,1,24)); fit24
## Call:
## arima(x = deseasonal_count, order = c(1, 1, 24))
##
## Coefficients:
##             ar1      ma1      ma2      ma3      ma4      ma5      ma6      ma7
##             0.9496  0.0711  0.0214  0.0054  -0.0025  -0.0070  -0.0161  -0.0149
## s.e.      0.0032  0.0049  0.0049  0.0048  0.0047  0.0046  0.0045  0.0044
##             ma8      ma9      ma10     ma11     ma12     ma13     ma14
##            -0.0162 -0.0118 -0.0100 -0.0136 -0.0045 -0.0055 -0.0075
## s.e.      0.0044  0.0043  0.0042  0.0042  0.0042  0.0041  0.0041
##             ma15     ma16     ma17     ma18     ma19     ma20     ma21     ma22
##            -0.0060 -0.0005 -0.0019  0.0066  0.0088  0.0156  0.0247  0.0117
## s.e.      0.0041  0.0041  0.0041  0.0041  0.0041  0.0040  0.0040  0.0040
##             ma23     ma24
##            0.0319  0.0156
## s.e.      0.0040  0.0039
##
## sigma^2 estimated as 0.004585: Log Likelihood = 88295.88, aic = -176539.8
tsdisplay(residuals(fit24), lag.max=36, main='Seasonal Model Residuals')

displayForecastErrors <- function(forecastErrors)
{
  # Generate a histogram of the Forecast Errors
  binsize <- IQR(forecastErrors)/4
  sd     <- sd(forecastErrors)
  min   <- min(forecastErrors) - sd
  max   <- max(forecastErrors) + sd

  # Generate 5K normal(0,sd) RVs
  norm <- rnorm(5000, mean=0, sd=sd)
  min2 <- min(norm)
  max2 <- max(norm)
}

```

```

if (min2 < min) { min <- min2 }
if (max2 > max) { max <- max2 }

# Plot red histogram of the forecast errors
bins <- seq(min, max, binsize)
hist(forecastErrors, col="red", freq=FALSE, breaks=bins)

myHist <- hist(norm, plot=FALSE, breaks=bins)

# Overlay the Blue normal curve on top of forecastErrors histogram
points(myHist$mid, myHist$density, type="l", col="blue", lwd=2)
}

displayForecastErrors(residuals(fit24))

```

19.1.6 Step 4: Forecasting with ARIMA Model

Now, we can use our models to make predictions for future PM2.5 AQI. We will use the function `forecast()` to make predictions. In this function, we have to specify the number of periods we want to forecast. Using the smoothed data, we can make predictions for the next month, July 2016. As each month has about $24 \times 30 = 720$ h, we specify a horizon $h = 720$ (Fig. 19.13).

```

par(mfrow=c(1, 1))
ts.forecasts<-forecast(fit, h=720)
plot(ts.forecasts, include = 2880)

```

When plotting the forecasted values with the original smoothed data, we include only the last 3 months in the original smoothed data to see the predicted values clearer. The shaded regions indicate ranges of expected errors. The darker (inner) region represents by 80% confidence range and the lighter (outer) region bounds by the

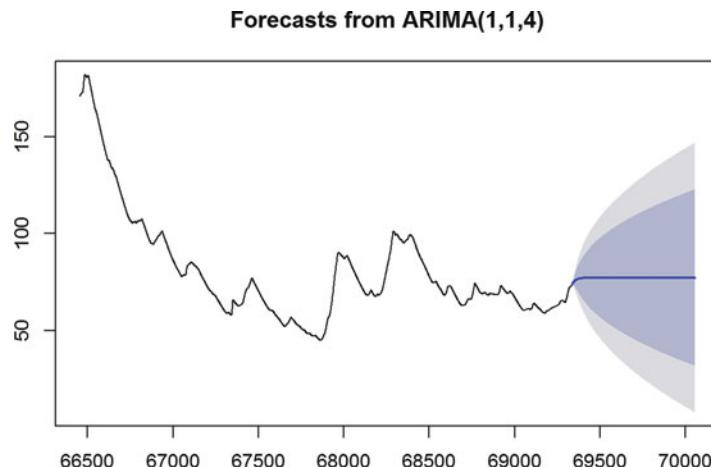


Fig. 19.13 Prospective out-of-range prediction intervals of the ARIMA(1,1,4) time-series model

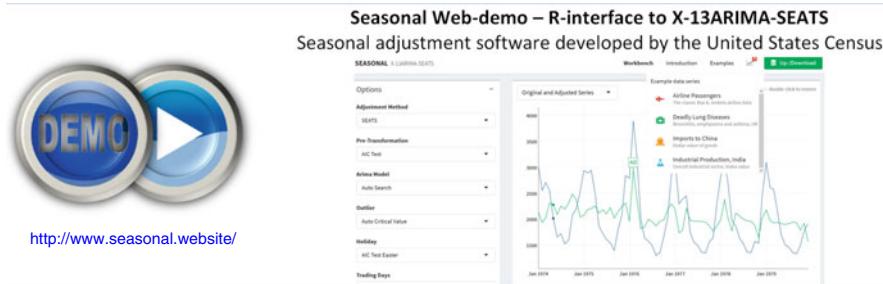


Fig. 19.14 Live Demo: Interactive US Census ARIMA modeling

95% interval. Obviously near-term forecasts have tighter ranges of expected errors, compared to longer-term forecasts where the variability naturally expands. A live demo of US Census data is shown on Fig. 19.14.

19.2 Structural Equation Modeling (SEM)-Latent Variables

Timeseries analyses provide effective strategies to interrogate longitudinal univariate data. What happens if we have multiple, potentially associated, measurements recorded at each time point?

SEM is a general multivariate statistical analysis technique that can be used for causal modeling/inference, path analysis, confirmatory factor analysis (CFA), covariance structure modeling, and correlation structure modeling. This method allows *separation of observed and latent variables*. Other standard statistical procedures may be viewed as special cases of SEM, where statistical significance may be less important, and covariances are the core of structural equation models.

Latent variables are features that are not directly observed but may be inferred from the actually observed variables. In other words, a combination or transformation of observed variables can create latent features, which may help us reduce the dimensionality of data. Also, SEM can address multi-collinearity issues when we fit models because we can combine some high collinearity variables to create a single (latent) variable, which can then be included into the model.

19.2.1 Foundations of SEM

SEMs consist of two complementary components: (1) a *path model*, quantifying specific cause-and-effect relationships between observed variables, and (2) a *measurement model*, quantifying latent linkages between unobservable components and observed variables. The LISREL (LInear Structural RELations) framework represents a unifying mathematical strategy to specify these linkages, see Grace 2006.

The most general kind of SEM is a structural regression path model with latent variables, which account for measurement errors of observed variables. *Model identification* determines whether the model allows for unique parameter estimates and may be based on model degrees of freedom ($df_M \geq 0$) or a known scale for every latent feature. If ν represents the number of observed variables, then the total degrees of freedom for a SEM, $\frac{\nu(1+\nu)}{2}$, corresponds to the number of variances and unique covariances in a variance-covariance matrix for all the features, and the model degrees of freedom, $df_M = \frac{\nu(1+\nu)}{2} - l$, where l is the number of estimated parameters.

Examples include:

- *Just-identified model* ($df_M = 0$) with unique parameter estimates,
- *Over-identified model* ($df_M > 0$) desirable for model testing and assessment,
- *Under-identified model* ($df_M < 0$) is not guaranteed unique solutions for all parameters. In practice, such models occur when the effective degrees of freedom are reduced due to two or more highly-correlated features, which presents problems with parameter estimation. In these situations, we can exclude or combine some of the features boosting the degrees of freedom.

The latent variables' *scale* property reflects their unobservable, not measurable, characteristics. The latent scale, or unit, may be inferred from one of its observed constituent variables, e.g., by imposing a unit loading identification constraint fixing at 1.0 the factor loading of one observed variable.

An SEM model with appropriate *scale* and degrees of freedom conditions may be identifiable subject to Bollen's two-step identification rule. When both the CFA path components of the SEM model are identifiable, then the whole SR model is identified, and model fitting can be initiated.

- For the confirmatory factor analysis (CFA) part of the SEM, identification requires (1) a minimum of two observed variables for each latent feature, (2) independence between measurement errors and the latent variables, and (3) independence between measurement errors.
- For the path component of the SEM, ignoring any observed variables used to measure latent variables, model identification requires: (1) errors associated with endogenous latent variables to be uncorrelated, and (2) all causal effects to be unidirectional.

The LISREL representation can be summarized by the following matrix equations:

$$\text{measurement model component } \begin{cases} x = \Lambda_x \xi + \delta, \\ y = \Lambda_y \eta + \epsilon. \end{cases}$$

And

$$\text{path model component } \eta = B\eta + \Gamma\xi + \zeta,$$

where:

- $x_{p \times 1}$ is a vector of observed *exogenous variables* representing a linear function of $\xi_{j \times 1}$, vector of *exogenous latent variables*,
- $\delta_{p \times 1}$ is a vector of *measurement error*, Λ_x is a $p \times j$ matrix of factor loadings relating x to ξ ,
- $y_{q \times 1}$ is a vector of observed *endogenous variables*,
- $\eta_{k \times 1}$ is a vector of *endogenous latent variables*,
- $\epsilon_{q \times 1}$ is a vector of *measurement error for the endogenous variables*, and
- Λ_y is a $q \times k$ matrix of factor loadings relating y to η .

Let's also denote the two variance-covariance matrices, $\Theta_\delta(p \times p)$ and $\Theta_\epsilon(q \times q)$, representing the variance-covariance matrices among the measurement errors δ and ϵ , respectively. The third equation describing the LISREL path model component as relationships among latent variables includes:

- $B_{k \times k}$ a matrix of path coefficients describing the *relationships among endogenous latent variables*,
- $\Gamma_{k \times j}$ as a matrix of path coefficients representing the *linear effects of exogenous variables on endogenous variables*,
- $\zeta_{k \times 1}$ as a vector of *errors of endogenous variables*, and the corresponding two variance-covariance matrices $\Phi_{j \times j}$ of the *latent exogenous variables*, and
- $\Psi_{k \times k}$ of the *errors of endogenous variables*.

The basic statistic for a typical SEM implementation is based on covariance structure modeling and model fitting relies on optimizing an objective function, $\min\{f(\Sigma, S)\}$, representing the difference between the model-implied variance-covariance matrix, Σ , predicted from the causal and non-causal associations specified in the model, and the corresponding observed variance-covariance matrix S , which is estimated from observed data. The objective function, $f(\Sigma, S)$ can be estimated as shown below, see Shipley 2016.

In general, causation implies correlation, suggesting that if there is a causal relationship between two variables, there must also be a systematic relationship between them. Specifying a set of theoretical causal paths, we can reconstruct the model-implied variance-covariance matrix, Σ , from total effects and unanalyzed associations. The LISREL strategy specifies the following mathematical representation:

$$\Sigma = \begin{vmatrix} \Lambda_y A (\Gamma \Phi \Gamma' + \Psi) A' \Lambda'_y + \Theta_\epsilon & \Lambda_y A \Gamma \Phi \Lambda'_x \\ \Lambda_x \Phi \Gamma' A' \Lambda'_y & \Lambda_x \Phi \Lambda'_x + \Theta_\delta \end{vmatrix},$$

where $A = (I - B)^{-1}$. This representation of Σ does not involve the observed and latent exogenous and endogenous variables, x, y, ξ, η . Maximum likelihood estimation (MLE) may be used to obtain the Σ parameters via iterative searches for a set of optimal parameters minimizing the element-wise deviations between Σ and S .

The process of optimizing the objective function $f(\Sigma, S)$ can be achieved by computing the log likelihood ratio, i.e., comparing the likelihood of a given fitted model to the likelihood of a perfectly fit model. MLE estimation requires

multivariate normal distribution for the endogenous variables and Wishart distribution for the observed variance-covariance matrix, S .

Using MLE estimation simplifies the objective function to:

$$f(\Sigma, S) = \ln |\Sigma| + \text{tr}(S \times \Sigma^{-1}) - \ln |S| - \text{tr}(SS^{-1}),$$

where $\text{tr}()$ is the trace of a matrix. The optimization of $f(\Sigma, S)$ also requires independent and identically distributed observations and positive definite matrices, Σ, S . The iterative MLE optimization generates estimated variance-covariance matrices and path coefficients for the specified model. More details on model assessment (using Root Mean Square Error of Approximation, RMSEA, and Goodness of Fit Index) and the process of defining a priori SEM hypotheses are available in Lam & Maguire, 2012.

19.2.2 SEM Components

The R Lavaan package uses the following SEM syntax, Table 19.1, to represent relationships between variables. We can follow the following table to specify Lavaan models:

For example in R we can write the following model

```
model<-
  ' # regressions
```

$$\begin{aligned} y1 + y2 &\sim f1 + f2 + x1 + x2 \\ f1 &\sim f2 + f3 \\ f2 &\sim f3 + x1 + x2 \end{aligned}$$

```
# latent variable definitions
```

$$\begin{aligned} f1 &= \sim y1 + y2 + y3 \\ f2 &= \sim y4 + y5 + y6 \\ f3 &= \sim y7 + y8 + y9 + y10 \end{aligned}$$

```
# variances and covariances
```

$$\begin{aligned} y1 &\sim\sim y1 \\ y1 &\sim\sim y2 \\ f1 &\sim\sim f2 \end{aligned}$$

```
# intercepts
```

$$\begin{aligned} y1 &\sim 1 \\ f1 &\sim 1 \end{aligned}$$

Note that the two " " symbols (in the beginning and ending of a model description) are very important in the R-syntax.

Table 19.1 Lavaan syntax for specifying the relations between variables and their variance-covariance structure

Formula type	Operator	Explanation
Latent variable definition	$=\sim$	Is measured by
Regression	\sim	Is regressed on
(Residual) (co)variance	$\sim\sim$	Is correlated with
Intercept	~ 1	Intercept

19.2.3 Case Study – Parkinson’s Disease (PD)

Let’s use the PPMI dataset in our class file as an example to illustrate SEM model fitting.

Step 1 – Collecting Data

The Parkinson’s Disease Data represents a realistic simulation case-study to examine associations between clinical, demographic, imaging and genetics variables for Parkinson’s disease. This is an example of Big Data for investigating important neurodegenerative disorders.

Step 2 – Exploring and Preparing the Data

Now, we can import the dataset into R and recode the ResearchGroup variable into a binary variable.

```
par(mfrow=c(1, 1))
PPMI<-read.csv("https://umich.instructure.com/files/330397/download?download_
_frd=1")
summary(PPMI)

##      FID_IID      L_insular_cortex_ComputeArea L_insular_cortex_Volume
##  Min. :3001  Min.   : 50.03  Min.   : 22.63
##  1st Qu.:3272  1st Qu.:1976.88  1st Qu.: 4881.36
##  Median :3476   Median :2498.65  Median : 7236.76
##  Mean   :3534   Mean   :2255.20  Mean   : 6490.84
##  3rd Qu.:3817  3rd Qu.:2744.05  3rd Qu.: 8405.43
##  Max.   :4139   Max.   :3650.81   Max.   :13499.92
...
##      UPDRS_part_I      UPDRS_part_II      UPDRS_part_III      time_visit
##  Min.   : 0.000  Min.   : 0.000  Min.   : 0.00  Min.   : 0.00
##  1st Qu.: 0.000  1st Qu.: 2.000  1st Qu.:12.00  1st Qu.: 8.25
##  Median : 1.000  Median : 5.000  Median :20.00  Median :21.00
##  Mean   : 1.286  Mean   : 6.087  Mean   :19.44  Mean   :23.50
##  3rd Qu.: 2.000  3rd Qu.: 9.000  3rd Qu.:27.00  3rd Qu.:37.50
##  Max.   :13.000  Max.   :28.000  Max.   :61.00  Max.   :54.00
##  NA's   :549      NA's   :553      NA's   :554

PPMI$ResearchGroup<-ifelse(PPMI$ResearchGroup=="Control", "1", "0")
```

Correlations

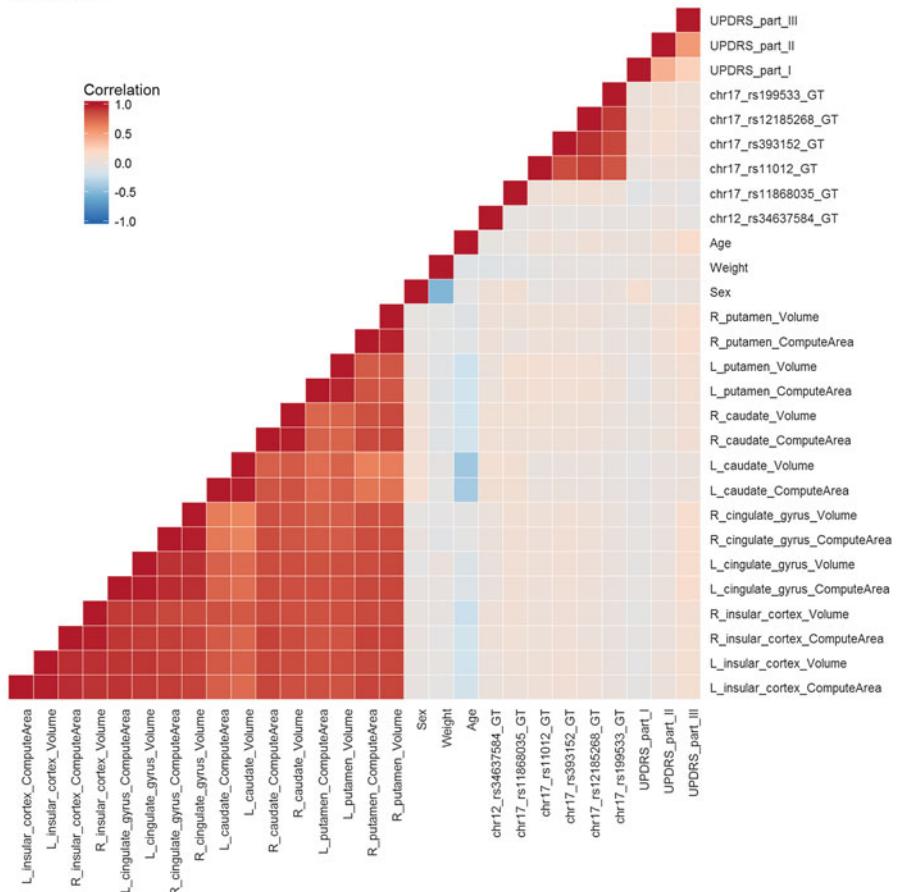


Fig. 19.15 Pair-wise correlation structure of the Parkinson's disease (PPMI) data.

This large dataset has 1,746 observations and 31 variables with missing data in some of them. A lot of the variables are highly correlated. You can inspect high correlation using *heat maps*, which reorders these covariates according to correlations to illustrate clusters of high-correlations (Fig. 19.15).

```
pp_heat <- PPMI[complete.cases(PPMI), -20]
corr_mat = cor(pp_heat)
# Remove upper triangle
corr_mat_lower = corr_mat
corr_mat_lower[upper.tri(corr_mat_lower)] = NA
# Melt correlation matrix and make sure order of factor variables is correct
corr_mat_melted = melt(corr_mat_lower)
colnames(corr_mat_melted) <- c("Var1", "Var2", "value")
corr_mat_melted$Var1 = factor(corr_mat_melted$Var1, levels=colnames(corr_mat))
corr_mat_melted$Var2 = factor(corr_mat_melted$Var2, levels=colnames(corr_mat))
```

```
# Plot
corr_plot = ggplot(corr_mat_melted, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile(color='white') +
  scale_fill_distiller(limits=c(-1, 1), palette='RdBu', na.value='white',
                       name='Correlation') +
  ggtitle('Correlations') +
  coord_fixed(ratio=1) +
  theme_minimal() +
  scale_y_discrete(position="right") +
  theme(axis.text.x=element_text(angle=45, vjust=1, hjust=1),
        axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        panel.grid.major=element_blank(),
        Legend.position=c(0.1,0.9),
        Legend.justification=c(0,1))
corr_plot
```

And here are some specific correlations

```
cor(PPMI$L_insular_cortex_ComputeArea, PPMI$L_insular_cortex_Volume)
## [1] 0.9837297
cor(PPMI$UPDRS_part_I, PPMI$UPDRS_part_II, use = "complete.obs")
## [1] 0.4027434
cor(PPMI$UPDRS_part_II, PPMI$UPDRS_part_III, use = "complete.obs")
## [1] 0.5326681
```

One way to solve this substantial multivariate correlation issue is to create some latent variables. We can consider the following model.

```
model1<-
  '
  Imaging =~ L_cingulate_gyrus_ComputeArea + L_cingulate_gyrus_Volume+R_c
  ingulate_gyrus_ComputeArea+R_cingulate_gyrus_Volume+R_insular_cortex_Compute
  Area+R_insular_cortex_Volume
  UPDRS=~UPDRS_part_I+UPDRS_part_II+UPDRS_part_III
  DemoGeno =~ Weight+Sex+Age
  '
  ResearchGroup ~ Imaging + DemoGeno + UPDRS
  '
```

Here we try to create three latent variables: Imaging, DemoGeno, and UPDRS. Let's fit a SEM model using `cfa()`, a confirmatory factor analysis function. Before fitting the data, we need to scale them. However, we don't need to scale our binary response variable. We can use the following code for normalizing the data.

```
mydata<-scale(PPMI[, -20])
mydata<-data.frame(mydata, PPMI$ResearchGroup)
colnames(mydata)[31]<- "ResearchGroup"
```

Step 3 – Fitting a Model on the Data

Now, we can start to build the model. The `cfa()` function we will use is part of the `lavaan` package.

```
# install.packages("lavaan")
library(lavaan)

fit<-cfa(model1, data=mydata, missing = 'FIML')
```

Here we can see some warning messages. Both our covariance and error term matrices are not positive definite. Non-positive definite matrices can cause the estimates of our model to be biased. There are many factors that can lead to this problem. In this case, we might create some latent variables that are not a good fit for our data. Let's try to delete the `DemoGeno` latent variable. We can add `Weight`, `Sex`, and `Age` directly to the regression model.

```
model2 <-
  '
# (1) Measurement Model
Imaging =~ L_cingulate_gyrus_ComputeArea + L_cingulate_gyrus_Volume+R_cing
ulate_gyrus_ComputeArea+R_cingulate_gyrus_Volume+R_insular_cortex_ComputeAre
a+R_insular_cortex_Volume
UPDRS =~ UPDRS_part_I +UPDRS_part_II + UPDRS_part_III
# (2) Regressions
ResearchGroup ~ Imaging + UPDRS +Age+Sex+Weight
'
```

When fitting `model2`, the warning messages are gone. We can see that falsely adding a latent variable can cause those matrices to be not positive definite. Currently, the `lavaan` functions `sem()` and `cfa()` are the same.

```
fit<-cfa(model2, data=mydata, missing = 'FIML')
summary(fit, fit.measures=TRUE)
## Lavaan (0.5-23.1097) converged normally after 107 iterations
##
##  Number of observations                           1764
##  Number of missing patterns                      4
##  Estimator                                         ML
##  Minimum Function Test Statistic                7714.119
##  Degrees of freedom                            60
##  P-value (Chi-square)                           0.000
##
##  Model test baseline model:
##  Minimum Function Test Statistic                30237.866
##  Degrees of freedom                            75
##  P-value                                         0.000
```

```

## 
## User model versus baseline model:
## 
## Comparative Fit Index (CFI)          0.746
## Tucker-Lewis Index (TLI)            0.683
## 
## Loglikelihood and Information Criteria:
## 
## Loglikelihood user model (H0)        NA
## Loglikelihood unrestricted model (H1) NA
## 
## Number of free parameters          35
## Akaike (AIC)                      NA
## Bayesian (BIC)                    NA
## 
## Root Mean Square Error of Approximation:
## 
## RMSEA                                0.269
## 90 Percent Confidence Interval       0.264 0.274
## P-value RMSEA <= 0.05                0.000
## 
## Standardized Root Mean Square Residual:
## 
## SRMR                                0.052
## 
## Parameter Estimates:
## 
## Information                         Observed
## Standard Errors                      Standard
## 
## Latent Variables:
## 
##             Estimate  Std.Err  z-value  P(>|z|/|)
## Imaging =~
##   L_cnglt_gyr_CA    1.000
##   L_cnglt_gyrs_V   0.994   0.004  260.366  0.000
##   R_cnglt_gyr_CA   0.961   0.007  134.531  0.000
##   R_cnglt_gyrs_V   0.955   0.008  126.207  0.000
##   R_nsrl_crtx_CA   0.930   0.009  101.427  0.000
##   R_nsrl_crtx_VL   0.920   0.010   94.505  0.000
## 
## UPDRS =~
##   UPDRS_part_I     1.000
##   UPDRS_part_II    1.890   0.177   10.699  0.000
##   UPDRS_part_III   2.345   0.248    9.468  0.000
## 
## Regressions:
## 
##             Estimate  Std.Err  z-value  P(>|z|/|)
## ResearchGroup ~
##   Imaging        0.008   0.010   0.788   0.431
##   UPDRS        -0.828   0.080  -10.299  0.000
##   Age          0.019   0.009   2.121   0.034
##   Sex          -0.010   0.010  -0.974   0.330
##   Weight        0.005   0.010   0.481   0.631

```

```

## 
## Covariances:
##           Estimate Std.Err  z-value P(>|z|)
## Imaging ~~
##   UPDRS      0.059   0.014   4.361   0.000
## 
## Intercepts:
##           Estimate Std.Err  z-value P(>|z|)
## .L_cnglt_gyr_CA -0.000   0.024  -0.001  1.000
## .L_cnglt_gyrs_V -0.000   0.024  -0.001  1.000
## .R_cnglt_gyr_CA -0.000   0.024  -0.001  1.000
## .R_cnglt_gyrs_V -0.000   0.024  -0.001  1.000
## .R_nsLr_crtx_CA -0.000   0.024  -0.001  1.000
## .R_nsLr_crtx_VL -0.000   0.024  -0.001  1.000
## .UPDRS_part_I   -0.135   0.032  -4.225  0.000
## .UPDRS_part_II  -0.255   0.033  -7.621  0.000
## .UPDRS_part_III -0.317   0.034  -9.181  0.000
## .ResearchGroup   1.290   0.011  119.239 0.000
##   Imaging       0.000
##   UPDRS        0.000
## 
## Variances:
##           Estimate Std.Err  z-value P(>|z|)
## .L_cnglt_gyr_CA 0.006   0.001   9.641  0.000
## .L_cnglt_gyrs_V 0.019   0.001  23.038 0.000
## .R_cnglt_gyr_CA 0.083   0.003  27.917 0.000
## .R_cnglt_gyrs_V 0.093   0.003  27.508 0.000
## .R_nsLr_crtx_CA 0.141   0.005  28.750 0.000
## .R_nsLr_crtx_VL 0.159   0.006  28.728 0.000
## .UPDRS_part_I   0.877   0.038  23.186 0.000
## .UPDRS_part_II  0.561   0.033  16.873 0.000
## .UPDRS_part_III 0.325   0.036  9.146  0.000
## .ResearchGroup   0.083   0.006  14.808 0.000
##   Imaging       0.993   0.034  29.509 0.000
##   UPDRS        0.182   0.035  5.213  0.000

```

19.2.4 Outputs of Lavaan SEM

In the output of our model, we have information about how to create these two latent variables (Imaging, UPDRS) and the estimated regression model. Specifically, it gives the following information.

1. First six lines are called the header contains the following information:
 - Lavaan version number.
 - Lavaan convergence information (normal or not), and #number of iterations needed.
 - The number of observations that were effectively used in the analysis.
 - The estimator that was used to obtain the parameter values (here: ML).
 - The model test statistic, the degrees of freedom, and a corresponding p-value.
2. Next, we have the Model test baseline model and the value for the SRMR

3. The last section contains the parameter estimates, standard errors (if the information matrix is expected or observed, and if the standard errors are standard, robust, or based on the bootstrap). Then, it tabulates all free (and fixed) parameters that were included in the model. Typically, first the latent variables are shown, followed by covariances and (residual) variances. The first column (Estimate) contains the (estimated or fixed) parameter value for each model parameter; the second column (Std.err) contains the standard error for each estimated parameter; the third column (Z-value) contains the Wald statistic (which is simply obtained by dividing the parameter value by its standard error); and the last column contains the p-value for testing the null hypothesis that the parameter equals zero in the population.

19.3 Longitudinal Data Analysis-Linear Mixed Models

As mentioned earlier, longitudinal studies take measurements for the same individual repeatedly through a period of time. Under this setting, we can measure the change after a specific treatment. However, the measurements for the same individual may be correlated with each other. Thus, we need special models that deal with this type of internal multivariate dependencies.

If we use the latent variable UPDRS (created in the output of SEM model) rather than the research group as our response we can obtain a longitudinal analysis model. In longitudinal analysis, time is often an important model variable.

19.3.1 Mean Trend

According to the output of model fit, our latent variable UPDRS is a combination of three observed variables-UPDRS_part_I, UPDRS_part_II, and UPDRS_part_III. We can visualize how average UPDRS values differ among the research groups over time.

```
mydata$UPDRS<-mydata$UPDRS_part_I+1.890*mydata$UPDRS_part_II+2.345*mydata$UPDRS_part_III
mydata$Imaging<-mydata$L_cingulate_gyrus_ComputeArea +0.994*mydata$L_cingulate_gyrus_Volume+0.961*mydata$R_cingulate_gyrus_ComputeArea+0.955*mydata$R_cingulate_gyrus_Volume+0.930*mydata$R_insular_cortex_ComputeArea+0.920*mydata$R_insular_cortex_Volume
```

The above code stores the latent UPDRS and Imaging variables into mydata. By now, we are experienced with using the package `ggplot2` for data visualization. Now, we will use it to set the x and y axes as time and UPDRS, and then display the trend of the individual level UPDRS.

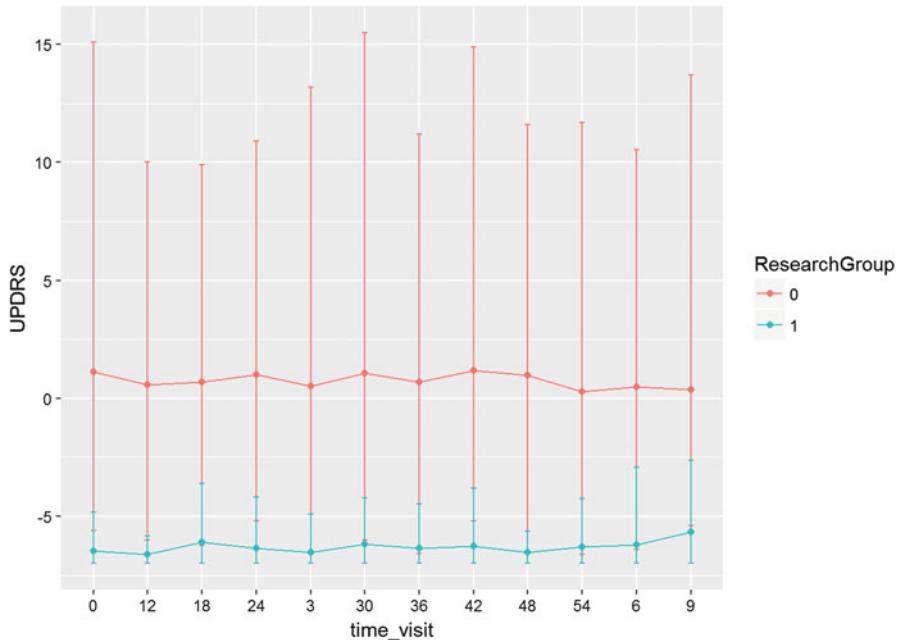


Fig. 19.16 Average UPDRS scores of the two cohorts in the PPMI dataset, patients (1) and controls (0)

```
require(ggplot2)
p<-ggplot(data=mydata, aes(x=time_visit, y=UPDRS, group=FID_IID))
dev.off()
p+geom_point()+geom_line()
```

This graph is a bit messy without a clear pattern emerging. Let's see if group-level graphs may provide more intuition. We will use the `aggregate()` function to get the mean, minimum and maximum of UPDRS for each time point. Then, we will use separate color for the two research groups and examine their mean trends (Fig. 19.16).

```
ppmi.mean<-aggregate(UPDRS~time_visit+ResearchGroup, FUN = mean, data=
mydata[, c(30, 31, 32)])
ppmi.min<-aggregate(UPDRS~time_visit+ResearchGroup, FUN = min, data=
mydata[, c(30, 31, 32)])
ppmi.max<-aggregate(UPDRS~time_visit+ResearchGroup, FUN = max, data=
mydata[, c(30, 31, 32)])
ppmi.boundary<-merge(ppmi.min, ppmi.max, by=c("time_visit", "ResearchGroup"))
ppmi.all<-merge(ppmi.mean, ppmi.boundary, by=c("time_visit", "ResearchGroup"))
pd <- position_dodge(0.1)
p1<-ggplot(data=ppmi.all, aes(x=time_visit, y=UPDRS, group=ResearchGroup,
colour=ResearchGroup))
p1+geom_errorbar(aes(ymin=UPDRS.x, ymax=UPDRS.y, width=0.1))+geom_point()+
geom_line()
```

Despite slight overlaps in some lines, the resulting graph illustrates better the mean differences between the two cohorts. The control group (1) appears to have relative lower means and tighter ranges compared to the PD patient group (0). However, we need further data interrogation to determine if this visual (EDA) evidence translates into statistically significant group differences.

Generally speaking we can always use the *General Linear Modeling (GLM)* framework. However, GLM may ignore the individual differences. So, we can try to fit a *Linear Mixed Model (LMM)* to incorporate different intercepts for each individual participant. Consider the following GLM:

$$\begin{aligned} UPDRS_{ij} \sim & \beta_0 + \beta_1 * Imaging_{ij} + \beta_2 * ResearchGroup_i + \beta_3 * timeVisit_j \\ & + \beta_4 * ResearchGroup_i * timeVisit_j + \beta_5 * Age_i + \beta_6 * Sex_i \\ & + \beta_7 * Weight_i + \epsilon_{ij}. \end{aligned}$$

If we fit a different intercept, b_i , for each individual (indicated by FID_IID), we obtain the following LMM model:

$$\begin{aligned} UPDRS_{ij} \sim & \beta_0 + \beta_1 * Imaging + \beta_2 * ResearchGroup + \beta_3 * timeVisit_j \\ & + \beta_4 * ResearchGroup_i * timeVisit_j + \beta_5 * Age_i + \beta_6 * Sex_i \\ & + \beta_7 * Weight_i + b_i + \epsilon_{ij}. \end{aligned}$$

The LMM actually has two levels:

Stage 1

$$Y_i = Z_i \beta_i + \epsilon_i,$$

where both Z_i and β_i are matrices.

Stage 2

The second level allows fitting random effects in the model.

$$\beta_i = A_i * \beta + b_i.$$

So, the full model in matrix form would be:

$$Y_i = X_i * \beta + Z_i * b_i + \epsilon_i.$$

In this case study, we only consider random intercept and avoid including random slopes, however the model can indeed be extended. In other words, $Z_i = 1$ in our simple model. Let's compare the two models (GLM and LMM). One R package implementing LMM is `lme4`.

```
#install.packages("lme4")
#install.packages("arm")
library(lme4)
library(arm)

#GLM
model.glm<-glm(UPDRS~Imaging+ResearchGroup*time_visit+Age+Sex+Weight, data=my
data)
summary(model.glm)
```

```

## 
## Call:
## glm(formula = UPDRS ~ Imaging + ResearchGroup * time_visit +
##      Age + Sex + Weight, data = mydata)
## 
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max 
## -7.6065 -2.4581 -0.3159  1.8328 14.9746 
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)            0.70000   0.10844   6.455 1.57e-10 ***
## Imaging                0.03834   0.01893   2.025  0.0431 *  
## ResearchGroup1        -6.93501   0.33445  -20.736 < 2e-16 ***
## time_visit              0.05077   0.10843   0.468  0.6397    
## Age                     0.54171   0.10839   4.998 6.66e-07 ***
## Sex                      0.16170   0.11967   1.351  0.1769    
## Weight                  0.20980   0.11707   1.792  0.0734 .  
## ResearchGroup1:time_visit -0.06842   0.32970  -0.208  0.8356  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for gaussian family taken to be 12.58436)
## 
## Null deviance: 21049  on 1205  degrees of freedom
## Residual deviance: 15076  on 1198  degrees of freedom
##  (558 observations deleted due to missingness)
## AIC: 6486.6
## 
## Number of Fisher Scoring iterations: 2

#LMM
model.Lmm<-Lmer(UPDRS~Imaging+ResearchGroup*time_visit+Age+Sex+Weight+(time_visit|FID_IID), data=mydata)
summary(model.Lmm)

## Linear mixed model fit by REML ['LmerMod']
## Formula:
## UPDRS ~ Imaging + ResearchGroup * time_visit + Age + Sex + Weight +
##      (time_visit | FID_IID)
## Data: mydata
## 
## REML criterion at convergence: 5737.9
## 
## Scaled residuals:
##    Min      1Q  Median      3Q     Max 
## -3.2660 -0.4617 -0.0669  0.3575  4.6158 
## 
## Random effects:
##   Groups   Name        Variance Std.Dev. Corr  
##   FID_IID (Intercept) 7.8821   2.8075      
##   FID_IID time_visit  0.2454   0.4954   0.16  
##   Residual            3.1233   1.7673      
## Number of obs: 1206, groups: FID_IID, 440
## 
## Fixed effects:

```

```

##                                     Estimate Std. Error t value
## (Intercept)                 0.69803   0.16881   4.135
## Imaging                   0.04200   0.02669   1.574
## ResearchGroup1             -6.93136   0.34425  -20.135
## time_visit                 0.02799   0.06385   0.438
## Age                        0.47720   0.15065   3.168
## Sex                         0.18662   0.17212   1.084
## Weight                      0.24146   0.17075   1.414
## ResearchGroup1:time_visit -0.04785   0.30496  -0.157
##
## Correlation of Fixed Effects:
##              (Intr) Imaging RsrcG1 tm_vst Age      Sex      Weight
## Imaging      -0.059
## ResrchGrp1  -0.496  0.101
## time_visit   0.067 -0.002 -0.033
## Age           -0.028  0.128  0.045  0.002
## Sex            -0.029  0.014  0.048  0.006  0.140
## Weight         -0.015  0.046  0.022  0.006  0.125  0.522
## RsrchGrp1:_ -0.011 -0.053 -0.001 -0.209 -0.010 -0.005  0.000

display(model.Lmm)

## Lmer(formula = UPDRS ~ Imaging + ResearchGroup * time_visit +
##       Age + Sex + Weight + (time_visit | FID_IID), data = mydata)
##                                     coef.est  coef.se
## (Intercept)                 0.70      0.17
## Imaging                   0.04      0.03
## ResearchGroup1             -6.93      0.34
## time_visit                 0.03      0.06
## Age                        0.48      0.15
## Sex                         0.19      0.17
## Weight                      0.24      0.17
## ResearchGroup1:time_visit -0.05      0.30
##
## Error terms:
##   Groups   Name       Std.Dev.  Corr
##   FID_IID (Intercept) 2.81
##   FID_IID time_visit  0.50      0.16
##   Residual             1.77
##   ---
##   number of obs: 1206, groups: FID_IID, 440
##   AIC = 5761.9, DIC = 5702.5
##   deviance = 5720.2

```

Note that we use the notation `ResearchGroup*time_visit` that is identical to `ResearchGroup + time_visit + ResearchGroup*time_visit`. Here R will include both terms and their interaction into the model. According to the model outputs, the LMM model has a relatively smaller AIC. In terms of AIC, LMM may represent a better model fit than GLM.

19.3.2 Modeling the Correlation

In the summary of the LMM model, we can see a section called `Correlation of Fixed Effects`. The original model made no assumption about the correlation (unstructured correlation). In R, we usually have the following 4 types of correlation models.

- **Independence:** No correlation:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

- **Exchangeable:** Correlations are constant across measurements:

$$\begin{pmatrix} 1 & \rho & \rho \\ \rho & 1 & \rho \\ \rho & \rho & 1 \end{pmatrix}.$$

- **Autoregressive order 1(AR(1)):** Correlations are stronger for closer measurements and weaker for more distanced measurements:

$$\begin{pmatrix} 1 & \rho & \rho^2 \\ \rho & 1 & \rho \\ \rho^2 & \rho & 1 \end{pmatrix}.$$

- **Unstructured:** Correlation is different for each occasion:

$$\begin{pmatrix} 1 & \rho_{1,2} & \rho_{1,3} \\ \rho_{1,2} & 1 & \rho_{2,3} \\ \rho_{1,3} & \rho_{2,3} & 1 \end{pmatrix}.$$

In the LMM model, the output also seems unstructured. So, we needn't worry about changing the correlation structure. However, if the output under unstructured correlation assumption looks like an Exchangeable or AR(1) structure, we may consider changing the LMM correlation structure accordingly.

19.4 GLMM/GEE Longitudinal Data Analysis

If the response is a binary variable like `ResearchGroup`, we need to use General Linear Mixed Model (GLMM) instead of LMM. The marginal model of GLMM is called GEE. However, GLMM and GEE are actually different.

In situations where the responses are discrete, there may not be a uniform or systematic strategy for dealing with the joint multivariate distribution of $Y_i = \{(Y_{i1}, Y_{i2}, \dots, Y_{in})\}^T$. That's where the GEE method comes into play as it's based on the concept of estimating equations. It provides a general approach for analyzing discrete and continuous responses with marginal models.

GEE is applicable when:

1. β , a generalized linear model regression parameter, characterizes systematic variation across covariate levels,

2. The data represents repeated measurements, clustered data, multivariate response, and
3. The correlation structure is a nuisance feature of the data.

Notation

- Response variables: $\{Y_{i,1}, Y_{i,2}, \dots, Y_{i,n_i}\}$, where $i \in [1, N]$ is the index for clusters or subjects, and $j \in [1, n_i]$ is the index of the measurement within cluster/subject.
- Covariate vector: $\{X_{i,1}, X_{i,2}, \dots, X_{i,n_i}\}$.

The primary focus of GEE is the estimation of the **mean model**: $E(Y_{i,j} | X_{i,j}) = \mu_{i,j}$, where

$$g(\mu_{i,j}) = \beta_0 + \beta_1 X_{i,j}(1) + \beta_2 X_{i,j}(2) + \beta_3 X_{i,j}(3) + \dots + \beta_p X_{i,j}(p) = X_{i,j} \times \beta.$$

This mean model can be any generalized linear model. For example: $P(Y_{i,j} = 1 | X_{i,j}) = \pi_{i,j}$ (marginal probability, as we don't condition on any other variables):

$$g(\mu_{i,j}) = \ln \left(\frac{\pi_{i,j}}{1 - \pi_{i,j}} \right) = X_{i,j} \times \beta.$$

Since the data could be clustered (e.g., within subject, or within unit), we need to choose a correlation model. Let's introduce some notation:

$$V_{i,j} = \text{var}(Y_{i,j} | X_i), \\ A_i = \text{diag}(V_{i,j}),$$

the paired correlations:

$$\rho_{i,j,k} = \text{corr}(Y_{i,j}, Y_{i,k} | X_i),$$

the correlation matrix:

$$R_i = (\rho_{i,j,k}), \text{ for all } j \text{ and } k,$$

and the paired predictor-response covariances are:

$$V_i = \text{cov}(Y_i | X_i) = A_i^{1/2} R_i A_i^{1/2}.$$

Assuming different correlation structures in the data leads to alternative models, see the examples above.

Notes

- GEE is a semi-parametric technique because:
 - The specification of a mean model, $\mu_{i,j}(\beta)$, and a correlation model, $R_i(\alpha)$, does not identify a complete probability model for Y_i

- The model $\{\mu_{i,j}(\beta), R_i(\alpha)\}$ is semi-parametric since it only specifies the first two multivariate moments (mean and covariance) of Y_i . Higher order moments are not specified.
- Without an explicit likelihood function, to estimate the parameter vector β (and perhaps the covariance parameter matrix $R_i(\alpha)$) and perform a valid statistical inference that takes the dependence into consideration, we need to construct an unbiased estimating function:
- $D_i(\beta) = \frac{\partial \mu_i}{\partial \beta}$, the partial derivative, w.r.t. β , of the mean-model for subject i .
- $D_i(j, k) = \frac{\partial \mu_{i,j}}{\partial \beta_k}$, the partial derivative, w.r.t. β , the partial derivative, w.r.t. the k th regression coefficient (β_k), of the mean-model for subject i and measurement (e.g., time-point) j .

Estimating (cost) function:

$$U(\beta) = \sum_{i=1}^N D_i^T(\beta) V_i^{-1}(\beta, \alpha) \{Y_i - \mu_i(\beta)\}.$$

Solving the Estimating Equations leads to parameter estimating solutions:

$$0 = U(\hat{\beta}) = \sum_{i=1}^N \underbrace{D_i^T(\hat{\beta})}_{\text{scale}} \underbrace{(V_i^{-1}(\hat{\beta}, \alpha))}_{\text{variance weight}} \underbrace{\{Y_i - \mu_i(\hat{\beta})\}}_{\text{model mean}}.$$

Scale: A change of scale term transforming the scale of the mean, μ_i , to the scale of the regression coefficients (covariates).

Variance weight: The inverse of the variance-covariance matrix is used to weight in the data for subject i , i.e., giving more weight to differences between observed and expected values for subjects that contribute more information.

Model Mean: Specifies the mean model, $\mu_i(\beta)$, compared to the observed data, Y_i . This fidelity term minimizes the difference between actually-observed and mean-expected (within the i th cluster/subject). See also the SMHS EBook.

19.4.1 GEE Versus GLMM

There is a difference in the interpretation of the model coefficients between GEE and GLMM. The fundamental difference between GEE and GLMM is in the target of the inference: population-average vs. subject-specific. For instance, consider an example where the observations are dichotomous outcomes (Y), e.g., single Bernoulli trials or death/survival of a clinical procedure, that are grouped/clustered into hospitals and units within hospitals, with N additional demographic, phenotypic, imaging and genetics predictors. To model the failure rate between genders (males vs. females) in a hospital, where all patients are spread among different hospital units (or clinical teams), let Y represent the binary response (death or survival).

In GLMM, the model will be pretty similar with the LMM model.

$$\log\left(\frac{P(Y_{ij} = 1)}{P(Y_{ij} = 0)} | X_{ij}, b_i\right) = \beta_0 + \beta_1 x_{ij} + b_i + \epsilon_{ij}.$$

The only difference between GLMM and LMM in this situation is that GLMM used a *logit link* for the binary response.

With GEE, we don't have random intercept or slope terms.

$$\log\left(\frac{P(Y_{ij} = 1)}{P(Y_{ij} = 0)} | X_{ij}, b_i\right) = \beta_0 + \beta_1 x_{ij} + \epsilon_{ij}.$$

In the marginal model (GEE), we are ignoring differences among hospital-units and just aim to obtain population (hospital-wise) rates of failure (patient death) and its association with patient gender. The GEE model fit estimates the odds ratio representing the population-averaged (hospital-wide) odds of failure associated with patient gender.

Thus, parameter estimates ($\hat{\beta}$) from GEE and GLMM models may differ because they estimate different things.

Let's compare the results of the GLM and GLMM models for our PPMI dataset.

```
# install.packages("gee")
library(gee)
model.glm1<-glm(ResearchGroup~UPDRS+Imaging+Age+Sex+Weight, data = mydata, family="binomial")

model.glm1

##
## Call: glm(formula = ResearchGroup ~ UPDRS + Imaging + Age + Sex + Weight
##           , family = "binomial", data = mydata)
##
## Coefficients:
## (Intercept)      UPDRS        Imaging        Age        Sex
## -10.64144     -1.96707     0.03889     0.71562     0.19361
##      Weight
##      0.40606
##
## Degrees of Freedom: 1205 Total (i.e. Null); 1200 Residual
##   (558 observations deleted due to missingness)
## Null Deviance: 811.9
## Residual Deviance: 195.8      AIC: 207.8

#mydata1<-na.omit(mydata)
#attach(mydata1)
#model.gee<-gee(ResearchGroup~L_insular_cortex_ComputeArea+L_insular_cortex_
#Volume+ Sex + Weight + Age + chr17_rs11012_GT + chr17_rs199533_GT + UPDRS_pa
#rt_I + UPDRS_part_II + time_visit, id=FID_IID, data = mydata1, family=binomi
#al(link = logit))
```

```

model.gLmm<-glmer(ResearchGroup~UPDRS+Imaging+Age+Sex+Weight+(1|FID_IID), data=mydata, family="binomial")
display(model.gLmm)

## glmer(formula = ResearchGroup ~ UPDRS + Imaging + Age + Sex +
##        Weight + (1 | FID_IID), data = mydata, family = "binomial")
##          coef.est coef.se
## (Intercept) -86.63    32.07
## UPDRS        -16.78     6.27
## Imaging       0.59     0.61
## Age          6.04     2.41
## Sex          0.65     2.15
## Weight       6.12     3.76
##
## Error terms:
## Groups   Name        Std.Dev.
## FID_IID (Intercept) 40.72
## Residual             1.00
## ---
## number of obs: 1206, groups: FID_IID, 440
## AIC = 129.5, DIC = -114.1
## deviance = 0.7

```

In terms of AIC, the GLMM model is a lot better than the GLM model.

Try to apply some of these longitudinal data analytics on the fMRI data we discussed in Chap. 4 (Visualization).

19.5 Assignment: 19. Big Longitudinal Data Analysis

19.5.1 Imaging Data

Review the 3D/4D MRI imaging data discussion in Chap. 4. Extract the time courses of several time series at different 3D spatial locations, some near-by, and some farther apart (distant voxels). Then, apply time-series analyses, report findings, determine if near-by or farther-apart voxels may be more correlated.

Example of extracting time series from 4D fMRI data:

```

#See examples here: https://cran.r-project.org/web/packages/oro.nifti/vignettes/nifti.pdf

fMRIURL <- "http://socr.umich.edu/HTML5/BrainViewer/data/fMRI_FilteredData_4D.nii.gz"
fMRIFile <- file.path(tempdir(), "fMRI_FilteredData_4D.nii.gz")
download.file(fMRIURL, dest=fMRIFile, quiet=TRUE)
(fMRIVolume <- readNIFTI(fMRIFile, reorient=FALSE))
# dimensions: 64 x 64 x 21 x 180 ; 4mm x 4mm x 6mm x 3 sec

fMRIVolDims <- dim(fMRIVolume); fMRIVolDims
time_dim <- fMRIVolDims[4]; time_dim

hist(fMRIVolume)

```

```

# To examine the time course of a specific 3D voxel (say the one at x=30, y=30, z=15):
plot(fMRIVolume[30, 30, 10,], type='l', main="Time Series of 3D Voxel \n (x=30, y=30, z=15)", col="blue")

x1 <- c(1:180)
y1 <- Loess(fMRIVolume[30, 30, 10,] ~ x1, family = "gaussian")
lines(x1, smooth(fMRIVolume[30, 30, 10,]), col = "red", lwd = 2)
lines(ksmooth(x1, fMRIVolume[30, 30, 10,], kernel = "normal", bandwidth = 5), col = "green", lwd = 3)

```

19.5.2 Time Series Analysis

Use Google Web-Search Trends and Stock Market Data to:

- Plot time series for the variable `Job`.
- Apply TTR to smooth the original graph by month.
- Determine the differencing parameter.
- Decide the auto-regressive (AR) and moving average (MA) parameters.
- Build an ARIMA model, forecast the `Job` variable over the next year and evaluate this model.

19.5.3 Latent Variables Model

Use the Hand written English Letters data to:

- Explore the data and evaluate the correlations between covariates.
- Justify the application of a latent variable model.
- Apply proper data conversion and scaling.
- Fit a Structural Equation Model (SEM) using the `lavaan::cfa()` function for these data by adding proper latent variable.
- Summarize and interpret the outputs.
- Use the model you found above to fit *GEE* and *GLMM* models using the latent variable as response and compare the models using AIC. (Hint: add a fake variable as random effect for GLMM).

References

- Box GE, Jenkins GM, Reinsel GC, Ljung GM. Time series analysis: forecasting and control: John Wiley & Sons; 2015.
- Grace JB. Structural equation modeling and natural systems: Cambridge University Press; 2006. <http://idaejin.github.io/bcam-courses/neiker-2016/material/mixed-models/>
- Liang K-Y, Zeger S. Longitudinal data analysis using generalized linear models. *Biometrika*. 1986;73(1):13-22. doi: <https://doi.org/10.1093/biomet/73.1.13>.
- McCulloch CE, Neuhaus JM. Generalized linear mixed models: Wiley Online Library; 2013.
- McIntosh A, Gonzalez-Lima F. Structural equation modeling and its application to network analysis in functional brain imaging. *Human Brain Mapping*. 1994;2(1-2):2-22.
- Shipley B. Cause and correlation in biology: a user's guide to path analysis, structural equations and causal inference with R: Cambridge University Press; 2016.

Chapter 20

Natural Language Processing/Text Mining



As we have seen in the previous chapters, traditional statistical analyses and classical data modeling are applied to *relational data* where the observed information is represented by tables, vectors, arrays, tensors, or data-frames containing binary, categorical, original, or numerical values. Such representations provide incredible advantages (e.g., quick reference and de-reference of elements, search, discovery, and navigation), but also limit the scope of applications. Relational data objects are quite effective for managing information that is based only on existing attributes. However, when data science inference needs to utilize attributes that are not included in the relational model, alternative non-relational representations are necessary. For instance, imagine that our data object includes a free text feature (e.g., physician/nurse clinical notes, biospecimen samples) that contains information about medical condition, treatment or outcome. It's very difficult, or sometimes even impossible, to include the raw text into the automated data analytics, using classical procedures and statistical models available for relational datasets.

Natural Language Processing (NLP) and Text Mining (TM) refer to automated machine-driven algorithms for semantically mapping, extracting information, and understanding of (natural) human language. Sometimes, this involves extracting salient information from large amounts of unstructured text. To do so, we need to build semantic and syntactic mapping algorithms for effective processing of heavy text. Related to NLP/TM, the work we did in Chap. 8 showed a powerful text classifier using the naive Bayes algorithm.

In this Chapter, we will present more details about various text processing strategies in R. Specifically, we will present simulated and real examples of text processing and computing document term frequency (TF), inverse document frequency (IDF), cosine similarity transformation, and machine learning based sentiment analysis.

Live demos will show various NLP tasks directly in the browser intermediate stages of processing, as well as full text processing performing complete text analysis.

20.1 A Simple NLP/TM Example

Text mining or text analytics (TM/TA) examines large volumes of unstructured text (corpus), aiming to extract new information, discover context, identify linguistic motifs, or transform the text into a structured data format leading to derived quantitative data that can be further analyzed. Natural language processing (NLP) is one example of a TM analytical technique. Whereas TM's goal is to discover relevant contextual information, which may be unknown, hidden, or obfuscated, NLP is focused on linguistic analysis that trains a machine to interpret voluminous textual content. To decipher the semantics and ambiguities in human-interpretable language, NLP employs automatic summarization, tagging, disambiguation, extraction of entities and relations, pattern recognition, and frequency analyses. As of 2018, the total amount of information generated by the human race exceeds 6 zetta-bytes ($1ZB = 10^{21} = 2^{70}$ bytes), which is projected to top 50ZB by 2020. The amount of data we obtain, and record, doubles every 12–14 months (Kryder's law). A small fraction of this massive information ($<0.0001\%$ or $<1PB = 10^{15}$ bytes) represents newly written or transcribed text, including code. However, it is impossible (cf. efficiency, time, resources) for humans to read, synthesize, interpret and react to all this information without direct assistance of TM/NLP. The information content in text could be substantially higher than that of other information media. Remember that “a picture may be worth a thousand words”, yet, “a word may also be worth a thousand pictures”. As an example, the simple sentence “*The data science and predictive analytics textbook includes 23 Chapters*” takes 63 bytes to store as text; however, a color image showing this as printed text could reach 10 megabytes (MB), and an HD video of a speaker reading the same sentence could easily surpass 50 MB. Text mining and natural language processing may be used to automatically analyze and interpret written, coded, or transcribed content to assess news, moods, emotions, clinical notes, and biosocial trends related to specific topics.

In general, text analysis protocols involve:

- Construction of a document-term matrix (DTM) from the input documents, vectorizing the text, e.g., creating a map of single words or n-grams into a vector space. That is, the *vectorizer is a function mapping terms to indices*.
- Application of a model-based statistical analysis or a model-free machine learning technique for prediction, clustering, classification, similarity search, network/sentiment analysis, or forecasting using the DTM. This step also includes tuning and internally validating the performance of the method.
- Application and evaluation of the technique to new data.

We are going to demonstrate this protocol with a very simple example. Figure 20.1 points to a separate online demo.



Fig. 20.1 Live demo: Dynamic NLP demonstration

20.1.1 Define and Load the Unstructured-Text Documents

Let's create some documents we can use to illustrate the use of the `tm` package for text mining. The five documents below represent portions of the syllabi of five recent courses taught by the author:

- HS650: Data Science and Predictive Analytics (DSP)
- Bootcamp: Predictive Big Data Analytics using R
- HS 853: Scientific Methods for Health Sciences: Special Topics
- HS851: Scientific Methods for Health Sciences: Applied Inference, and
- HS550: Scientific Methods for Health Sciences: Fundamentals

We import the syllabi into several separate segments represented as documents.

- As an *exercise*, try to use the `rvest::read_html` method to load in the five course syllabi directly from the course websites listed above.

`doc1 <- "HS650: The Data Science and Predictive Analytics(DSPA) course (offered as a massive open online course, MOOC, as well as a traditional University of Michigan class) aims to build computational abilities, inferential thinking, and practical skills for tackling core data scientific challenges. It explores foundational concepts in data management, processing, statistical computing, and dynamic visualization using modern programming tools and agile web-services. Concepts, ideas, and protocols are illustrated through examples of real observational, simulated and research-derived datasets. Some prior quantitative experience in programming, calculus, statistics, mathematical models, or linear algebra will be necessary. This open graduate course will provide a general overview of the principles, concepts, techniques, tools and services for managing, harmonizing, aggregating, preprocessing, modeling, analyzing and interpreting large, multsource, incomplete, incongruent, and heterogeneous data (Big Data). The focus will be to expose students to common challenges related to handling Big Data and present the enormous opportunities and power associated with our ability to interrogate such complex datasets, extract useful information, derive knowledge, and provide actionable forecasting. Biomedical, healthcare, and social datasets will provide context for addressing specific driving challenges. Students will learn about modern data analytic techniques and develop skills for importing and exporting, cleaning and fusing, modeling and visualizing, analyzing and synthesizing complex datasets. The collaborative design, implementation, sharing and community validation of high throughput analytic workflows will be emphasized throughout the course."`

doc2 < -"Bootcamp: A week-long intensive Bootcamp focused on methods, techniques, tools, services and resources for big healthcare and biomedical data analytics using the open-source statistical computing software R. Morning sessions (3 hrs) will be dedicated to methods and technologies and applications. Afternoon sessions (3 hrs) will be for group-based hands-on practice and team work. Commitment to attend the full week of instruction (morning sessions) and self-guided work (afternoon sessions) is required. Certificates of completion will be issued only to trainees with perfect attendance that complete all work. This hands-on intensive graduate course (Bootcamp) will provide a general overview of the principles, concepts, techniques, tools and services for managing, harmonizing, aggregating, preprocessing, modeling, analyzing and interpreting Large, multi-source, incomplete, incongruent, and heterogeneous data (Big Data). The focus will be to expose students to common challenges related to handling Big Data and present the enormous opportunities and power associated with our ability to interrogate such complex datasets, extract useful information, derive knowledge, and provide actionable forecasting. Biomedical, healthcare, and social datasets will provide context for addressing specific driving challenges. Students will learn about modern data analytic techniques and develop skills for importing and exporting, cleaning and fusing, modeling and visualizing, analyzing and synthesizing complex datasets. The collaborative design, implementation, sharing and community validation of high-throughput analytic workflows will be emphasized throughout the course."

doc3 <- "HS 853: This course covers a number of modern analytical methods for advanced healthcare research. Specific focus will be on reviewing and using innovative modeling, computational, analytic and visualization techniques to address concrete driving biomedical and healthcare applications. The course will cover the 5 dimensions of Big-Data (volume, complexity, multiple scales, - multiple sources, and incompleteness). HS853 is a 4 credit hour course (3 lectures + 1 lab/discussion). Students will learn how to conduct research, employ and report on recent advanced health sciences analytical methods; read, comprehend and present recent reports of innovative scientific methods; apply a broad range of health problems; experiment with real Big-Data. Topics Covered include: Foundations of R, Scientific Visualization, Review of Multivariate and Mixed Linear Models, Causality/Causal Inference and Structural Equation Models, Generalized Estimating Equations, PCOR/CER methods Heterogeneity of Treatment Effects, Big-Data, Big-Science, Internal statistical cross-validation, Missing data, Genotype-Environment-Phenotype, associations, Variable selection (regularized regression and controlled/knockoff filtering), medical imaging, Databases/registries, Meta-analyses, classification methods, Longitudinal data and time-series analysis, Geographic Information Systems(GIS), Psychometrics and Rasch measurement model analysis, Bayesian inference, and Network Analysis."

doc3 <- "HS 851: This course introduces students to applied inference methods in studies involving multiple variables. Specific methods that will be discussed include Linear regression, analysis of variance, and different regression models. This course will emphasize the scientific formulation, analytical modeling, computational tools and applied statistical inference in diverse health-sciences problems. Data interrogation, modeling approaches, rigorous interpretation and inference will be emphasized throughout. HS851 is a 4 credit hour course (3 lectures + 1 lab/discussion). Students will learn how to:, Understand the commonly used statistical methods of published scientific papers, Conduct statistical calculations/analyses on available data, Use software tools to analyze specific case-studies data, Communicate advanced statistical concepts/techniques, Determine, explain and interpret assumptions and limitations. Topics Covered include Epidemiology, Correlation/SLR, and slope inference, 1-2 samples, ROC Curve, ANOVA, Non-parametric

inference, Cronbach's α , Measurement Reliability/Validity, Survival Analysis, Decision theory, CLT/LLNs - limiting results and misconceptions, Association Tests, Bayesian Inference, PCA/ICA/Factor Analysis, Point/Interval Estimation (CI) - MoM, MLE, Instrument performance Evaluation, Study/Research Critiques, Common mistakes and misconceptions in using probability and statistics, identifying potential assumption violations, and avoiding them."

doc5 <- "HS550: This course provides students with an introduction to probability reasoning and statistical inference. Students will learn theoretical concepts and apply analytic skills for collecting, managing, modeling, processing, interpreting and visualizing (mostly univariate) data. Students will learn the basic probability modeling and statistical analysis methods and acquire knowledge to read recently published health research publications. HS550 is a 4 credit hour course (3 lectures + 1 lab/discussion). Students will learn how to: Apply data management strategies to sample data files, Carry out statistical tests to answer common healthcare research questions using appropriate methods and software tools, Understand the core analytical data modeling techniques and their appropriate use Examples of Topics Covered, EDA/Charts, Ubiquitous variation, Parametric inference, Probability Theory, Odds Ratio/Relative Risk, Distributions, Exploratory data analysis, Resampling/Simulation, Design of Experiments, Intro to Epidemiology, Estimation, Hypothesis testing, Experiments vs. Observational studies, Data management (tables, streams, cloud, warehouses, DBs, arrays, binary, ASCII, handling, mechanics), Power, sample-size, effect-size, sensitivity, specificity, Bias/-Precision, Association vs. Causality, Rate-of-change, Clinical vs. Stat significance, Statistical Independence Bayesian Rule."

20.1.2 Create a New VCorpus Object

The VCorpus object includes all the text and some meta-data (e.g., indexing) about the entire text.

```
docs<-c(doc1, doc2, doc3, doc4, doc5)
class(docs)
## [1] "character"
```

We can make a VCorpus object using `tm` package. To complete this task, we need to know the source type. Here, `docs` has a vector with "character" class, so we should use `VectorSource()`. If it is a dataframe, we should use `DataframeSource()` instead. `VCorpus()` creates a *volatile corpus*, which is the data type used by the `tm` package for text mining.

```

library(tm)
doc_corpus<-VCorpus(VectorSource(docs))
doc_corpus

## <<VCorpus>>
## Metadata: corpus specific: 0, document Level (indexed): 0
## Content: documents: 5

doc_corpus[[1]]$content

## [1] "HS650: The Data Science and Predictive Analytics (DSP) course (offered as a massive open online course, MOOC, as well as a traditional University of Michigan class) aims to build computational abilities, inferential thinking, ... throughout the course."

```

This generates a list containing the information for the five documents we have created. Now we can apply `tm_map()` function on this object to preprocess the text. The goal is to automatically interpret the text and output more succinct information.

20.1.3 To-Lower Case Transformation

The text itself contains upper case letters as well as lower case letters. The first thing to do is to convert all characters to lower case.

```

doc_corpus<-tm_map(doc_corpus, tolower)
doc_corpus[[1]]

## [1] "hs650: the data science and predictive analytics (dsp) course (offered as a massive open online course, mooc, as well as a traditional university of michigan class) ... community validation of high-throughput analytic workflows will be emphasized throughout the course."

```

20.1.4 Text Pre-processing

Remove Stopwords

These documents contains a lot of "stopwords", or common words, that have important semantic meaning but low analytic value. We can remove these by the following command.

```

stopwords("english")

## [1] "i"          "me"         "my"         "myself"      "we"
## [6] "our"        "ours"        "ourselves"   "you"        "your"
## [11] "yours"      "yourself"    "yourselves"  "he"         "him"
## [16] "his"        "himself"    "she"        "her"        "hers"
...
## [171] "so"         "than"        "too"        "very"

```

```
doc_corpus<-tm_map(doc_corpus, removeWords, stopwords("english"))
doc_corpus[[1]]

## [1] "hs650: data science predictive analytics (dsp) course (offered
massive open online course, mooc, well traditional university michigan c
lass) aims build ..., sharing community validation high-throughput analytic
workflows will emphasized throughout course."
```

We removed all the stopwords specified in the stopwords ("english") list. You can always make your own stopword list and just use `doc_corpus<--tm_map(doc_corpus, removeWords, your_own_words_list)` to apply this list.

From the output of `doc1` we notice the removal of stopwords creates extra blank spaces. Thus, the next step would be to remove them.

```
doc_corpus<-tm_map(doc_corpus, stripWhitespace)
doc_corpus[[1]]

## [1] "hs650: data science predictive analytics (dsp) course (offered mass
ive open online course, mooc, well traditional university michigan class) ai
ms build computational abilities, ..., sharing community validation high-throu
ghput analytic workflows will emphasized throughout course."
```

Remove Punctuation

Now we notice the irrelevant punctuation in the text, which can be removed by using a combination of `tm_map()` and `removePunctuation()` functions.

```
doc_corpus<-tm_map(doc_corpus, removePunctuation)
doc_corpus[[2]]

## [1] "bootcamp weeklong intensive bootcamp focused methods techniques tool
s services resources big healthcare biomedical data analytics using opensour
ce statistical computing software r morning sessions 3 hrs ... collaborative d
esign implementation sharing community validation hightthroughput analytic wo
rkflows will emphasized throughout course"
```

The above `tm_map` commands changed the structure of our `doc_corpus` object. We may apply `PlainTextDocument` function if we need to convert it back to the original format.

```
doc_corpus<-tm_map(doc_corpus, PlainTextDocument)
```

Stemming: Removal of Plurals and Action Suffixes

Let's inspect the first three documents. We notice that there are some words ending with "ing", "es", or "s".

```
doc_corpus[[1]]$content
## [1] "hs650 data science predictive analytics dspa course offered massive
open online course mooc well traditional university michigan class aims buil
d computational abilities inferential ... validation hightthroughput analytic w
orkflows will emphasized throughout course"

doc_corpus[[2]]$content
## [1] "bootcamp weeklong intensive bootcamp focused methods techniques tool
s services resources big healthcare biomedical data analytics using opensour
ce statistical computing software r morning sessions 3 ... design implementat
ion sharing community validation hightthroughput analytic workflows will emph
asized throughout course"

doc_corpus[[3]]$content
## [1] "hs 853 course covers number modern analytical methods advanced healt
hcare research specific focus will reviewing using innovative modeling compu
tational analytic visualization ... information systems gis psychometrics rasc
h measurement model analysis mcmc sampling bayesian inference network analys
is"
```

If we have multiple terms that only differ in their endings (e.g., past, present, present-perfect-continuous tense), the algorithm will treat them differently because it does not understand language semantics, the way a human would. To make things easier for the computer, we can delete these endings by “stemming” documents. Remember to load the package `SnowballC` before using the function `stemDocument()`. The earliest stemmer was written by Julie Beth Lovins in 1968, which had great influence on all subsequent work. Currently, one of the most popular stemming approaches was proposed by Martin Porter and is used in `stemDocument()`, and you can read more on Porter algorithm [online](#).

```
# install.packages("SnowballC")
library(SnowballC)
doc_corpus<-tm_map(doc_corpus, stemDocument)
doc_corpus[[1]]$content
## [1] "hs650 data scienc predict analyt dspa cours offer massiv open onlin
cours mooc well tradit univers michigan class aim build comput abil inferent
i think practic skill tackl core data scientif ... fuse model visual analyz sy
nthes complex dataset collabor design implement share communiti valid highth
roughput analyt workflow will emphas throughout cours"
```

This stemming process has to be done after the `PlainTextDocument` function because `stemDocument` can only be applied to plain text.

20.1.5 Bags of Words

It's very useful to be able to tokenize text documents into n-grams, sequences of words, e.g., a 2-gram represents two-word phrases that appear together in order. This allows us to form bags of words and extract information about word ordering.

The *bag of words model* is a common way to represent documents in matrix form based on their term frequencies (TFs). We can construct an $n \times t$ document-term matrix (DTM), where n is the number of documents, and t is the number of unique terms. Each column in the DTM represents a unique term. For instance, the $(i, j)^{th}$ cell represents how many of term j are present in document i .

The basic bag of words model is invariant to ordering of the words within a document. Once we compute the DTM, we can use machine learning techniques to interpret the derived signature information contained in the resulting matrices.

20.1.6 Document Term Matrix

Now that the `doc_corpus` object is quite clean, we can make a document-term matrix to explore all the terms in the five initial documents. The document term matrix includes dummy variables that tell us if a specific term appears in a specific document.

```
doc_dtm<-TermDocumentMatrix(doc_corpus)
doc_dtm

## <<TermDocumentMatrix (terms: 329, documents: 5)>>
## Non-/sparse entries: 540/1105
## Sparsity           : 67%
## Maximal term Length: 27
## Weighting          : term frequency (tf)
```

The summary of document term matrix is informative. We have 329 different terms in the five documents. There are 540 non-zero and 1,105 sparse entries. Thus, the sparsity is $\frac{1105}{(540+1105)} \approx 67\%$, which measures the term sparsity across all documents. A high sparsity means that the terms are not repeated often among different documents.

Recall that we applied `PlainTextDocument` function to your `doc_corpus` object. This removed all document meta data. To relabel the documents in the document term matrix, we can use the following commands:

```
doc_dtm$dimnames$Docs<-as.character(1:5)
inspect(doc_dtm)

## <<TermDocumentMatrix (terms: 329, documents: 5)>>
## Non-/sparse entries: 540/1105
## Sparsity           : 67%
## Maximal term Length: 27
## Weighting          : term frequency (tf)
## Sample             :
##                 Docs
## Terms      1 2 3 4 5
## analyt    3 3 3 1 2
## cours     4 2 3 3 2
## data      7 5 2 3 6
## infer     0 0 2 6 2
## method    0 2 5 3 2
## model     3 2 4 3 3
## statist   2 1 1 5 4
## student   2 2 1 2 4
## use       2 2 1 3 2
## will      6 8 3 4 3
```

We might want to find and report the frequent terms using this document term matrix.

```
findFreqTerms(doc_dtm, lowfreq = 2)

## [1] "abil"          "action"        "address"       "advanc"
## [5] "afternoon"     "aggreg"        "analysi"       "analyt"
## [9] "analyz"        "appli"         "applic"        "appropri"
## [13] "associ"        "assumpt"       "attend"        "bayesian"
## [17] "big"           "bigdata"       "biomed"        "bootcamp"
## [21] "challeng"      "clean"         "collabor"      "common"
## [25] "communiti"     "complet"        "complex"       "comput"
## [29] "concept"       "conduct"        "context"       "core"
## [33] "cours"         "cover"         "credit"        "data"
## [37] "dataset"       "deriv"         "design"        "develop"
## [41] "drive"          "emphas"        "enorm"         "epidemiolog"
## [45] "equat"          "estim"         "exampl"        "experi"
## [49] "export"         "expos"         "extract"       "focus"
## [53] "forecast"       "foundat"       "fuse"          "general"
## [57] "graduat"        "hand"          "handl"         "harmon"
## [61] "health"          "healthcar"     "heterogen"     "highthroughput"
## [65] "hour"           "hrs"           "hs550"         "implement"
## [69] "import"          "includ"        "incomplet"     "incongru"
## [73] "infer"          "inform"        "innov"         "intens"
## [77] "interpret"      "interrog"      "knowledg"      "Labdiscuss"
## [81] "larg"            "Learn"         "Lectur"        "Limit"
## [85] "Linear"          "manag"         "measur"        "method"
## [89] "misconcept"     "model"         "modern"        "morn"
## [93] "multipl"        "multisourc"   "observ"        "open"
## [97] "opportun"       "overview"      "power"         "practic"
## [101] "preprocess"    "present"       "principl"      "probabl"
## [105] "problem"        "process"       "program"       "provid"
## [109] "publish"        "read"          "real"          "recent"
## [113] "regress"        "relat"         "report"        "research"
## [117] "review"         "sampl"         "scienc"        "scientif"
## [121] "servic"         "session"       "share"         "skill"
## [125] "social"         "softwar"       "specif"        "statist"
## [129] "student"        "studi"         "synthes"       "techniqu"
## [133] "test"           "theori"        "throughout"   "tool"
## [137] "topic"          "understand"   "use"           "valid"
## [141] "variabl"        "visual"        "will"          "work"
## [145] "workflow"
```

This gives us the terms that appear in at least two documents. High-frequency terms like `comput`, `statist`, `model`, `healthcar`, `learn` make perfect sense to be included as these syllabi describe courses that cover modeling, statistical and computational methods with applications to health sciences.

The `tm` package also provides the correlation between terms. Here is a mechanism to determine the words that are highly correlated with `statist`, $(\rho(statist, ?) \geq 0.8)$.

```
findAssocs(doc_dtm, "statist", corlimit = 0.8)

## $statist
## epidemiolog    publish      studi      theori  understand      appli
## 0.95          0.95        0.95      0.95    0.95        0.83
## test          0.80
```

20.2 Case-Study: Job Ranking

Let's explore some real datasets. First, we will import the 2011 USA Jobs Ranking Dataset from SOCR data archive.

```
library(rvest)

wiki_url <- read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_2011_US_JobsRanking")
html_nodes(wiki_url, "#content")

## {xml_nodeset (1)}
## [1] <div id="content" class="mw-body-primary" role="main">\n\t<a id="top"
...
job <- html_table(html_nodes(wiki_url, "table"))[[1]]
head(job)

##   Index          Job_Title Overall_Score Average_Income(USD)
## 1    1   Software_Engineer        60            87140
## 2    2   Mathematician          73            94178
## 3    3      Actuary            123            87204
## 4    4   Statistician           129            73208
## 5    5 Computer_Systems_Analyst        147            77153
## 6    6   Meteorologist          175            85210
##   Work_Environment Stress_Level Stress_Category Physical_Demand
## 1      150.00       10.40            1            5.00
## 2      89.72       12.78            1            3.97
## 3     179.44       16.04            1            3.97
## 4      89.52       14.08            1            3.95
## 5      90.78       16.53            1            5.08
## 6     179.64       15.10            1            6.98
##   Hiring_Potential
## 1      27.40
## 2      19.78
## 3      17.04
## 4      11.08
## 5      15.53
## 6      12.10
##
## Description
## 1 Researches_designs_develops_and_maintains_software_systems_along_with_h
## ardware_development_for_medical_scientific_and_industrial_purposes
## 2 Applies_mathematical_theories_and_formulas_to_teach_or_solve_problems_i
## n_a_business_educational_or_industrial_climate
## 3 Interprets_statistics_to_determine_probabilities_of_accidents_sickness
## _and_death_and_Loss_of_property_from_theft_and_natural_disasters
## 4 Tabulates_analyzes_and_interprets_the_numeric_results_of_experiments_
## and_surveys
## 5 Plans_and_develops_computer_systems_for_businesses_and_scientific_in
## stitutions
## 6 Studies_the_physical_characteristics_motions_and_processes_of_the_ear
## th's_atmosphere
```

Note that low indices represent jobs that in 2011 were highly desirable. Thus, in 2011, the most desirable job among the top 200 common jobs would be Software

Engineer. The aim of our case study is to explore the difference between the top 30 desirable jobs and the last 100 jobs in the list.

We will go through the same procedure as we did for the simple course syllabi example. The documents we will be using include the `Description` column (a vector) in the dataset.

20.2.1 Step 1: Make a VCorpus Object

```
jobCorpus<-VCorpus(VectorSource(job[, 10]))
```

20.2.2 Step 2: Clean the VCorpus Object

```
jobCorpus<-tm_map(jobCorpus, tolower)
for(j in seq(jobCorpus)){
  jobCorpus[[j]]<-gsub("_", " ", jobCorpus[[j]])
}
```

Here we used a loop to substitute "_" with blank space. This is because when we use `removePunctuation`, all the underline characters will disappear and there will be no separation between terms. In this situation, `gsub` will be the best choice to use.

```
jobCorpus<-tm_map(jobCorpus, removeWords, stopwords("english"))
jobCorpus<-tm_map(jobCorpus, removePunctuation)
jobCorpus<-tm_map(jobCorpus, stripWhitespace)
jobCorpus<-tm_map(jobCorpus, PlainTextDocument)
jobCorpus<-tm_map(jobCorpus, stemDocument)
```

20.2.3 Step 3: Build the Document Term Matrix

The Document Term Matrix (DTM) objects (`tm::DocumentTermMatrix`) contains a sparse term-document matrix, or document-term matrix, and attribute weights of the matrix.

First, make sure that we got a clean VCorpus object.

```
jobCorpus[[1]]$content
## [1] "research design develop maintain softwar system along hardwar develo
p medic scientif industri purpos"
```

Then, we can start to build the DTM and reassign the labels to the Docs.

```
dtm<-DocumentTermMatrix(jobCorpus)
dtm

## <<DocumentTermMatrix (documents: 200, terms: 846)>>
## Non-/sparse entries: 1818/167382
## Sparsity : 99%
## Maximal term Length: 15
## Weighting : term frequency (tf)

dtm$dimnames$Docs<-as.character(1:200)
inspect(dtm[1:10], 1:10)

## <<DocumentTermMatrix (documents: 10, terms: 10)>>
## Non-/sparse entries: 2/98
## Sparsity : 98%
## Maximal term Length: 7
## Weighting : term frequency (tf)
## Sample :
## Terms
## Docs 16wheel abnorm access accid accord account accur achiev act activ
## 1 0 0 0 0 0 0 0 0 0 0 0 0
## 10 0 0 0 0 0 0 0 0 0 0 0 0
## 2 0 0 0 0 0 0 0 0 0 0 0 0
## 3 0 0 0 0 1 0 0 0 0 0 0 0
## 4 0 0 0 0 0 0 0 0 0 0 0 0
## 5 0 0 0 0 0 0 0 0 0 0 0 0
## 6 0 0 0 0 0 0 0 0 0 0 0 0
## 7 0 0 0 0 0 0 0 0 0 0 0 0
## 8 0 0 0 0 0 1 0 0 0 0 0 0
## 9 0 0 0 0 0 0 0 0 0 0 0 0
```

Let's subset the dtm into the top 30 jobs and the bottom 100 jobs.

```
dtm_top30<-dtm[1:30, ]
dtm_bot100<-dtm[101:200, ]
dtm_top30

## <<DocumentTermMatrix (documents: 30, terms: 846)>>
## Non-/sparse entries: 293/25087
## Sparsity : 99%
## Maximal term Length: 15
## Weighting : term frequency (tf)

dtm_bot100

## <<DocumentTermMatrix (documents: 100, terms: 846)>>
## Non-/sparse entries: 870/83730
## Sparsity : 99%
## Maximal term Length: 15
## Weighting : term frequency (tf)
```

In this case, since the sparsity is very high, we can try to remove some words that rarely appear in the job descriptions.

```

dtms_top30<-removeSparseTerms(dtm_top30, 0.90)
dtms_top30

## <<DocumentTermMatrix (documents: 30, terms: 19)>>
## Non-/sparse entries: 70/500
## Sparsity           : 88%
## Maximal term Length: 10
## Weighting          : term frequency (tf)

dtms_bot100<-removeSparseTerms(dtm_bot100, 0.94)
dtms_bot100

## <<DocumentTermMatrix (documents: 100, terms: 14)>>
## Non-/sparse entries: 122/1278
## Sparsity           : 91%
## Maximal term Length: 10
## Weighting          : term frequency (tf)

```

Now, instead of 846 terms, we only have 19 that appear in the top 30 job descriptions (JDs) and 14 that appear in the bottom 100 JDs.

Similar to what we did in [Chap. 8](#), visualization of the terms-world clouds may be accomplished by combining the `tm` with `wordcloud` packages. First, we can count the term frequencies in the two document term matrices (Fig. [20.2](#)).

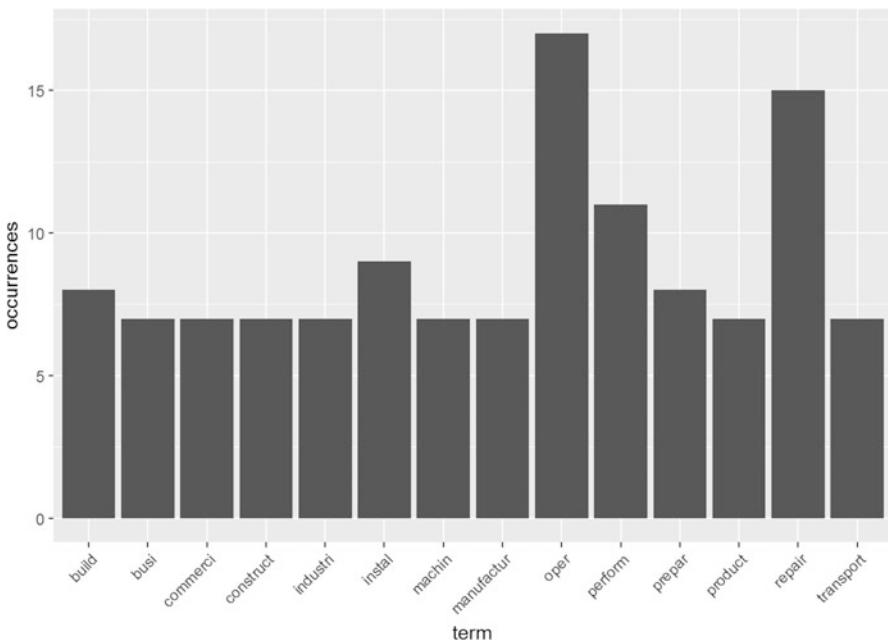


Fig. 20.2 Frequency plot of commonly occurring terms (bottom 100 jobs)

```

# Calculate the cumulative frequencies of words across documents and sort:
freq1<-sort(colSums(as.matrix(dtms_top30)), decreasing=T)
freq1

##      develop      assist      natur      studi      analyz      concern
##          6          5          5          5          4          4
##      individu      industri      physic      plan      busi      inform
##          4          4          4          4          3          3
##      institut      problem      research      scientif      theori      treatment
##          3          3          3          3          3          3
##      understand
##          3

freq2<-sort(colSums(as.matrix(dtms_bot100)), decreasing=T)
freq2

##      oper      repair      perform      instal      build      prepar
##          17          15          11          9          8          8
##      busi      commerci      construct      industri      machin      manufactur
##          7          7          7          7          7          7
##      product      transport
##          7          7

# Plot
wf=data.frame(term=names(freq2), occurrences=freq2)
library(ggplot2)

## 
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
## 
##     annotate

p <- ggplot(subset(wf, freq2>2), aes(term, occurrences))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))
p

```

Then, we apply the `wordcloud` function to the `freq` dataset (Figs. 20.3 and 20.4).

Fig. 20.3 Wordle plot of the frequently occurring terms in the top-30 jobs



Fig. 20.4 The appearance of the wordle plot may be customized as shown here for the bottom-100 jobs



```
library(wordCloud)
set.seed(123)
wordCloud(names(freq1), freq1)

# Color code the frequencies using an appropriate color map:
# Sequential palettes names include:
# Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples Rd
Pu Reds YlGn YlGnBu YlOrBr YlOrRd

# Diverging palettes include
# BrBG PiYG PRGn PuOr RdBu RdGy RdYlBu RdYlGn Spectral
wordCloud(names(freq2), freq2, min.freq=5, colors=brewer.pal(6, "Spectral"))
```

It is apparent that the top 30 jobs focus more on research or discovery of new things, and include frequent keywords like “study”, “nature”, and “analyze.” The bottom 100 jobs more focused of operating on existing objects, with frequent keywords like “operation”, “repair”, and “perform”.

20.2.4 Area Under the ROC Curve

In Chap. 14, we talked about the ROC curve. We can use document term matrices to build classifiers and use the area under the ROC curve (AUC) to evaluate those classifiers. Assume that we want to predict whether a job ranks in the top 30, i.e., the most desired jobs. The first task would be to create an indicator of high rank jobs (top 30). We can use the `ifelse()` function that we are already familiar with.

```
job$highrank<-ifelse(job$Index<30, 1, 0)
```

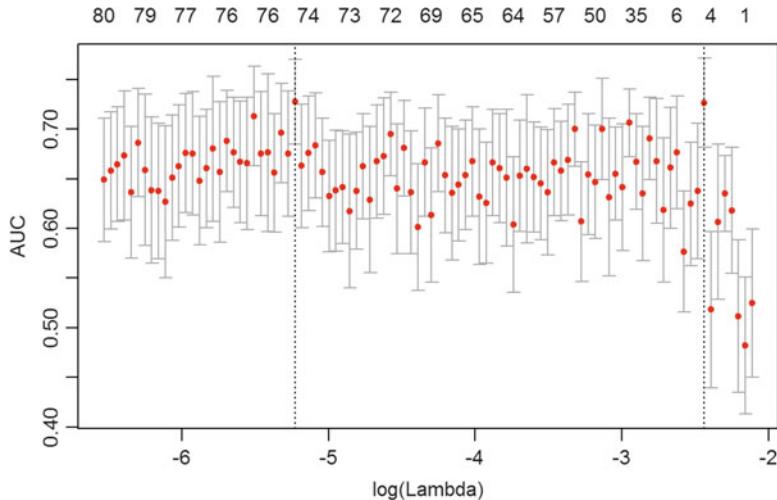


Fig. 20.5 The area under the curve (AUC) measures the performance of the cross-validated LASSO-regularized model of job-ranking against the magnitude of the regularization parameter (bottom axis), and the efficacy of the model selection, i.e., number of non-trivial coefficients (top axis). The vertical dash lines suggest an optimal range for the penalty term and the number of coefficients, see Chap. 18

Next we load the `glmnet` package to help us build the model and draw the corresponding graphs.

```
#install.packages("glmnet")
library(glmnet)
```

The function we will be using is the `cv.glmnet`, `cv` stands for cross-validation. Since the `highrank` variable is binary, we specify the option `family = 'binomial'`. Also, we want to use 10-fold CV method for re-sampling (Fig. 20.5).

```
set.seed(25)
fit <- cv.glmnet(x = as.matrix(dtm), y = job[['highrank']],
                  family = 'binomial',
                  # lasso penalty
                  alpha = 1,
                  # interested in the area under ROC curve
                  type.measure = "auc",
                  # 10-fold cross-validation
                  nfolds = 10,
                  # high value is less accurate, but has faster training
                  thresh = 1e-3,
                  # again lower number of iterations for faster training
                  maxit = 1e3)
plot(fit)
```

```
print(paste("max AUC =", round(max(fit$cvm), 4)))
## [1] "max AUC = 0.7276"
```

Here, x is a matrix and y is the response variable. The last line of code helps us select the best AUC among all models. The resulting $AUC \sim 0.73$ represents a relatively good prediction model for this small sample size.

20.3 TF-IDF

To enhance the performance of the DTM matrix, we introduce **TF-IDF (term frequency – inverse document frequency)**. Unlike pure frequency, TF-IDF measures the relative importance of a term. If a term appears in almost every document, the term will be considered common with a small weight. Alternatively, the rare terms would be considered more informational.

20.3.1 Term Frequency (TF)

TF is the ratio $\frac{\text{a term's occurrences in a document}}{\text{the number of occurrences of the most frequent word within the same document}}$. Symbolically,

$$TF(t, d) = \frac{f_d(t)}{\max_{w \in d} f_d(w)}.$$

20.3.2 Inverse Document Frequency (IDF)

The **TF** definition may allow high scores for irrelevant words that naturally show up often in a long text, even after triaging common words in a prior preprocessing step. The **IDF** attempts to rectify that. **IDF** represents the inverse of the share of the documents in which the regarded term can be found. The lower the number of documents containing the term, relative to the size of the corpus, the higher the term factor.

IDF involves a logarithm function, to temper the effective scoring penalty of showing up in two documents, which otherwise may be too extreme. Typically, the IDF for a term found in just one document is twice the IDF for another term found in two docs. The `ln()` function rectifies this bias of ranking in favor of rare terms, even if

the TF-factor may be high. It is rather unlikely that a term's relevance is only high in one doc and not all others.

$$IDF(t, D) = \ln \left(\frac{|D|}{|\{d \in D : t \in d\}|} \right).$$

20.3.3 TF-IDF

Both TF and IDF yield high scores for highly relevant terms. TF relies on local information (search over d), whereas IDF incorporates a more global perspective (search over D). The product $TF \times IDF$, gives the classical **TF-IDF** formula. However, alternative expressions may be formulated to get other univariate expressions using alternative weights for TF and IDF.

$$TF_IDF(t, d, D) = TF(t, d) \times IDF(t, D).$$

An example of an alternative TF-IDF metric can be defined by:

$$TF_IDF'(t, d, D) = \frac{IDF(t, D)}{|D|} + TF_IDF(t, d, D).$$

Let's make another DTM with TF-IDF weights and compare the differences between the *unweighted* and *weighted* DTM.

```
dtm.tfidf<-DocumentTermMatrix(jobCorpus, control = List(weighting=weightTfIdf))
dtm.tfidf

## <<DocumentTermMatrix (documents: 200, terms: 846)>>
## Non-/sparse entries: 1818/167382
## Sparsity : 99%
## Maximal term length: 15
## Weighting : term frequency - inverse document frequency (normalized) (tf-idf)

dtm.tfidf$dimnames$Docs <- as.character(1:200)
inspect(dtm.tfidf[1:9, 1:10])

## <<DocumentTermMatrix (documents: 9, terms: 10)>>
## Non-/sparse entries: 2/88
## Sparsity : 98%
## Maximal term length: 7
## Weighting : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample :
##     Terms
## Docs 16wheel abnorm access accid accord account accur achiev act
##   1     0     0     0 0.0000000 0.0000000     0     0     0     0
##   2     0     0     0 0.0000000 0.0000000     0     0     0     0
```

```

##   3      0      0      0 0.5536547 0.0000000      0      0      0      0
##   4      0      0      0 0.0000000 0.0000000      0      0      0      0
##   5      0      0      0 0.0000000 0.0000000      0      0      0      0
##   6      0      0      0 0.0000000 0.0000000      0      0      0      0
##   7      0      0      0 0.0000000 0.0000000      0      0      0      0
##   8      0      0      0 0.0000000 0.4321928      0      0      0      0
##   9      0      0      0 0.0000000 0.0000000      0      0      0      0
##   Terms
## Docs activ
##   1      0
##   2      0
##   3      0
##   4      0
##   5      0
##   6      0
##   7      0
##   8      0
##   9      0

inspect(dtm[1:9, 1:10])

## <<DocumentTermMatrix (documents: 9, terms: 10)>>
## Non-/sparse entries: 2/88
## Sparsity          : 98%
## Maximal term length: 7
## Weighting          : term frequency (tf)
## Sample             :
##   Terms
## Docs 16wheel abnorm access accid accord account accur achiev act activ
##   1      0      0      0      0      0      0      0      0      0      0      0
##   2      0      0      0      0      0      0      0      0      0      0      0
##   3      0      0      0      1      0      0      0      0      0      0      0
##   4      0      0      0      0      0      0      0      0      0      0      0
##   5      0      0      0      0      0      0      0      0      0      0      0
##   6      0      0      0      0      0      0      0      0      0      0      0
##   7      0      0      0      0      0      0      0      0      0      0      0
##   8      0      0      0      0      0      1      0      0      0      0      0
##   9      0      0      0      0      0      0      0      0      0      0      0

```

An inspection of the two different DTMs suggests that TF-IDF is not only counting the frequency but also assigning different weights to each term according to the importance of the term. Next, we are going to fit another model with this new DTM (dtm.tfidf) (Fig. 20.6).

```

set.seed(2)
fit1 <- cv.glmnet(x = as.matrix(dtm.tfidf), y = job[['highrank']],
                   family = 'binomial',
                   # lasso penalty
                   alpha = 1,
                   # interested in the area under ROC curve
                   type.measure = "auc",
                   # 10-fold cross-validation
                   nfolds = 10,
                   # high value is less accurate, but has faster training
                   thresh = 1e-3,
                   # again lower number of iterations for faster training
                   maxit = 1e3)
plot(fit1)

```

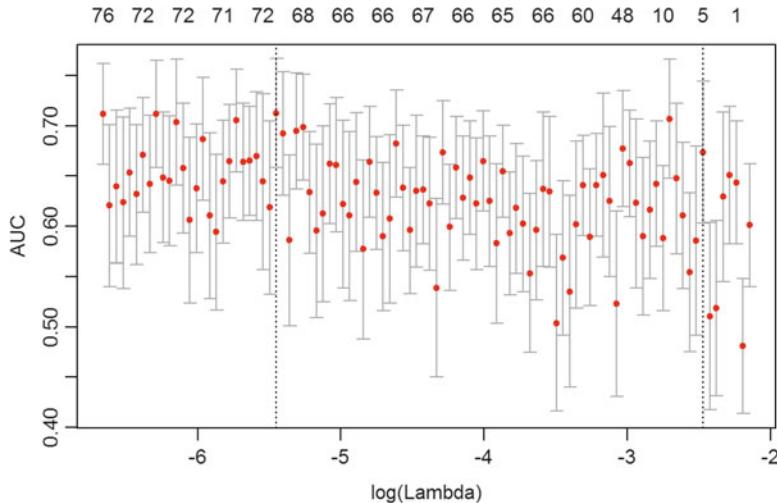


Fig. 20.6 AUC-based performance of the cross-validated LASSO-regularized model of job-ranking based on the new DTM (dtm.tfidf), see Fig. 20.5 and Chap. 18

```
print(paste("max AUC =", round(max(fit1$cvm), 4)))
## [1] "max AUC = 0.7125"
```

This output is about the same as the previous jobs ranking prediction classifier (based on the unweighted DTM). Due to random sampling, each run of the protocols may generate slightly different results. The idea behind using TF-IDF is that one would expect to get more unbiased estimates of word importance. If the document includes stopwords, like “the” or “one”, the DTM may distort the results, but TF-IDF may resolve some of these problems.

Next, we can report a more intuitive representation of the job ranking prediction reflecting the agreement of the binary (top-30 or not) classification between the real labels and the predicted labels. Notice that this applies only to the training data itself.

```
# Binarize the LASSO probability prediction
preffit1 <- predict(fit1, newx=as.matrix(dtm.tfidf), s="Lambda.min", type =
"class")
binPredfit1 <- ifelse(preffit1<0.5, 0, 1)
table(binPredfit1, job[["highrank"]])

##
## binPredfit1 0 1
## 0 171 0
## 1 0 29
```

Let’s try to predict the job ranking of several new (testing or validation) job descriptions (JDs). There are many job descriptions provided online that we can extract text from to predict the job ranking of the corresponding positions. Trying

several alternative job categories, e.g., some high-tech or fin-tech, and some manufacturing or construction jobs, may provide some intuition to the power of the jobs-classifier we built. Below, we will compare the JDs for the positions of *accountant*, *attorney*, and *machinist*.

```
# install.packages("text2vec"); install.packages("data.table")
library(text2vec)
library(data.table)

# Choose the JD for a PUBLIC ACCOUNTANTS 1430 (https://www.bls.gov/ocs/ocsjobde.htm)
xTestAccountant <- "Performs professional auditing work in a public accounting firm. Work requires at least a bachelor's degree in accounting. Participates in or conducts audits to ascertain the fairness of financial representations made by client companies. May also assist the client in improving accounting procedures and operations. Examines financial reports, accounting records, and related documents and practices of clients. Determines whether all important matters have been disclosed and whether procedures are consistent and conform to acceptable practices. Samples and tests transactions, internal controls, and other elements of the accounting system(s) as needed to render the accounting firm's final written opinion. As an entry level public accountant, serves as a junior member of an audit team. Receives classroom and on-the-job training to provide practical experience in applying the principles, theories, and concepts of accounting and auditing to specific situations. (Positions held by trainee public accountants with advanced degrees, such as MBA's are excluded at this level.) Complete instructions are furnished and work is reviewed to verify its accuracy, conformance with required procedures and instructions, and usefulness in facilitating the accountant's professional growth. Any technical problems not covered by instructions are brought to the attention of a superior. Carries out basic audit tests and procedures, such as: verifying reports against source accounts and records; reconciling bank and other accounts; and examining cash receipts and disbursements, payroll records, requisitions, receiving reports, and other accounting documents in detail to ascertain that transactions are properly supported and recorded. Prepares selected portions of audit working papers"

xTestAttorney <- "Performs consultation, advisory and/or trial work and carries out the legal processes necessary to effect the rights, privileges, and obligations of the organization. The work performed requires completion of law school with an L.L.B. degree or J.D. degree and admission to the bar. Responsibilities or functions include one or more of the following or comparable duties:
1. Preparing and reviewing various legal instruments and documents, such as contracts, leases, licenses, purchases, sales, real estate, etc. ;
2. Acting as agent of the organization in its transactions;
3. Examining material (e.g., advertisements, publications, etc.) for legal implications; advising officials of proposed legislation which might affect the organization;
4. Applying for patents, copyrights, or registration of the organization's products, processes, devices, and trademarks; advising whether to initiate or defend law suits;
5. Conducting pretrial preparations; defending the organization in lawsuits;"
```

6. Prosecuting criminal cases for a local or state government or defending the general public (for example, public defenders and attorneys rendering legal services to students); or

7. Advising officials on tax matters, government regulations, and/or legal rights.

Attorney jobs are matched at one of six levels according to two factors:

1. Difficulty level of legal work; and

2. Responsibility level of job.

Attorney jobs which meet the above definitions are to be classified and coded in accordance with a chart available upon request.

Legal questions are characterized by: facts that are well-established; clearly applicable legal precedents; and matters not of substantial importance to the organization. (Usually relatively limited sums of money, e.g., a few thousand dollars, are involved.)

a. legal investigation, negotiation, and research preparatory to defending the organization in potential or actual lawsuits involving alleged negligence where the facts can be firmly established and there are precedent cases directly applicable to the situation;

b. searching case reports, legal documents, periodicals, textbooks, and other legal references, and preparing draft opinions on employee compensation or benefit questions where there is a substantial amount of clearly applicable statutory, regulatory, and case material;

c. drawing up contracts and other legal documents in connection with real property transactions requiring the development of detailed information but not involving serious questions regarding titles to property or other major factual or legal issues.

d. preparing routine criminal cases for trial when the legal or factual issues are relatively straight forward and the impact of the case is limited; and

e. advising public defendants in regard to routine criminal charges or complaints and representing such defendants in court when legal alternatives and facts are relatively clear and the impact of the outcome is limited primarily to the defendant.

Legal work is regularly difficult by reason of one or more of the following: the absence of clear and directly applicable legal precedents; the different possible interpretations that can be placed on the facts, the laws, or the precedents involved; the substantial importance of the legal matters to the organization (e.g., sums as large as \$100,000 are generally directly or indirectly involved); or the matter is being strongly pressed or contested in formal proceedings or in negotiations by the individuals, corporations, or government agencies involved.

a. advising on the legal implications of advertising representations when the facts supporting the representations and the applicable precedent cases are subject to different interpretations;

b. reviewing and advising on the implications of new or revised laws affecting the organization;

c. presenting the organization's defense in court in a negligence lawsuit which is strongly pressed by counsel for an organized group;

d. providing legal counsel on tax questions complicated by the absence of precedent decisions that are directly applicable to the organization's situation;

e. preparing and prosecuting criminal cases when the facts of the cases are complex or difficult to determine or the outcome will have a significant impact within the jurisdiction; and

f. advising and representing public defendants in all phases of criminal proceedings when the facts of the case are complex or difficult to determine, complex or unsettled legal issues are involved, or the prosecutorial jurisdiction devotes substantial resources to obtaining a conviction."

xTestMachinist <- "Produces replacement parts and new parts in making repairs of metal parts of mechanical equipment. Work involves most of the following: interpreting written instructions and specifications; planning and laying out of work; using a variety of machinist's handtools and precision measuring instruments; setting up and operating standard machine tools; shaping of metal parts to close tolerances; making standard shop computations relating to dimensions of work, tooling, feeds, and speeds of machining; knowledge of the working properties of the common metals; selecting standard materials, parts, and equipment required for this work; and fitting and assembling parts into mechanical equipment. In general, the machinist's work normally requires a rounded training in machine-shop practice usually acquired through a formal apprenticeship or equivalent training and experience. Industrial machinery repairer. Repairs machinery or mechanical equipment. Work involves most of the following: examining machines and mechanical equipment to diagnose source of trouble; dismantling or partly dismantling machines and performing repairs that mainly involve the use of handtools in scraping and fitting parts; replacing broken or defective parts with items obtained from stock; ordering the production of a replacement part by a machine shop or sending the machine to a machine shop for major repairs; preparing written specifications for major repairs or for the production of parts ordered from machine shops; reassembling machines; and making all necessary adjustments for operation. In general, the work of a machinery maintenance mechanic requires rounded training and experience usually acquired through a formal apprenticeship or equivalent training and experience. Excluded from this classification are workers whose primary duties involve setting up or adjusting machines. Vehicle and mobile equipment mechanics and repairers. Repairs, rebuilds, or overhauls major assemblies of internal combustion automobiles, buses, trucks, or tract tractors. Work involves most of the following: Diagnosing the source of trouble and determining the extent of repairs required; replacing worn or broken parts such as piston rings, bearings, or other engine parts; grinding and adjusting valves; rebuilding carburetors; overhauling transmissions; and repairing fuel injection, lighting, and ignition systems. In general, the work of the motor vehicle mechanic requires rounded training and experience usually acquired through a formal apprenticeship or equivalent training and experience"

```

testJDS <- c(xTestAccountant, xTestAttorney, xTestMachinist)

# define the preprocessing (tolower case) function
# preproc_fun = tolower

# define the tokenization function
token_fun = text2vec::word_tokenizer

# loop to substitute "_" with blank space
for(j in seq(job[, 10])){
  job[j, 10] <- gsub("_", " ", job[j, 10])
}

# iterator for Job training and testing JDS
iter_Jobs = itoken(job[, 10],
  preprocessor = preproc_fun,
  tokenizer = token_fun,
  progressbar = TRUE)
iter_testJDS = itoken(testJDS,
  preprocessor = preproc_fun,
  tokenizer = token_fun,
  progressbar = TRUE)

```

```

jobs_Vocab= create_vocabulary(iter_Jobs, stopwords=tm::stopwords("english"),
ngram = c(1L, 2L))

jobsVectorizer = vocab_vectorizer(jobs_Vocab)

dtm_jobsTrain = create_dtm(iter_Jobs, jobsVectorizer)

dtm_testJDs = create_dtm(iter_testJDs, jobsVectorizer)

dim(dtm_jobsTrain); dim(dtm_testJDs)
## [1] 200 2675

## [1] 3 2675

set.seed(2)
fit1 <- cv.glmnet(x = as.matrix(dtm_jobsTrain), y = job[['highrank']],
family = 'binomial',
# lasso penalty
alpha = 1,
# interested in the area under ROC curve
type.measure = "auc",
# 10-fold cross-validation
nfolds = 10,
# high value is less accurate, but has faster training
thresh = 1e-3,
# again lower number of iterations for faster training
maxit = 1e3)
print(paste("max AUC =", round(max(fit1$cvm), 4)))
## [1] "max AUC = 0.7934"

```

Note that we somewhat improved the $AUC \sim 0.79$. Below, we will assess the JD predictive model using the three out of bag job descriptions (Fig. 20.7).

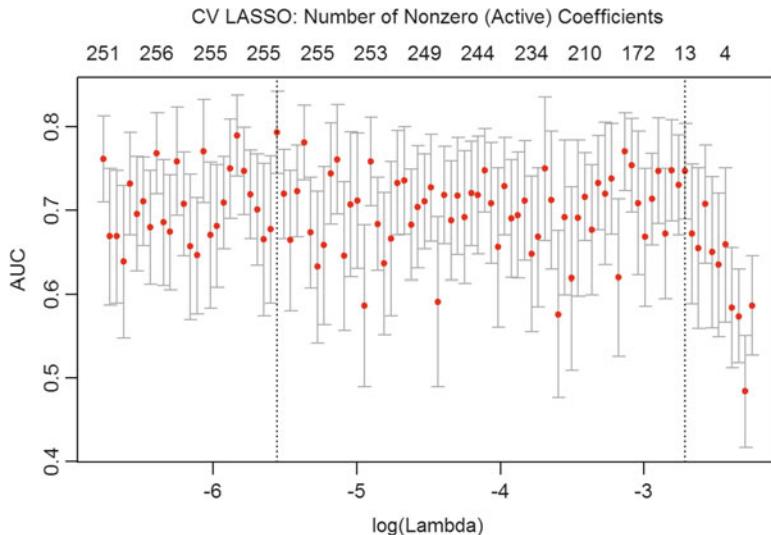


Fig. 20.7 AUC-based performance of the cross-validated LASSO-regularized model of job-ranking based on dtm_jobsTrain, see Figs. 20.5 and 20.6

```

plot(fit1)
# plot(fit1, xvar="lambda", label="TRUE")
mtext("CV LASSO: Number of Nonzero (Active) Coefficients", side=3, line=2.5)
predTestJDS <- predict(fit1, s = fit1$lambda.1se,
                       newx = dtm_testJDS, type="response"); predTestJDS

## 1
## 1 0.2153011
## 2 0.2200925
## 3 0.1257575

predTrainJDS <- predict(fit1, s = fit1$lambda.1se, newx = dtm_jobsTrain, type="response"); predTrainJDS

## 1
## 1 0.3050636
## 2 0.4118190
## 3 0.1288656
## 4 0.1493051
## 5 0.6432706
## 6 0.1257575
## 7 0.1257575
## 8 0.2561290
## 9 0.3866247
## 10 0.1262752
...
## 196 0.1257575
## 197 0.1257575
## 198 0.1257575
## 199 0.1257575
## 200 0.1257575

# Type can be: "link", "response", "coefficients", "class", "nonzero"

```

The output of the predictions shows that:

- On the *training data*, the predicted probabilities rapidly decrease with the indexing of the jobs, corresponding to the *overall job ranking* (highly ranked/desired jobs are listed on the top).
- On the three *testing job description data* (accountant, attorney, and machinist), there is a clear ranking difference between the machinist and the other two professions.

Also see the discussion in Chap. 18 about the different *types of predictions* that can be generated as outputs of `cv.glmnet` regularized forecasting methods.

20.4 Cosine Similarity

As we mentioned above, text data are often *transformed* in terms of Term Frequency-Inverse Document Frequency (TF-IDF), which offers a better input than the raw frequencies for many text-mining methods. An alternative transformation can be represented as a different distance measure such as the *cosine distance*, which is defined by:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2},$$

where θ represents the angle between the pair of vectors A and B in the Euclidean space spanned by the DTM matrix (Fig. 20.8).

```
cos_dist = function(mat){
  numer = tcrossprod(mat)
  denom1 = sqrt(apply(mat, 1, crossprod))
  denom2 = sqrt(apply(mat, 1, crossprod))
  1 - numer / outer(denom1, denom2)
}

dist_cos = cos_dist(as.matrix(dtm))

set.seed(2000)
fit_cos <- cv.glmnet(x = dist_cos, y = job[['highrank']],
  family = 'binomial',
  # lasso penalty
  alpha = 1,
```

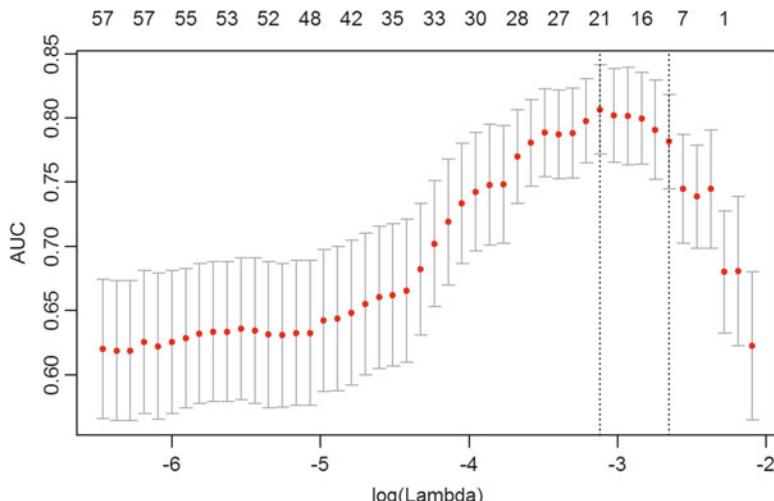


Fig. 20.8 AUC-based performance of the cross-validated LASSO-regularized model of job-ranking based on cosine-similarity distance (dist_cos), see Figs. 20.5, 20.6, and 20.7

```

# interested in the area under ROC curve
type.measure = "auc",
# 10-fold cross-validation
nfolds = 10,
# high value is less accurate, but has faster training
thresh = 1e-3,
# again lower number of iterations for faster training
maxit = 1e3)
plot(fit_cos)

print(paste("max AUC =", round(max(fit_cos$cvm), 4)))
## [1] "max AUC = 0.8065"

```

The AUC now is greater than 0.8, which is a pretty good result; even better than what we obtained from DTM or TF-IDF. This suggests that our machine “understanding” of the textual content, i.e., the natural language processing, leads to a more acceptable content classifier.

20.5 Sentiment Analysis

Let’s use the `text2vec:::movie_review` dataset, which consists of 5,000 movie reviews dichotomized as positive or negative. In the subsequent predictive analytics, this *sentiment* will represent our output feature:

$$Y = \text{Sentiment} = \begin{cases} 0, & \text{negative} \\ 1, & \text{positive} \end{cases}.$$

20.5.1 Data Preprocessing

The `data.table` package will also be used for some data manipulation. Let’s start with splitting the data into *training* and *testing* sets.

```

# install.packages("text2vec"); install.packages("data.table")
library(text2vec)
library(data.table)

# Load the movie reviews data
data("movie_review")

# coerce the movie reviews data to a data.table (DT) object
setDT(movie_review)

# create a key for the movie-reviews data table
setkey(movie_review, id)

```

```
# View the data
# View(movie_review)
head(movie_review); dim(movie_review); colnames(movie_review)

##      id sentiment
## 1: 10000_8      1
## 2: 10001_4      0
## 3: 10004_3      0
## 4: 10004_8      1
## 5: 10006_4      0
## 6: 10008_7      1
##
review
## 1: Homelessness (or Houselessness as George Carlin stated) has been an is
sue for years but never a plan to help those on the street that were once co
nsidered human who did everything from going to school ... Maybe they should
give it to the homeless instead of using it like Monopoly money.<br /><br />
Or maybe this film will inspire you to help others.
## 2:
This film lacked something I couldn't put my finger on at first: charisma on
the part of the leading actress. This inevitably translated to lack of chemi
stry when she shared the screen with her leading man. Even the romantic scen
es came across as being merely the actors at play ... I was disappointed in th
is movie. But, don't forget it was nominated for an Oscar, so judge for your
self.
## 3:
\"It appears that many critics find the idea of a Woody Allen drama unpalat
able.\\" And for good reason: they are unbearably wooden and pretentious imi
tations of Bergman. And let's ... \"ripping off\" Hitchcock in his suspense/
horror films? In Robin Wood's view, it's a strange form of cultural snobbery
. I would have to agree with that.
## 4:
This isn't the comedic Robin Williams, nor is it the quirky/insane Robin Wil
liams of recent thriller fame. This is a hybrid of the classic drama without
over-dramatization, mixed with Robin's new love of the thriller. But this is
n't a thriller, per se ... <br /><br />All in all, it's worth a watch, though
it's definitely not Friday/Saturday night fare.<br /><br />It rates a 7.7/10
from...<br /><br />the Fiend :.
## 5:
I don't know who to blame, the timid writers or the clueless director. It se
emed to be one of those movies where so much was paid to the stars (Angie, C
harlie, Denise, Rosanna and Jon) ... If they were only looking for laughs why
not cast Whoopi Goldberg and Judy Tenuta instead? This was so predictable I
was surprised to find that the director wasn't a five year old. What a waste
, not just for the viewers but for the actors as well.
## 6:
You know, Robin Williams, God bless him, is constantly shooting himself in t
he foot lately with all these dumb comedies he has done this decade (with pe
rhaps the exception of \"Death To Smoochy ... It's incredible that there is a
t least one woman in this world who is like this, and it's scary too.<br /><
br />This is a good, dark film that I highly recommend. Be prepared to be un
settled, though, because this movie leaves you with a strange feeling at the
end.

## [1] 5000      3
## [1] "id"      "sentiment" "review"
```

```
# Generate 80-20% training-testing split of the reviews
all_ids = movie_review$id
set.seed(1234)
train_ids = sample(all_ids, 5000*0.8)
test_ids = setdiff(all_ids, train_ids)
train = movie_review[train_ids, ]
test = movie_review[test_ids, ]
```

Next, we will vectorize the reviews by creating terms to *termID* mappings. Note that terms may include arbitrary *n-grams*, not just single words. The set of reviews will be represented as a sparse matrix, with rows and columns corresponding to reviews/reviewers and terms, respectively. This vectorization may be accomplished in several alternative ways, e.g., by using the corpus vocabulary, feature hashing, etc.

The vocabulary-based DTM, created by the `create_vocabulary()` function, relies on all unique terms from all reviews, where each term has a unique ID. In this example, we will create the review vocabulary using an *iterator* construct abstracting the input details and enabling *in memory* processing of the (training) data by chunks.

```
# define the text preprocessing
# either a simple (tolower case) function
preproc_fun = toLower

# or a more elaborate "cleaning" function
preproc_fun = function(x)                                     # text data
{ require("tm")
  x = gsub("<.*?>", " ", x)                                # regex removing HTML tags
  x = iconv(x, "latin1", "ASCII", sub="") # remove non-ASCII characters
  x = gsub("[^[:alnum:]]", " ", x)      # remove non-alpha-numeric values
  x = toLower(x)                                         # convert to lower case characters
  # x = removeNumbers(x)                                # removing numbers
  x = stripWhitespace(x)                                # removing white space
  x = gsub("^\\s+|\\s+$", "", x) # remove leading and trailing white space
  return(x)
}

# define the tokenization function
token_fun = word_tokenizer

# iterator for both training and testing sets
iter_train = itoken(train$review,
                    preprocessor = preproc_fun,
                    tokenizer = token_fun,
                    ids = train$id,
                    progressbar = TRUE)
```

```

iter_test = itoken(test$review,
  preprocessor = preproc_fun,
  tokenizer = token_fun,
  ids = test$id,
  progressbar = TRUE)
reviewVocab = create_vocabulary(iter_train)

# report the head and tail of the reviewVocab
reviewVocab

## Number of docs: 4000
## 0 stopwords: ...
## ngram_min = 1; ngram_max = 1
## Vocabulary:
##           terms terms_counts doc_counts
## 1:      lowlife           1           1
## 2:      sorin            1           1
## 3:      ewell            1           1
## 4: negligence          1           1
## 5: stribor            1           1
## ...
## 35661: Landscaping      1           1
## 35662: bikes            1           1
## 35663: primer           1           1
## 35664: Loosely          26          25
## 35665: cycling           1           1

```

Next, we can compute the *document term matrix* (DTM).

```

reviewVectorizer = vocab_vectorizer(reviewVocab)
t0 = Sys.time()
dtm_train = create_dtm(iter_train, reviewVectorizer)
dtm_test = create_dtm(iter_test, reviewVectorizer)

t1 = Sys.time()
print(difftime(t1, t0, units = 'sec'))

## Time difference of 3.844368 secs

# check the DTM dimensions
dim(dtm_train); dim(dtm_test)

## [1] 4000 35665
## [1] 1000 35665

# confirm that the training data review DTM dimensions are consistent
# with training review IDs, i.e., #rows = number of documents, and
# #columns = number of unique terms (n-grams), dim(dtm_train)[[2]]
identical(rownames(dtm_train), train$id)

## [1] TRUE

```

20.5.2 NLP/TM Analytics

We can now fit statistical models or derive machine learning model-free predictions. Let's start by using `glmnet()` to fit a *logit model* with LASSO (L_1) regularization and 10-fold cross-validation, see Chap. 18 (Fig. 20.9).

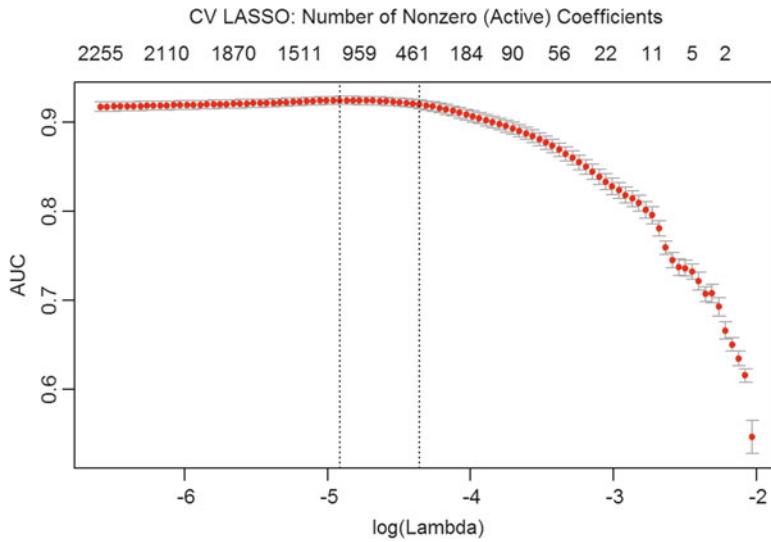


Fig. 20.9 AUC-based performance of the cross-validated LASSO-regularized model of movie sentiment analysis training data (dtm_train), see Fig. 20.8

```

library(glmnet)
nFolds = 10
t0 = Sys.time()
glmnet_classifier = cv.glmnet(x = dtm_train, y = train[['sentiment']],
  family = "binomial",
  # LASSO L1 penalty
  alpha = 1,
  # interested in the area under ROC curve or MSE
  type.measure = "auc",
  # n-fold internal (training data) stats cross-validation
  nfolds = nFolds,
  # threshold: high value is less accurate / faster training
  thresh = 1e-2,
  # again lower number of iterations for faster training
  maxit = 1e3
)

Lambda.best <- glmnet_classifier$Lambda.min
Lambda.best

## [1] 0.007344319
# report execution time
t1 = Sys.time()
print(difftime(t1, t0, units = 'sec'))

## Time difference of 5.923289 secs

# some prediction plots
plot(glmnet_classifier)
# plot(glmnet_classifier, xvar="lambda", label="TRUE")
mtext("CV LASSO: Number of Nonzero (Active) Coefficients", side=3, line=2.5)

```

Now let's look at external validation, i.e., testing the model on the independent 20% of the reviews we kept aside. The performance of the binary prediction (binary

sentiment analysis of these movie reviews) on the test data is roughly the same as we had from the internal statistical 10-fold cross-validation.

```

# report the mean internal cross-validated error
print(paste("max AUC =", round(max(glmnet_classifier$cvm), 4)))
## [1] "max AUC = 0.9246"

# report TESTING data prediction accuracy
xTest = dtm_test
yTest = test[['sentiment']]
predLASSO <- predict(glmnet_classifier,
                      s = glmnet_classifier$lambda.1se, newx = xTest)
testMSE_LASSO <- mean((predLASSO - yTest)^2); testMSE_LASSO
## [1] 2.869523

# Binarize the LASSO probability prediction
binPredLASSO <- ifelse(predLASSO<0.5, 0, 1)
table(binPredLASSO, yTest)

##          yTest
## binPredLASSO 0 1
##               0 455 152
##               1 40 353

# and testing data AUC
glmnet:::auc(yTest, predLASSO)
## [1] 0.9175598

# report the top 20 negative and positive predictive terms
summary(predLASSO)

##          1
##  Min.   :-12.80392
##  1st Qu.: -0.94586
##  Median :  0.14755
##  Mean   : -0.07101
##  3rd Qu.:  1.01894
##  Max.   :  6.60888

sort(predict.cv.glmnet(glmnet_classifier, s = Lambda.best, type = "coefficients"))[1:20]

## <sparse>[ <Logic> ] : .M.sub.i.Logical() maybe inefficient
## [1] -4.752272 -2.199304 -1.987171 -1.966585 -1.902009 -1.866655 -1.84496
## [6] -1.750693 -1.518148 -1.502966 -1.436081 -1.405963 -1.349566 -1.34285
## [15] -1.320218 -1.283414 -1.270231 -1.257663 -1.242869 -1.209449

rev(sort(predict.cv.glmnet(glmnet_classifier, s = Lambda.best, type = "coefficients")))[1:20]

## <sparse>[ <Logic> ] : .M.sub.i.Logical() maybe inefficient
## [1] 2.559487 2.416333 2.101371 1.913529 1.899846 1.684176 1.600367
## [8] 1.530320 1.519663 1.435103 1.430446 1.376056 1.343108 1.309902
## [15] 1.300156 1.287921 1.131859 1.078685 1.074059 1.015887

```

The (external) prediction performance, measured by AUC, on the testing data is about the same as the internal 10-fold stats cross-validation we reported above.

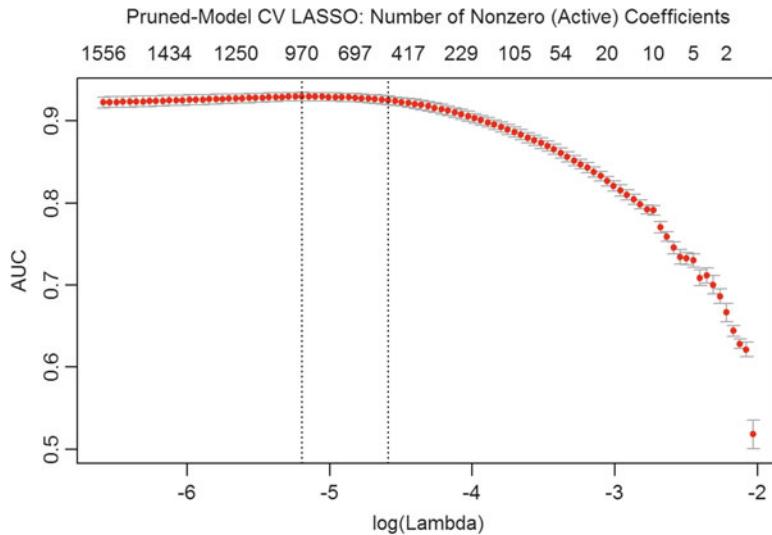


Fig. 20.10 AUC-based performance of the pruned cross-validated LASSO-regularized model of movie sentiment analysis (`glmnet_prunedClassifier`), see Fig. 20.9

20.5.3 Prediction Optimization

Earlier, we saw that we can also prune the vocabulary and perhaps improve prediction performance, e.g., by removing non-salient terms like stopwords and by using n -grams instead of single words (Fig. 20.10).

```
reviewVocab = create_vocabulary(iter_train,
  stopwords=tm::stopwords("english"), ngram = c(1L, 2L))

prunedReviewVocab = prune_vocabulary(reviewVocab,
  term_count_min = 10,
  doc_proportion_max = 0.5,
  doc_proportion_min = 0.001)
prunedVectorizer = vocab_vectorizer(prunedReviewVocab)

t0 = Sys.timecreate_dtm(iter_train, prunedVectorizer)
dtm_test = create_dtm(iter_test, prunedVectorizer)

t1 = Sys.time()
print(difftime(t1, t0, units = 'sec'))

## Time difference of 3.778152 secs
```

Next, let's refit the model and report its performance. Would there be an improvement in the prediction accuracy?

```
glmnet_prunedClassifier=cv.glmnet(x=dtm_train,
  y=train[['sentiment']],
  family = "binomial",
  # LASSO L1 penalty
  alpha = 1,
  # interested in the area under ROC curve or MSE
  type.measure = "auc",
  # n-fold internal (training data) stats cross-validation
  nfolds = nFolds,
  # threshold: high value is less accurate / faster training
  thresh = 1e-2,
  # again lower number of iterations for faster training
  maxit = 1e3
)

Lambda.best <- glmnet_prunedClassifier$Lambda.min
Lambda.best

## [1] 0.005555708
# report execution time
t1 = Sys.time()
print(difftime(t1, t0, units = 'sec'))

## Time difference of 6.978195 secs

# some prediction plots
plot(glmnet_prunedClassifier)
mtext("Pruned-Model CV LASSO: Number of Nonzero (Active) Coefficients",
  side=3, line=2.5)

# report the mean internal cross-validated error
print(paste("max AUC =", round(max(glmnet_prunedClassifier$cvm), 4)))

## [1] "max AUC = 0.9295"

# report TESTING data prediction accuracy
xTest = dtm_test
yTest = test[['sentiment']]
predLASSO = predict(glmnet_prunedClassifier,
  dtm_test, type = 'response')[,1]

testMSE_LASSO <- mean((predLASSO - yTest)^2); testMSE_LASSO

## [1] 0.1194583

# Binarize the LASSO probability prediction
binPredLASSO <- ifelse(predLASSO<0.5, 0, 1)
table(binPredLASSO, yTest)
##          yTest
## binPredLASSO 0 1
##                0 416 60
##                1  79 445

# and testing data AUC
glmnet:::auc(yTest, predLASSO)

## [1] 0.9252405

# report the top 20 negative and positive predictive terms
summary(predLASSO)
```

```

##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.0000014 0.2518000 0.5176000 0.5026000 0.7604000 0.9995000
sort(predict.cv.glmnet(glmnet_classifier, s = Lambda.best, type = "coefficients"))[1:20]
## <sparsed[ <logic> ] : .M.sub.i.Logical() maybe inefficient
## [1] -5.695082 -2.774694 -2.756099 -2.540456 -2.508213 -2.474586 -2.432767
## [8] -2.429874 -1.999731 -1.941299 -1.934803 -1.929788 -1.819220 -1.774936
##[15] -1.765978 -1.737596 -1.717957 -1.661592 -1.611752 -1.599558
rev(sort(predict.cv.glmnet(glmnet_classifier, s = Lambda.best, type = "coefficients")))[1:20]
## <sparsed[ <logic> ] : .M.sub.i.Logical() maybe inefficient
## [1] 3.276620 2.695083 2.575524 2.436630 2.366057 2.139067 2.087892
## [8] 2.027113 1.980694 1.894909 1.839621 1.777573 1.743082 1.599660
## [15] 1.579711 1.569817 1.533461 1.509555 1.453862 1.425065
# Binarize the LASSO probability prediction
# and construct an approximate confusion matrix
binPredLASSO <- ifelse(predLASSO<0.5, 0, 1)
table(binPredLASSO, yTest)

##          yTest
## binPredLASSO 0 1
##                0 416 60
##                1 79 445

```

Using n-grams improved a bit the sentiment prediction model.

Try these NLP techniques to other data like:

- MIMIC-III, a freely accessible critical care database. Johnson AEW, Pollard TJ, Shen L, Lehman L, Feng M, Ghassemi M, Moody B, Szolovits P, Celi LA, and Mark RG. Scientific Data (2016). DOI: 10.1038/sdata.2016.35. Available from: <http://www.nature.com/articles/sdata201635>
- Other data from the list of our Case-Studies.
- Your own free text.

20.6 Assignment: 20. Natural Language Processing/Text Mining

20.6.1 Mining Twitter Data

Use these R Data Mining Twitter data to apply NLP/TM methods and investigate the Twitter corpus.

- Construct a VCorpus object
- Clean the VCorpus object

- Build document term matrix (DTM)
- Compute the TF-IDF(term frequency - inverse document frequency)
- Use the DTM to construct a wordcloud.

20.6.2 *Mining Cancer Clinical Notes*

Use Head and Neck Cancer Medication Data to apply NLP/TM methods and investigate the corpus. You have already seen this data in Chap. 8; now we can go a step further.

- Use MEDICATION_SUMMARY to construct a VCorpus object.
- Clean the VCorpus object.
- Build the document term matrix (DTM).
- Add a column to indicate early and later stage according to seer_stage (refer to Chap. 8).
- Use the DTM to construct a word cloud for early stage, later stage and whole.
- Interpret according to the word cloud.
- Compute the TF-IDF (Term Frequency - Inverse Document Frequency).
- Apply LASSO on the unweighted and weighted DTM respectively and evaluate the results according to AUC.
- Try cosine similarity transformation, apply LASSO and compare the result.
- Use other measures such as “class” for `cv.glmnet()`.
- Does it appear that these classifiers understand well human language?

References

Kumar, E. (2011) *Natural Language Processing*, I. K. International Pvt Ltd, ISBN 9380578776, 9789380578774.

Kao, A, Poteet, SR (eds.) (2007) *Natural Language Processing and Text Mining*, Springer Science & Business Media, ISBN 1846287545, 9781846287541.

<https://github.com/kbenoit/spacyr>

<https://tartarus.org/martin/PorterStemmer/>

Chapter 21

Prediction and Internal Statistical Cross Validation



We should start by reviewing Chap. 14 (Model Performance Assessment). Cross-validation is a statistical approach for validating predictive methods, classification models, and clustering techniques. It assesses the reliability and stability of the results of the corresponding statistical analyses (e.g., predictions, classifications, forecasts) based on independent datasets. For prediction of trend, association, clustering, and classification, a model is usually trained on one dataset (*training data*) and subsequently tested on new data (*testing or validation data*). Statistical internal cross-validation uses iterative bootstrapping to define test datasets, evaluates the model predictive performance, and assesses its power to avoid overfitting. *Overfitting* is the process of computing a predictive or classification model that describes random error, i.e., fits to the noise components of the observations, instead of the actual underlying relationships and salient features in the data.

In this Chapter, we will use the Google Flu Trends, Autism, and Parkinson's disease case-studies to (1) illustrate exhaustive and non-exhaustive internal statistical cross-validation; (2) explore alternative forecasting types using linear and non-linear predictions; and (3) compare complementary predictor functions.

21.1 Forecasting Types and Assessment Approaches

In Chap. 7, we discussed the types of classification and prediction methods, including supervised and unsupervised learning. The former are direct and predictive, as there are known outcome variables that can be predicted, and the corresponding forecasts can be evaluated. The latter are indirect and descriptive, as there are no *a priori* labels or specific outcomes.

There are alternative metrics used for evaluation of model performance, see Chap. 14. For example, assessment of supervised prediction and classification methods depends on the type of the labeled outcome responses: categorical (binary or polytomous) vs. continuous.

- Confusion matrices reporting accuracy, FP, FN, PPV, NPV, LOR and other metrics may be used to assess predictions of dichotomous (binary) or polytomous outcomes.
- R^2 , correlations (between predicted and observed outcomes), and RMSE measures may be used to quantify the performance of various supervised forecasting methods on continuous features.

21.2 Overfitting

Before we go into the cross-validation of predictive analytics, we will present several examples of *overfitting* that illustrate why a certain amount of skepticism and mistrust may be appropriate when dealing with forecasting models that are based on large and complex data.

21.2.1 Example (US Presidential Elections)

By 2017, there were only **57 US presidential elections** and **45 presidents**. That is a small dataset, and learning from it may be challenging. For instance:

- If the predictor space expands to include things like *having false teeth*, it's pretty easy for the model to go from fitting the generalizable features of the data (the signal, e.g., presidential actions) to matching noise patterns (e.g., irrelevant characteristics like gender of the children of presidents, or types of dentures they may wear).
- When overfitting noise patterns takes place, the quality of the model fit assessed on the historical data may improve (e.g., better R^2 , more about the Coefficient of Determination is available [online](#)). At the same time, however, the model performance may be suboptimal when used to make inference about prospective data, e.g., future presidential elections.

Figure 21.1 shows a cartoon that includes some of the (unique) noisy presidential characteristics that are thought to be unimportant to electability, fitness for office, or expectations of presidential performance.

21.2.2 Example (Google Flu Trends)

A March 14, 2014 article in Science (DOI: <https://doi.org/10.1126/science.1248506>), identified problems in a Google Flu Trends (GFT) study, DOI <https://doi.org/10.1371/journal.pone.0023610>, which may be attributed in part to

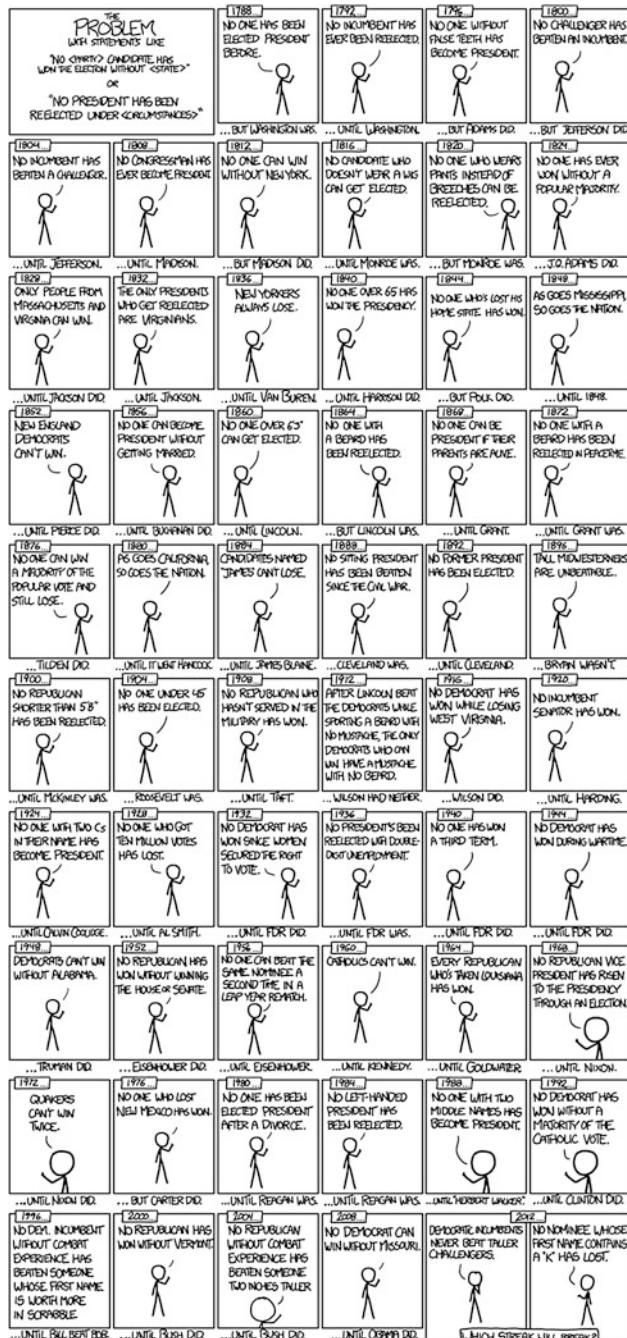


Fig. 21.1 Example of an overfitting based on extreme stratification of traits of presidential candidates

overfitting. The GFT model was built to predict the future Centers for Disease Control and Prevention (CDC) reports of doctor office visits for influenza-like illness (ILI). In February 2013, Nature reported that GFT was predicting more than double the proportion of doctor visits compared to the CDC forecast for the same period.

The GFT model found the best matches among 50 million web search terms to fit 1,152 data points. It predicted quite high odds of finding search terms that match the propensity of the flu, but which are structurally unrelated, and hence are not prospectively predictive. In fact, the GFT investigators reported weeding out seasonal search terms that were unrelated to the flu, which may have been strongly correlated to the CDC data, e.g., high school basketball season. The big GFT data may have overfitted the relatively small number of cases. This false-alarm result was also paired with a false-negative finding. The GFT model also missed the non-seasonal 2009 H1N1 influenza pandemic. This provides a cautionary tale about prediction, overfitting, and prospective validation.

21.2.3 *Example (Autism)*

Autistic brains constantly overfit visual and cognitive stimuli. To an autistic person, a general conversation of several adults may seem like a cacophony due to super-sensitive detail-oriented hearing and perception tuned to literally pick up all elements of the conversation and clues of the surrounding environment. At the same time, autistic brains may downplay body language, sarcasm, and non-literal cues. We can miss the forest for the trees when we start “overfitting”, i.e., when we over interpret the noise on top of the actual salient information. Ambient noise, trivial observations, and unrelated perceptions may obfuscate the true communication details.

Human conversations and communications involve exchanges of both critical information and random noise. Fitting a perfect model requires focus only on the “relevant” information. Overfitting occurs when attention is (excessively) consumed with peripheral noise, or worse, overwhelmed by inconsequential noise drowning the salient aspects of the communication exchange.

Any dataset is a mix of signal and noise. The main task of our brains is to sort these components and interpret the useful information while ignoring the noise. However, we should be cognizant that

“One person’s noise is another person’s treasure map!”

Our predictions are most accurate if we can model as much of the signal and as little of the noise as possible. Note that in these terms, R^2 is a poor metric to identify predictive power – it measures how much of the signal *and* the noise is explained by our model. In practice, it’s hard to always identify what’s signal and what’s noise. This is why practical applications tend to favor simpler models, since the more complicated a model is, the easier it is to overfit the noise component of the observed information.

21.3 Internal Statistical Cross-Validation is an Iterative Process

Internal statistical cross-validation assesses the expected performance of a prediction method in cases (e.g., subjects, units, regions, etc.) drawn from a similar population as the original training data sample. Internal validation is distinct from external validation, as the latter potentially allows for the existence of differences between the populations: training data, used to develop, or train, the technique, and testing data, used to independently quantify the performance of the technique.

Each step in the internal statistical cross-validation protocol involves:

- Randomly partitioning a sample of data into 2 complementary subsets (training + testing),
- Performing the analysis, fitting or estimating the model using the training set,
- Validating the analysis or evaluating the performance of the model using the separate testing set,
- Increasing the iteration index and repeating the process. Various termination criterial can be chosen like a fixed number of iterations, a desired mean variability, or an upper bound on the error-rate.

One example of internal statistical cross-validation used for predictive diagnostic modeling in Parkinson's disease is available [online](#).

To reduce the noise and variability at each iteration, the final validation results may include the averaged performance results across iterations.

In cases when new observations are hard to obtain (due to costs, reliability, time, or other constraints), cross-validation guards against testing hypotheses suggested by the data themselves (also known as **Type III error** or False-Suggestion).

Cross-validation is different from *conventional-validation* (e.g. 80–20% partitioning the data set into training and testing subsets) where the prediction error (e.g., Root Mean Square Error, RMSE) evaluated on the training data is not a useful estimator of model performance, as it does not generalize across multiple samples.

In general, the errors of the conventional-validation are based on the results of a specific test dataset and may not accurately represent the model performance. A more appropriate strategy to properly estimate model prediction performance is to use cross-validation (CV), which combines (e.g., averages) multiple prediction errors to measure the expected model performance. CV corrects for the expected stochastic nature of partitioning the training and testing sets and generates a more accurate and robust estimate of the expected model performance.

Relative to a simpler model, a more complex model may *overfit-the-data* if it has a short foresight, i.e., it may generate accurate fitting results for known data but less accurate results when predicting based on new data. Knowledge from past experiences may include either *relevant* or *irrelevant* (noise) information. In challenging data-driven prediction models where the uncertainty (entropy) is high, more noise is present in past information that needs to be accounted for in prospective

forecasting. However, it is generally hard to discriminate patterns from noise in complex systems, which makes it difficult to decide what part to model and what to ignore. Models that reduce the chance of fitting noise are called **robust**.

21.4 Example (Linear Regression)

Let's demonstrate a simple model assessment using linear regression. Suppose we observe the response values $\{y_1, \dots, y_n\}$, and the corresponding k predictors represented as a kD vector of covariates $\{x_1, \dots, x_n\}$, where subjects/cases are indexed by $1 \leq i \leq n$, and the data-elements (variables) are indexed by $1 \leq j \leq k$.

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{pmatrix}.$$

Using least squares to estimate the linear function parameters (effect-sizes), β_1, \dots, β_k , allows us to compute a hyperplane $y = a + x\beta$ that best fits the observed data $(x_i, y_i)_{1 \leq i \leq n}$. This is expressed as a matrix by:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} x_{1,1} & \cdots & x_{1,k} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,k} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_k \end{pmatrix}.$$

Corresponding to the system of linear hyperplanes:

$$\begin{cases} y_1 = a_1 + x_{1,1}\beta_1 + x_{1,2}\beta_2 + \cdots + x_{1,k}\beta_k \\ y_2 = a_2 + x_{2,1}\beta_1 + x_{2,2}\beta_2 + \cdots + x_{2,k}\beta_k \\ \vdots \\ y_n = a_1 + x_{n,1}\beta_1 + x_{n,2}\beta_2 + \cdots + x_{n,k}\beta_k \end{cases}.$$

One measure to evaluate the model fit may be the mean squared error (MSE). The MSE for a given value of the parameters α and β on the observed training data $(x_i, y_i)_{1 \leq i \leq n}$ is expressed as:

$$MSE = \frac{1}{n} \sum_{i=1}^n \left(y_i - \underbrace{\left(a_1 + x_{i,1}\beta_1 + x_{i,2}\beta_2 + \cdots + x_{i,k}\beta_k \right)}_{\text{predicted value } \hat{y}_i \text{ at } x_{i,1}, \dots, x_{i,k}} \right)^2.$$

And the corresponding root mean square error (RMSE) is:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(y_i - \underbrace{(a_1 + x_{i,1}\beta_1 + x_{i,2}\beta_2 + \cdots + x_{i,k}\beta_k)}_{\text{predicted value } \hat{y}_i \text{ at } x_{i,1}, \dots, x_{i,k}} \right)^2}.$$

In the linear model case, the expected value of the MSE (over the distribution of training sets) for the **training set** is $\frac{n-k-1}{n+k+1}E$, where E is the expected value of the MSE for the **testing/validation data**. Therefore, fitting a model and computing the MSE on the training set, we may produce an over optimistic evaluation assessment (smaller RMSE) of how well the model may fit another dataset. This bias represents *in-sample* estimate of the fit, whereas we are interested in the cross-validation estimate as an *out-of-sample* estimate.

In the linear regression model, cross validation may not be as useful, since we can compute the **exact** correction factor $\frac{n-k-1}{n+k+1}$ to obtain an estimate of the (unknown) exact expected *out-of-sample* fit using the (known) *in-sample* MSE (under)estimate. However, even in this situation, cross-validation remains useful as it can be used to select an optimal regularized cost function.

In most other modeling procedures (e.g. logistic regression), there are no simple general closed-form expressions (formulas) to adjust the cross-validation error estimate of the known in-sample fit to estimate the unknown out-of-sample error rate. Cross-validation is general strategy to predict the performance of a model on a validation set using stochastic computation instead of obtaining experimental, theoretical, mathematical, or closed-form analytic error estimates.

21.4.1 Cross-Validation Methods

There are two classes of cross-validation approaches, *exhaustive* and *non-exhaustive*.

21.4.2 Exhaustive Cross-Validation

Exhaustive cross-validation methods are based on determining all possible ways to divide the original sample into training and testing data. For instance, the *Leave- m -out cross-validation* involves using m observations for testing and the remaining $(n - m)$ observations as training. The case when $m = 1$, i.e., leave-one-out method, is only applicable when n is small, due to its huge computational cost. This process is repeated on all partitions of the original sample. This method requires model fitting and validating C_m^n times (n is the total number of observations in the original sample and m is the number of observations left out for validation). This requires a very large number of iterations.

21.4.3 Non-Exhaustive Cross-Validation

Non-exhaustive cross validation methods use bootstrap approximation to avoid computing estimates/errors using all possible partitionings of the original sample. For example, in the **k-fold cross-validation**, the original sample is randomly partitioned into k equal sized subsamples, or *folds*. Of all k subsamples, a single subsample is kept as final testing data for validation of the model. The other $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times, corresponding to the k folds. Each of the k subsamples is used once as the validation data. In the end, the corresponding k results are averaged (or otherwise aggregated) to generate a final pooled model-quality estimation. In k -fold validation, all observations are used for both training and validation, and each observation is used for validation exactly once. In general, k is a parameter that needs to be selected by the investigator (common values may be 5 or 10).

A general case of the k -fold validation is $k = n$ (the total number of observations), when it coincides with the **leave-one-out cross-validation**.

A variation of the k -fold validation is **stratified k-fold cross-validation**, where each fold has (approximately) the same mean response value. For instance, if the model represents a binary classification of cases (e.g., controls vs. patients), this implies that each fold contains roughly the same proportion of the two class labels.

Repeated random sub-sampling validation splits randomly the entire dataset into a training set, where the model is fit, and a testing set, where the predictive accuracy is assessed. Again, the results are averaged over all iterative splits. This method has an advantage over k -fold cross validation, as the proportion of the training/testing split is not dependent on the number of iterations (folds). However, its drawback is that some observations may never be selected in the testing/validation subsample, whereas others may be selected multiple times. As validation subsets may overlap, the results may vary each time we repeat the validation protocol, unless we set a seed point in the algorithm.

Asymptotically, as the number of random splits increases, the *repeated random sub-sampling* validation approaches the *leave- k -out cross-validation*.

21.5 Case-Studies

In the examples below, we have intentionally suppressed some of the R output to save space. This is accomplished using this Rmarkdown command, `{r eval=TRUE, results='hide'}`, however, the reader is encouraged to try hands-on all the protocols, to make modifications, to inspect, and finally to interpret the outputs.

21.5.1 Example 1: Prediction of Parkinson's Disease Using Adaptive Boosting (AdaBoost)

This Parkinson's Diseases study, which involves heterogeneous neuroimaging, genetics, clinical, and phenotypic data for over 600 volunteers including multivariate data for three cohorts (HC=Healthy Controls, PD=Parkinson's, SWEDD = subjects without evidence for dopaminergic deficit).

```
# update packages
# update.packages()
```

Load the data: 06_PPMI_ClassificationValidationData_Short.csv.

```
ppmi_data <-
read.csv("https://umich.instructure.com/files/330400/downLoad?download_frd=1",
header=TRUE)
```

Binarize the Dx (clinical diagnosis) classes.

```
# binarize the Dx classes
ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control",
"Control", "Patient")
attach(ppmi_data)

head(ppmi_data)
# View (ppmi_data)
```

Obtain a model-free predictive analytics, e.g., AdaBoost classification, and report the results.

```
# Model-free analysis, classification
# install.packages("crossval")
# install.packages("ada")
# library("crossval")
require(crossval)
require(ada)
#set up adaboosting prediction function

# Define a new AdaBoost classification result-reporting function
my.ada <- function (train.x, train.y, test.x, test.y, negative, formula){
  ada.fit <- ada(train.x, train.y)
  predict.y <- predict(ada.fit, test.x)
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
  # negative is the label of a negative "null" sample (default: "control").
  return (out)
}
```

When group sizes are imbalanced, we may need to rebalance them to avoid potential biases of the dominant cohorts. In this case, we will re-balance the groups using the package SMOTE Synthetic Minority Oversampling Technique. SMOTE may be used to handle class imbalance in binary classification, see Chap. 3.

```

# balance cases
# SMOTE: Synthetic Minority Oversampling Technique to handle class misbalance
# in binary classification.
set.seed(1000)
# install.packages("unbalanced") to deal with unbalanced group data
require(unbalanced)
ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
uniqueID <- unique(ppmi_data$FID_IID)
ppmi_data <- ppmi_data[ppmi_data$VisitID==1, ]
ppmi_data$PD <- factor(ppmi_data$PD)

colnames(ppmi_data)
# ppmi_data.1<-ppmi_data[, c(3:281, 284, 287, 336:340, 341)]
n <- ncol(ppmi_data)
output.1 <- ppmi_data$PD

# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
# remove Default Real Clinical subject classifications!
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD",
"X", "FID_IID"))]
# output <- as.matrix(ppmi_data[, which(names(ppmi_data) %in% {"PD"})])
output <- as.factor(ppmi_data$PD)
c(dim(input), dim(output))

#balance the dataset
set.seed(123)
data.1<-ubBalance(X= input, Y=output, type="ubSMOTE", percOver=300, percUnder=150,
verbose=TRUE)
balancedData<-cbind(data.1$X, data.1$Y)
table(data.1$Y)

nrow(data.1$X); ncol(data.1$X)
nrow(balancedData); ncol(balancedData)
nrow(input); ncol(input)

colnames(balancedData) <- c(colnames(input), "PD")

```

Next, we'll check the re-balanced cohort sizes (Fig. 21.2).

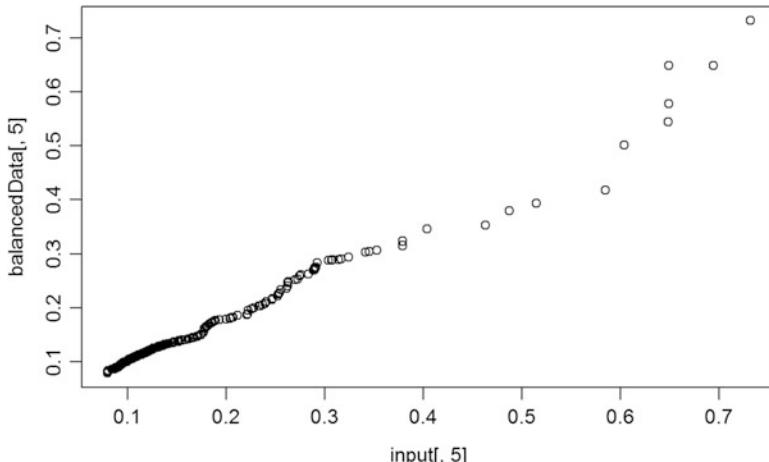


Fig. 21.2 Quantile-quantile plot of the original and rebalanced data distributions for one feature

```

###Check balance
## T test
alpha.0.05 <- 0.05
test.results.bin <- NULL          # binarized/dichotomized p-values
test.results.raw <- NULL          # raw p-values

# get a better error-handling t.test function that gracefully handles NA's and trivial variances
my.t.test.p.value <- function(input1, input2) {
  obj <- try(t.test(input1, input2), silent=TRUE)
  if (is(obj, "try-error"))
    return(NA)
  else
    return(obj$p.value)
}

for (i in 1:ncol(balancedData))
{
  test.results.raw[i] <- my.t.test.p.value(input[, i], balancedData [, i])
  test.results.bin[i] <- ifelse(test.results.raw[i] > alpha.0.05, 1, 0)
# binarize the p-value (0=significant, 1=otherwise)
  print(c("i=", i, "var=", colnames(balancedData[i]), "t-test_raw_p_value=",
  test.results.raw[i]))
}

# we can also employ (e.g., FDR, Bonferronni) correction for multiple
# testing!
# test.results.corr <- stats::p.adjust(test.results.raw, method = "fdr",
n = length(test.results.raw))
# where methods are "holm", "hochberg", "hommel", "bonferroni", "BH",
# "BY", "fdr", "none")
# plot(test.results.raw, test.results.corr)
# sum(test.results.raw < alpha.0.05, na.rm=T)/length(test.results.raw)
#check proportion of inconsistencies
# sum(test.results.corr < alpha.0.05, na.rm =T)/length(test.results.corr)

qqplot(input[, 5], balancedData [, 5]) # check visually for differences
# between the distributions of the raw (input) and rebalanced data (for only
# one variable, in this case)

```

```

# Now, check visually for differences between the distributions of the raw
# (input) and rebalanced data.
# par(mar=c(1,1,1,1))
# par(mfrow=c(10,10))
# for(i in c(1:62,64:101)){ qqplot(balancedData [, i],input[, i]) } #except
VisitID
# as the sample-size is changed:
Length(input[, 5]); Length(balancedData [, 5])
# to plot raw vs. rebalanced pairs (e.g., var="L_insular_cortex_Volume"), we
need to equalize the lengths
#plot (input[, 5] +0*balancedData [, 5], balancedData [, 5]) # [, 5] ==
# "L_insular_cortex_Volume"

# print(c("T-test results: ", test.results))
# zeros (0) are significant independent between-group T-test differences,
# ones (1) are insignificant

for (i in 1:(ncol(balancedData)-1))
{

```

```

  test.results.raw [i] <- wilcox.test(input[, i], balancedData [, i])$p.value
  test.results.bin [i] <- ifelse(test.results.raw [i] > alpha.0.05, 1, 0)
  print(c("i=", i, "Wilcoxon-test=", test.results.raw [i]))
}
print(c("Wilcoxon test results: ", test.results.bin))
# test.results.corr <- stats::p.adjust(test.results.raw, method = "fdr", n = length(test.results.raw))
# where methods are "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none")
# plot(test.results.raw, test.results.corr)

```

The next step will be the actual cross-validation.

```

# using raw data:
X <- as.data.frame(input); Y <- output
neg <- "1" # "Control" == "1"

# using Rebalanced data:
X <- as.data.frame(data.1$X); Y <- data.1$Y
# balancedData<-cbind(data.1$X, data.1$Y); dim(balancedData)

# Side note: There is a function name collision for "crossval", the same method is present in the "mlr" (machine Learning in R) package and in the "crosval" package.
# To specify a function call from a specific package do: packagename::functionname()

set.seed(115)
cv.out <- crossval::crossval(my.ada, X, Y, K = 5, B = 1, negative = neg)
  # the label of a negative "null" sample (default: "control")
out <- diagnosticErrors(cv.out$stat)

print(cv.out$stat)
##      FP      TP      TN      FN
## 0.6 109.6  97.0   0.2

print(out)
##          acc      sens      spec      ppv      npv      lor
## 0.9961427 0.9981785 0.9938525 0.9945554 0.9979424 11.3918119

```

As we can see from the reported metrics, the overall averaged AdaBoost-based diagnostic predictions are quite good.

21.5.2 Example 2: Sleep Dataset

These data contain the effect of two soporific drugs to increase hours of sleep (treatment-compared design) on 10 patients. The data are available in R by default (`sleep {datasets}`).

First, load the data and report some graphs and summaries (Fig. 21.3).

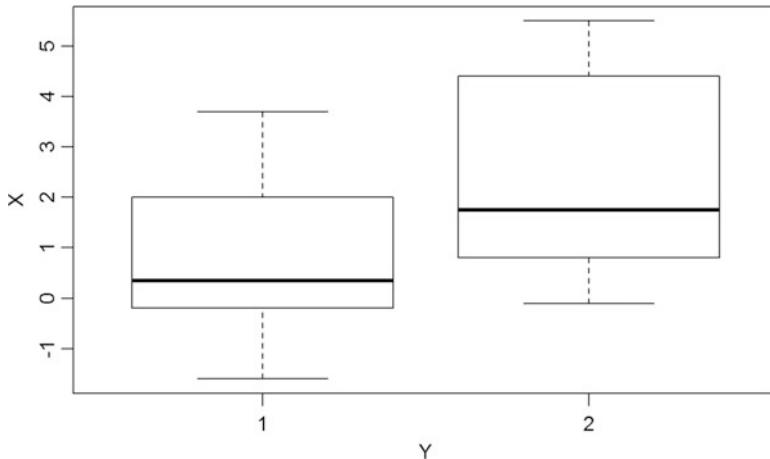


Fig. 21.3 Box-and whisker plots of the hours of sleep for the two cohorts in the sleep dataset

```

data(sleep); str(sleep)
X = as.matrix(sleep[, 1, drop=FALSE]) # increase in hours of sleep,
  # drop is logical, if TRUE the result is coerced to the lowest possible
  dimension.
# The default is to drop if only one column is left, but not to drop if only
one row is left.
Y = sleep[, 2] # drug given
plot(X ~ Y)

levels(Y) # "1" "2"
dim(X) # 20  1

```

Next, we will define a new LDA (linear discriminant analysis) predicting function and perform the cross-validation (CV) on the resulting predictor.

```

require("MASS") # for lda function

predfun.Lda = function(train.x, train.y, test.x, test.y, negative)
{   lda.fit = lda(train.x, grouping=train.y)
  ynew = predict(lda.fit, test.x)$class
  # count TP, FP etc.
  out = confusionMatrix(test.y, ynew, negative=negative)
  return( out )
}

# install.packages("crossval")
library("crossval")

set.seed(123456)
cv.out <- crossval::crossval(predfun.Lda, X, Y, K=5, B=20, negative="1",
  verbose=FALSE)
cv.out$stat
diagnosticErrors(cv.out$stat)

```

Execute the above code and interpret the diagnostic results measuring the performance of the LDA prediction.

21.5.3 Example 3: Model-Based (Linear Regression) Prediction Using the Attitude Dataset

These data represent a survey of clerical employees of an organization with 35 employees in 30 (randomly selected) departments. The data include the proportion of favorable responses to 7 questions in each department.

Let's load and summarize the data, which is available in the R `{datasets}` as `attitude`.

```
# ?attitude, colnames(attitude)
# Note: when using a data frame, a time-saver is to use "." to indicate "include all covariates" in the DF.
# E.g., fit <- lm(Y ~ ., data = D)

data("attitude")
y = attitude[, 1]           # rating variable
x = attitude[, -1]          # data frame with the remaining variables
is.factor(y)
summary( lm(y ~ ., data=x) ) # R-squared: 0.7326
# set up lm prediction function
```

We will demonstrate model-based analytics using `lm` and `lda`, and then will validate the forecasting using CV.

```
predfun.lm = function(train.x, train.y, test.x, test.y)
{
  lm.fit = lm(train.y ~ ., data=train.x)
  ynew = predict(lm.fit, test.x)
  # compute squared error risk (MSE)
  out = mean( (ynew - test.y)^2)
  # note that, in general, when fitting linear model to continuous
  # outcome variable (Y),
  # we can't use the out<-confusionMatrix(test.y, ynew, negative=n
  # egative), as it requires a binary outcome
  # this is why we use the MSE as an estimate of the discrepancy b
  # etween observed & predicted values
  return(out)
}
# require("MASS")
#predfun.lda = function(train.x, train.y, test.x, test.y, negative)
#{  lda.fit = lda(train.x, grouping=train.y)
#  ynew = predict(lda.fit, test.x)$class
#  # count TP, FP etc.
#  #  out = confusionMatrix(test.y, ynew, negative=negative)
#  #return( out )
#}
```

```

# prediction MSE using all variables
set.seed(123456)
cv.out.Lm = crossval::crossval(predfun.Lm, x, y, K=5, B=20, verbose=FALSE)
c(cv.out.Lm$stat, cv.out.Lm$stat.se) # 72.581198 3.736784
# reducing to using only two variables
cv.out.Lm = crossval::crossval(predfun.Lm, x[, c(1, 3)], y, K=5, B=20, verbose=FALSE)
c(cv.out.Lm$stat, cv.out.Lm$stat.se) # 52.563957 2.015109

```

21.5.4 Example 4: Parkinson's Data (ppmi_data)

Let's go back to the more elaborate PD data and start by loading and preprocessing the derived-PPMI data.

```

# ppmi_data <- read.csv("https://umich.instructure.com/files/330400/download?download_frd=1", header=TRUE)
# ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control", "Control", "Patient")
# attach(ppmi_data); head(ppmi_data)
# install.packages("crossval")
# library("crossval")
# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)
# input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD", "X", "FID_IID"))]
# output <- as.factor(ppmi_data$PD)

# remove the irrelevant variables (e.g., visit ID)
output <- as.factor(ppmi_data$PD)
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD", "X", "FID_IID", "VisitID"))]

X = as.matrix(input)      # Predictor variables
Y = as.matrix(output)    # Actual PD clinical assessment
dim(X);
dim(Y)

layout(matrix(c(1, 2, 3, 4), 2, 2))           # optional 4 graphs/page
fit <- lm(Y~X);
plot(fit)          # plot the fit

Levels(as.factor(Y))           # "0" "1"
## [1] "0" "1"
c(dim(X), dim(Y))           # 1043 103
## [1] 422 100 422   1

```

Apply cross-validation to assess the performance of the linear model (Fig. 21.4).

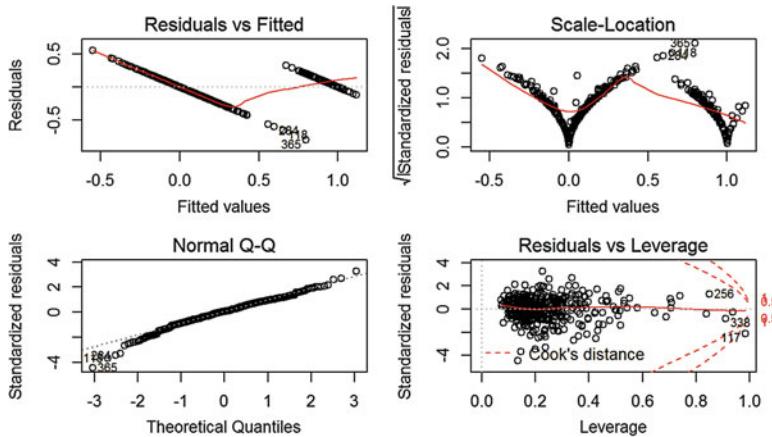


Fig. 21.4 Residual plots provide exploratory analytics of the model quality

```

set.seed(12345)
# cv.out.lm = crossval::crossval(predfun.lm, as.data.frame(X), as.numeric(Y),
# , K=5, B=20)

cv.out.Lda = crossval::crossval(predfun.Lda, X, Y, K=5, B=20, negative="1",
verbose=FALSE)
# K=Number of folds; B=Number of repetitions.

# Results
#cv.out.lda$stat;
#cv.out.lda;
diagnosticErrors(cv.out.Lda$stat)

##      acc      sens      spec      ppv      npv      lor
## 0.9617299 0.9513333 0.9872951 0.9945984 0.8918919 7.3258500

#cv.out.lm$stat;
#cv.out.lm;
#diagnosticErrors(cv.out.lm$stat)

```

21.6 Summary of CV output

The cross-validation (CV) output object includes the following three components:

- `stat.cv`: Vector of statistics returned by `predfun` for each cross validation run.
- `stat`: Mean statistic returned by `predfun` averaged over all cross validation runs.
- `stat.se`: Variability measuring the corresponding standard error.

21.7 Alternative Predictor Functions

We have already seen a number of `predict()` functions, e.g., Chap. 18. Below, we will add to the collection of predictive analytics and forecasting functions.

21.7.1 Logistic Regression

We already saw the *logit* model in Chap. 18. Now, we will demonstrate a logit-predictor function by applying it to the PD dataset.

```
# ppmi_data <- read.csv("https://umich.instructure.com/files/330400/download?download_frd=1", header=TRUE)
# ppmi_data$ResearchGroup <- ifelse(ppmi_data$ResearchGroup == "Control",
# "Control", "Patient")
# install.packages("crossval"); library("crossval")
# ppmi_data$PD <- ifelse(ppmi_data$ResearchGroup=="Control", 1, 0)

# remove the irrelevant variables (e.g., visit ID)
output <- as.factor(ppmi_data$PD)
input <- ppmi_data[, -which(names(ppmi_data) %in% c("ResearchGroup", "PD",
"X", "FID_IID", "VisitID"))]
X = as.matrix(input)      # Predictor variables
Y = as.matrix(output)
```

Note that the predicted values are in *log* terms, so they need to be *exponentiated* to be correctly interpreted.

```
Lm.Logit <- glm(as.numeric(Y) ~ ., data = as.data.frame(X), family =
"binomial")
ynew <- predict(Lm.Logit, as.data.frame(X)); #plot(ynew)
ynew2 <- ifelse(exp(ynew)<0.5, 0, 1); # plot(ynew2)

predfun.Logit = function(train.x, train.y, test.x, test.y, neg)
{ Lm.Logit <- glm(train.y ~ ., data = train.x, family = "binomial")
  ynew = predict(Lm.Logit, test.x)
  # compute TP, FP, TN, FN
  ynew2 <- ifelse(exp(ynew)<0.5, 0, 1)
  out = confusionMatrix(test.y, ynew2, negative=neg) # Binary outcome,
we can use confusionMatrix
  return( out )
}

# Reduce the bag of explanatory variables, purely to simplify the
interpretation of the analytics in this example!
input.short <- input[, which(names(input) %in% c("R_fusiform_gyrus_Volume",
"R_fusiform_gyrus_ShapeIndex", "R_fusiform_gyrus_Curvedness",
"Sex", "Weight", "Age", "chr12_rs34637584_GT", "chr17_rs11868035_GT",
"UPDRS_Part_I_Summary_Score_Baseline",
"UPDRS_Part_I_Summary_Score_Month_03",
"UPDRS_Part_II_Patient_Questionnaire_Summary_Score_Baseline",
"UPDRS_Part_III_Summary_Score_Baseline",
"X_Assessment_Non.Motor_Epworth_Sleepiness_Scale_Summary_Score_Baseline"
))]
X = as.matrix(input.short)

cv.out.Logit = crossval::crossval(predfun.Logit, as.data.frame(X),
as.numeric(Y), K=5, B=2, neg="1", verbose=FALSE)
cv.out.Logit$stat.cv
```

```

##      FP  TP  TN  FN
## B1.F1  1 50 31  2
## B1.F2  0 60 19  6
## B1.F3  2 55 19  8
## B1.F4  3 58 23  0
## B1.F5  3 60 21  1
## B2.F1  2 56 22  4
## B2.F2  0 57 23  5
## B2.F3  3 60 20  1
## B2.F4  1 58 23  2
## B2.F5  1 54 27  3

diagnosticErrors(cv.out.Logit$stat)

##      acc      sens      spec      ppv      npv      Lor
## 0.9431280 0.9466667 0.9344262 0.9726027 0.8769231 5.5331424

```

Caution: Note that if you forget to exponentiate the values of the predicted logistic model (see `ynew2` in `predict.logit`), you will get nonsense results, e.g., all cases may be predicted to be in one class, trivial sensitivity, or incorrect NPP.

21.7.2 Quadratic Discriminant Analysis (QDA)

In Chaps. 8 and 21, we discussed the *linear* and *quadratic* discriminant analysis models. Let's now introduce a `predfun.qda()` function.

```

predfun.qda = function(train.x, train.y, test.x, test.y, negative)
{
  require("MASS") # for lda function
  qda.fit = qda(train.x, grouping=train.y)
  ynew = predict(qda.fit, test.x)$class
  out.qda = confusionMatrix(test.y, ynew, negative=negative)
  return( out.qda )
}

cv.out.qda = crossval::crossval(predfun.qda, as.data.frame(input.short),
  as.factor(Y), K=5, B=20, neg="1")

## Error in qda.default(x, grouping, ...): rank deficiency in group 1
diagnosticErrors(cv.out.Lda$stat); diagnosticErrors(cv.out.qda$stat);

## Error in diagnosticErrors(cv.out.qda$stat): object 'cv.out.qda' not found

```

This error message: "Error in `qda.default(x, grouping, ...)`: rank deficiency in group 1" indicates that there is a rank deficiency, i.e. some variables are collinear and one or more covariance matrices cannot be inverted to obtain the estimates in group 1 (Controls).

If you remove the strongly correlated data elements ("R_fusiform_gyrus_Volume", "R_fusiform_gyrus_ShapeIndex", and "R_fusiform_gyrus_Curvedness"), the rank-deficiency problem goes away.

```



```

It makes sense to contrast the QDA and GLM/Logit predictions.

```

diagnosticErrors(cv.out.qda$stat); diagnosticErrors(cv.out.Logit$stat)
##      acc      sens      spec      ppv      npv      lor
## 0.9407583 0.9533333 0.9098361 0.9629630 0.8880000 5.3285694
##      acc      sens      spec      ppv      npv      lor
## 0.9431280 0.9466667 0.9344262 0.9726027 0.8769231 5.5331424

```

Clearly, both the QDA and Logit model predictions are quite similar and reliable.

21.7.3 *Foundation of LDA and QDA for Prediction, Dimensionality Reduction, and Forecasting*

Previously, in Chap. 8 we saw some examples of LDA/QDA methods. Now, we'll provide more details. Both LDA (Linear Discriminant Analysis) and QDA (Quadratic Discriminant Analysis) use probabilistic models of the class conditional distribution of the data $P(X | Y = k)$ for each class k . Their predictions are obtained by using the Bayesian theorem (http://wiki.socr.umich.edu/index.php/SMHS_BayesianInference#Bayesian_Rule):

$$P(Y = k | X) = \frac{P(X | Y = k)P(Y = k)}{P(X)} = \frac{P(X | Y = k)P(Y = k)}{\sum_{l=0}^{\infty} P(X | Y = l)P(Y = l)}$$

Thus, we select the class k , which **maximizes** this conditional probability (maximum likelihood estimation). In linear and quadratic discriminant analysis, $P(X | Y)$ is modelled as a multivariate Gaussian distribution with density:

$$P(X | Y = k) = \frac{1}{(2\pi)^n |\Sigma_k|^{\frac{1}{2}}} \times e^{\left(-\frac{1}{2}(X - \mu_k)^T \Sigma_k^{-1} (X - \mu_k)\right)}.$$

This model can be used to classify data by using the training data to **estimate**:

- (1) The class prior probabilities $P(Y = k)$ by counting the proportion of observed instances of class k ,

- (2) The class means μ_k by computing the empirical sample class means, and
- (3) The covariance matrices by computing either the empirical sample class covariance matrices, or by using a regularized estimator, e.g., LASSO).

In the **linear case** (LDA), the Gaussians for each class are assumed to share the same covariance matrix: $\Sigma_k = \Sigma$ for each class k . This leads to linear decision surfaces separating different classes. This is clear from comparing the log-probability ratios of a pair of 2 classes (k and l):

$$LOR = \log\left(\frac{P(Y=k|X)}{P(Y=l|X)}\right), \text{ (the LOR} = 0 \Leftrightarrow \text{the two probabilities are identical, i.e., same class)}$$

$$LOR = \log\left(\frac{P(Y=k|X)}{P(Y=l|X)}\right) = (\mu_k - \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) = \frac{1}{2} (\mu_k^T \Sigma^{-1} \mu_k - \mu_l^T \Sigma^{-1} \mu_l).$$

But, in the more general, **quadratic case** of QDA, there are no assumptions on the covariance matrices Σ_k of the Gaussians, leading to more flexible quadratic decision surfaces separating the classes.

LDA (Linear Discriminant Analysis)

LDA is similar to GLM (e.g., ANOVA and regression analyses), as it also attempts to express one dependent variable as a linear combination of the other features or data elements. However, ANOVA uses categorical independent variables and a continuous dependent variable, whereas LDA has continuous independent variables and a categorical dependent variable (i.e., Dx/class label). Logistic regression and probit regression are more similar to LDA than ANOVA, as they also explain a categorical variable by the values of continuous independent variables.

```
predfun.Lda = function(train.x, train.y, test.x, test.y, neg)
{
  require("MASS")
  Lda.fit = lda(train.x, grouping=train.y)
  ynew = predict(Lda.fit, test.x)$class
  out.Lda = confusionMatrix(test.y, ynew, negative=neg)
  return( out.Lda )
}
```

QDA (Quadratic Discriminant Analysis)

Similarly to LDA, the QDA prediction function can be defined by:

```
predfun.qda = function(train.x, train.y, test.x, test.y, neg)
{
  require("MASS") # for lda function
  qda.fit = qda(train.x, grouping=train.y)
  ynew = predict(qda.fit, test.x)$class
  out.qda = confusionMatrix(test.y, ynew, negative=neg)
  return( out.qda )
}
```

21.7.4 Neural Networks

We already saw Artificial Neural Networks (NNs) in Chap. 11. Applying NNs is not straightforward. We have to create a design matrix with an indicator column for the response feature. In addition, we need to write a *predict function* to translate the output of `neuralnet()` into analytical forecasts.

```
# predict nn
library("neuralnet")
pred = function(nn, dat) {
  yhat = compute(nn, dat)$net.result
  yhat = apply(yhat, 1, which.max)-1
  return(yhat)
}

my.neural <- function (train.x, train.y, test.x,
test.y,method,Layer=c(5,5)){
  train.x <- as.data.frame(train.x)
  train.y <- as.data.frame(train.y)
  colnames(train.x) <- paste0('V', 1:ncol(X))
  colnames(train.y) <- "V1"
  train_y_ind = model.matrix(~factor(train.y$V1)-1)
  colnames(train_y_ind) = paste0('out', 0:1)
  train = cbind(train.x, train_y_ind)
  y_names = paste0('out', 0:1)
  x_names = paste0('V', 1:ncol(train.x))
  nn = neuralnet(
    paste(paste(y_names, collapse='+'),
          '~~',
          paste(x_names, collapse='+')),
    train,
    hidden=layer,
    linear.output=FALSE,
    lifesign='full', lifesign.step=1000)
#predict
predict.y <- pred(nn, test.x)
out <- crossval::confusionMatrix(test.y, predict.y, negative = 0)
return (out)
}

set.seed(1234)
cv.out.nn <- crossval::crossval(my.neural, scale(X), Y, K = 5, B =
1,layer=c(20,20),verbose = F) # scale predictors is necessary.

## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps: 63 error: 1.02185
time: 0.08 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps: 79 error: 1.01374
time: 0.2 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps: 73 error: 1.02399
time: 0.09 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps: 66 error: 1.03016
time: 0.09 secs
## hidden: 20, 20    thresh: 0.01    rep: 1/1    steps: 72 error: 1.01491
time: 0.11 secs
```

```
crossval::diagnosticErrors(cv.out.nn$stat)
##           acc      sens      spec      ppv      npv
## 0.9454976303 0.9016393443 0.9633333333 0.9090909091 0.9601328904
##      LOR
## 5.4841051313
```

Again the forecasting results on the PD dataset are quite good.

21.7.5 SVM

In Chap. 11, we also saw SVM classification. Let's try cross-validation using Linear and Gaussian (radial) kernel SVM. We may expect that linear SVM would achieve a similar result to Gaussian, or even better than Gaussian SVM, since this dataset has a large k (# features) compared with n (# cases), which we explored in detail in Chap. 11.

```
library("e1071")

my.svm <- function (train.x, train.y, test.x,
test.y, method, cost=1, gamma=1/ncol(dx_norm), coef0=0, degree=3){
  svm_l.fit <- svm(x = train.x, y=as.factor(train.y), kernel = method)
  predict.y <- predict(svm_l.fit, test.x)
  out <- crossval::confusionMatrix(test.y, predict.y, negative = 0)
  return (out)
}

# Linear kernel
set.seed(123)
cv.out.svml <- crossval::crossval(my.svm, as.data.frame(X), Y, K = 5, B = 1,
method = "Linear", cost=tune_svm$best.parameters$cost, verbose = F)
diagnosticErrors(cv.out.svml$stat)

##           acc      sens      spec      ppv      npv
## 0.9502369668 0.9098360656 0.9666666667 0.9173553719 0.9634551495
##      LOR
## 5.6789307585

# Gaussian kernel
set.seed(123)
cv.out.svmg <- crossval::crossval(my.svm, as.data.frame(X), Y, K = 5, B = 1,
method = "radial", cost=tune_svmg$best.parameters$cost, gamma=tune_svmg$best.parameters$gamma, verbose = F)
diagnosticErrors(cv.out.svmg$stat)

##           acc      sens      spec      ppv      npv
## 0.9454976303 0.9262295082 0.9533333333 0.8897637795 0.9694915254
##      LOR
## 5.5470977226
```

Indeed, both types of kernels yield good quality predictors according to the assessment metrics reported by the `diagnosticErrors()` method.

21.7.6 *k*-Nearest Neighbors Algorithm (*k*-NN)

As we saw in Chap. 7, *k*-NN is a non-parametric method for either classification or regression, where the **input** consists of the *k* closest **training examples** in the feature space, but the **output** depends on whether *k*-NN is used for classification or regression:

- In *k*-NN *classification*, the output is a class membership (labels). Objects in the testing data are classified by a majority vote of their neighbors. Each object is assigned to a class that is most common among its *k* nearest neighbors (*k* is always a small positive integer). When *k* = 1, then an object is assigned to the class of its single nearest neighbor.
- In *k*-NN *regression*, the output is the property value for the object representing the average of the values of its *k* nearest neighbors.

Let's now build the corresponding `predfun.knn()` method.

```
# X = as.matrix(input) # Predictor variables X = as.matrix(input.short2)
# Y = as.matrix(output) # Outcome
# KNN (k-nearest neighbors)
library("class")
# knn.fit.test <- knn(X, X, cl = Y, k=3, prob=F); predict(as.matrix(knn.fit.
test), X)$class
# table(knn.fit.test, Y); confusionMatrix(Y, knn.fit.test, negative="1")
# This can be used for polytomous variable (multiple classes)

predfun.knn = function(train.x, train.y, test.x, test.y, neg)
{
  require("class")
  knn.fit = knn(train.x, test.x, cl = train.y, prob=T) # knn is already
  a prediction function!!!
  # ynew = predict(knn.fit, test.x)$class # no need of another
  prediction, in this case
  out.knn = confusionMatrix(test.y, knn.fit, negative=neg)
  return( out.knn )
}
cv.out.knn = crossval::crossval(predfun.knn, X, Y, K=5, B=2, neg="1")

cv.out.knn = crossval::crossval(predfun.knn, X, Y, K=5, B=2, neg="1")
#Compare all 3 classifiers (lda, qda, knn, and logit)
diagnosticErrors(cv.out.Lda$stat); diagnosticErrors(cv.out.qda$stat);
diagnosticErrors(cv.out.qda$stat); diagnosticErrors(cv.out.Logit$stat);
```

We can also examine the performance of *k*-NN prediction on the PPMI (Parkinson's disease) data. Start by partitioning the data into **training** and **testing** sets.

```

# TRAINING: 75% of the sample size
sample_size <- floor(0.75 * nrow(input))
## set the seed to make your partition reproducible
set.seed(1234)
input.train.ind <- sample(seq_len(nrow(input)), size = sample_size)
input.train <- input[input.train.ind, ]
output.train <- as.matrix(output)[input.train.ind, ]

# TESTING DATA
input.test <- input[-input.train.ind, ]
output.test <- as.matrix(output)[-input.train.ind, ]

```

Then, we can fit the k-NN model and report the results.

```

library("class")
knn_model <- knn(train= input.train, input.test, cl=as.factor(output.train),
k=2)
#plot(knn_model)
summary(knn_model)
attributes(knn_model)

# cross-validation
knn_model.cv <- knn.cv(train= input.train, cl=as.factor(output.train), k=2)
summary(knn_model.cv)

```

21.7.7 k-Means Clustering (k-MC)

In Chap. 13, we showed that k-MC aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean, which acts as a prototype of a cluster. The k-MC partitions the data space into Voronoi cells. In general, there is no computationally tractable solution for this, i.e., the problem is NP-hard. However, there are efficient algorithms that converge quickly to local optima, e.g., the *expectation-maximization* algorithm for mixtures of Gaussian distributions via an iterative refinement approach (Figs. 21.5, 21.6 and 21.7).

```

kmeans_model <- kmeans(input.train, 2)
Layout(matrix(1, 1))
# tiff("C:/Users/User/Desktop/test.tiff", width = 10, height = 10, units =
' in', res = 300)
fpc::plotcluster(input.train, output.train, col = kmeans_model$cluster)

cluster::clusplot(input.train, kmeans_model$cluster, color=TRUE, shade=TRUE,
Labels=2, Lines=0)

```

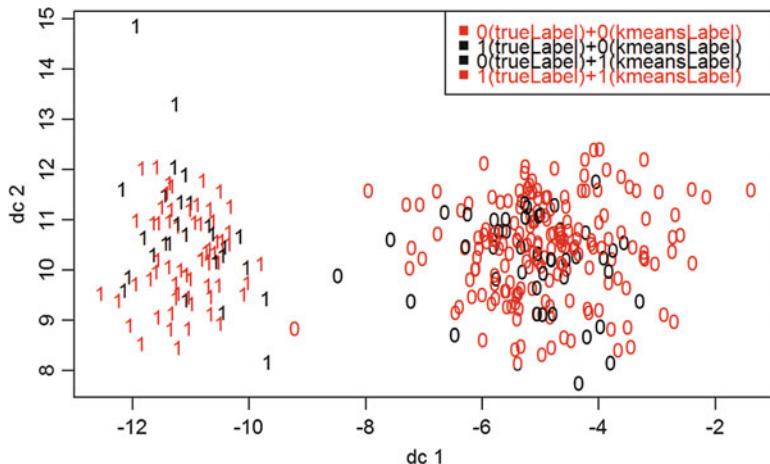


Fig. 21.5 k-Means clustering plot () of the Parkinson's disease data (PPMI)

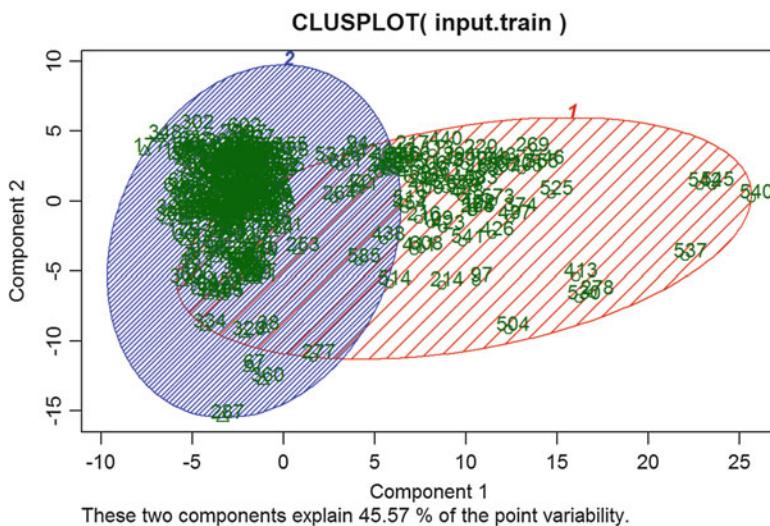


Fig. 21.6 Clusterplot of the k-means clustering of the PPMI data

```

par(mfrow=c(10,10))
# the next figure is very large and will not render in RStudio, you may need
# to save it as PDF file!
# pdf("C:/Users/User/Desktop/test.pdf", width = 50, height = 50)
# with(ppmi_data[,1:10], pairs(input.train[,1:10], col=c(1:2)[kmeans_model$c
# luster]))
# dev.off()
with(ppmi_data[,1:10], pairs(input.train[,1:10],
col=c(1:2)[kmeans_model$cluster]))

```

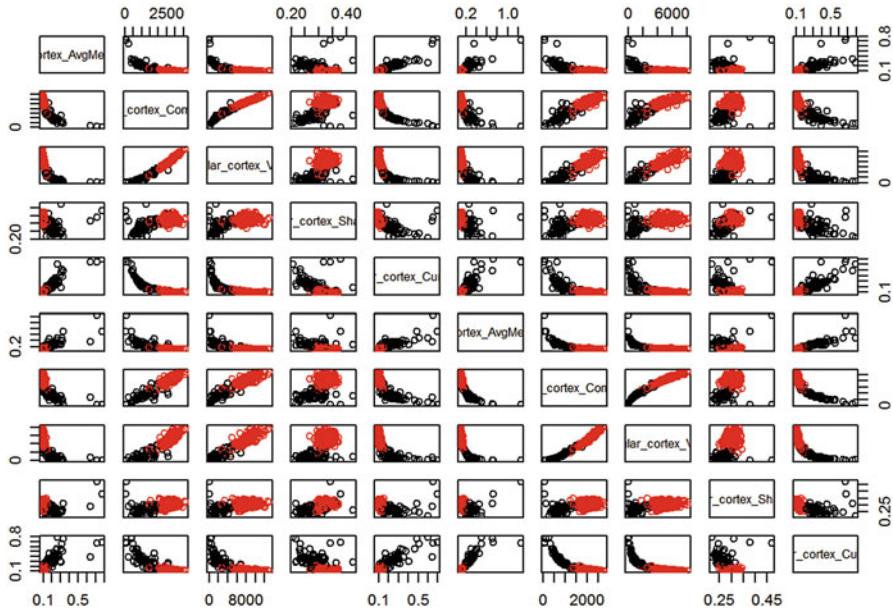


Fig. 21.7 Pair plots of the two clustering labels along the first 10 PPMI features

```

# plot(input.train, col = kmeans_model$cluster)
# points(kmeans_model$centers, col = 1:2, pch = 8, cex = 2)

## cluster centers "fitted" to each obs.:
fitted.kmeans <- fitted(kmeans_model); head(fitted.kmeans)

##      L_insular_cortex_AvgMeanCurvature L_insular_cortex_ComputeArea
## 2          0.1071299082                2635.580514
## 2          0.1071299082                2635.580514
## 1          0.2221893533                1134.578902
## 2          0.1071299082                2635.580514
## 2          0.1071299082                2635.580514
## 2          0.1071299082                2635.580514
##      L_insular_cortex_Volume L_insular_cortex_ShapeIndex
## 2          7969.485443                0.3250065829
## 2          7969.485443                0.3250065829
## 1          2111.385018                0.2788562513
## 2          7969.485443                0.3250065829
## 2          7969.485443                0.3250065829
## 2          7969.485443                0.3250065829
...
resid.kmeans <- (input.train - fitted(kmeans_model))

# define the sum of squares function
ss <- function(data) sum(scale(data, scale = FALSE)^2)

```

```

## Equalities
cbind(kmeans_model[c("betweenss", "tot.withinss", "totss")], # the same two
      columns
           c (ss(fitted.kmeans), ss(resid.kmeans), ss(input.train)))

##          [,1]      [,2]
## betweenss 15462062254 15462062254
## tot.withinss 12249286905 12249286905
## totss     27711349159 27711349159

# validation
stopifnot(all.equal(kmeans_model$totss,           ss(input.train)),
          all.equal(kmeans_model$tot.withinss, ss(resid.kmeans)),
          ## these three are the same:
          all.equal(kmeans_model$betweenss,    ss(fitted.kmeans)),
          all.equal(kmeans_model$betweenss, kmeans_model$totss -
kmeans_model$tot.withinss),
          ## and hence also
          all.equal(ss(input.train), ss(fitted.kmeans) + ss(resid.kmeans)))
)
# kmeans(input.train, 1)$withinss
# trivial one-cluster, (its W.SS == ss(input.train))
clust_kmeans2 = kmeans(scale(X), center=X[1:2,], iter.max=100,
algorithm='Lloyd')

```

We may get empty clusters, instead of two clusters, when we randomly select two points as the initial centers. The way to solve this problem is using k-means++.

```

# k++ initialize
kpp_init = function(dat, K) {
  x = as.matrix(dat)
  n = nrow(x)
  # Randomly choose a first center
  centers = matrix(NA, nrow=K, ncol=ncol(x))
  centers[1,] = as.matrix(x[sample(1:n, 1),])
  for (k in 2:K) {
    # Calculate dist^2 to closest center for each point
    dists = matrix(NA, nrow=n, ncol=k-1)
    for (j in 1:(k-1)) {
      temp = sweep(x, 2, centers[j,], '-')
      dists[,j] = rowSums(temp^2)
    }
    dists = rowMeans(dists)
    # Draw next center with probability proportional to dist^2
    cumdists = cumsum(dists)
    prop = runif(1, min=0, max=cumdists[n])
    centers[k,] = as.matrix(x[min(which(cumdists > prop)),])
  }
  return(centers)
}
clust_kmeans2_plus = kmeans(scale(X), kpp_init(scale(X), 2), iter.max=100,
algorithm='Lloyd')

```

Now let's evaluate the model. The first step is to justify the selection of $k=2$. We use the method `silhouette()` in package `cluster`. Recall from Chap. 14 that the *silhouette value* is between -1 and 1 . Negative silhouette values represent “mis-clustered” cases (Fig. 21.8).

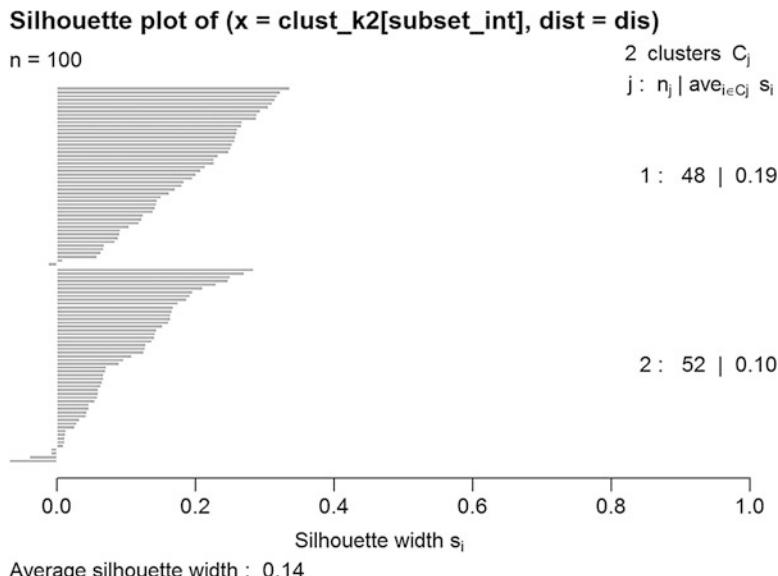


Fig. 21.8 Silhouette plot of the 2-class k-means clustering of the Parkonson's disease data

```

clust_k2 = clust_kmeans2_plus$cluster
require(cluster)

## Loading required package: cluster

# X = as.matrix(input.short2)
# as the data is too large for the silhouette plot, we'll just subsample and
plot 100 random cases
subset_int <- sample(nrow(X), 100) #100 cases from 661 total cases
dis = dist(as.data.frame(scale(X[subset_int,])))
sil_k2 = silhouette(clust_k2[subset_int], dis) #best
plot(sil_k2)

summary(sil_k2)

## Silhouette of 100 units in 2 clusters from silhouette.default(x = clust_k2[subset_int], dist = dis) :
##  Cluster sizes and average silhouette widths:
##        48      52
## 0.1895633766 0.1018642857
##  Individual silhouette widths:
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## -0.06886907 0.06533312 0.14169240 0.14395980 0.22658680 0.33585520

mean(sil_k2<0)
## [1] 0.016666666667

```

The result is pretty good. Only a very small number of samples are “mis-clustered” (having negative silhouette values). Furthermore, you can observe that when $k=3$ or $k=4$, the overall silhouette decreases, which indicates suboptimal clustering.

```
dis = dist(as.data.frame(scale(X)))
clust_kmeans3_plus = kmeans(scale(X), kpp_init(scale(X), 3), iter.max=100, a
lgorithm='Lloyd')
summary(silhouette(clust_kmeans3_plus$cluster,dis))

## Silhouette of 422 units in 3 clusters from silhouette.default(x =
clust_kmeans3_plus$cluster, dist = dis) :
## Cluster sizes and average silhouette widths:
##      139      157      126
## 0.08356111542 0.19458813829 0.17237138090
## Individual silhouette widths:
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## -0.06355399 0.08376430 0.16639550 0.15138420 0.21855670 0.33107050

clust_kmeans4_plus = kmeans(scale(X), kpp_init(scale(X), 4), iter.max=100, a
lgorithm='Lloyd')
summary(silhouette(clust_kmeans4_plus$cluster,dis))

## Silhouette of 422 units in 4 clusters from silhouette.default(x =
clust_kmeans4_plus$cluster, dist = dis) :
## Cluster sizes and average silhouette widths:
##      138      121      111      52
## 0.124165755516 0.170228092125 0.193359499726 0.008929262925
## Individual silhouette widths:
##      Min. 1st Qu. Median Mean 3rd Qu. Max.
## -0.16300240 0.08751445 0.15091580 0.14137370 0.21035560 0.32293680
```

Then, let's calculate the unsupervised classification error. Here, p represents the percentage of class 0 cases, which provides the weighting factor for labelling each cluster.

```
mat = matrix(1,nrow = Length(Y))
p = sum(Y==0)/Length(Y)
for (i in 1:2){
  id = which(clust_k2==i)
  if(sum(Y[id]==0)>Length(id)*p){
    mat[id] = 0
  }
}
caret::confusionMatrix(Y,mat)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##           0 195 105
##           1 1 121
##
##           Accuracy : 0.7488152
## 95% CI : (0.7045933, 0.7895087)
## No Information Rate : 0.535545
```

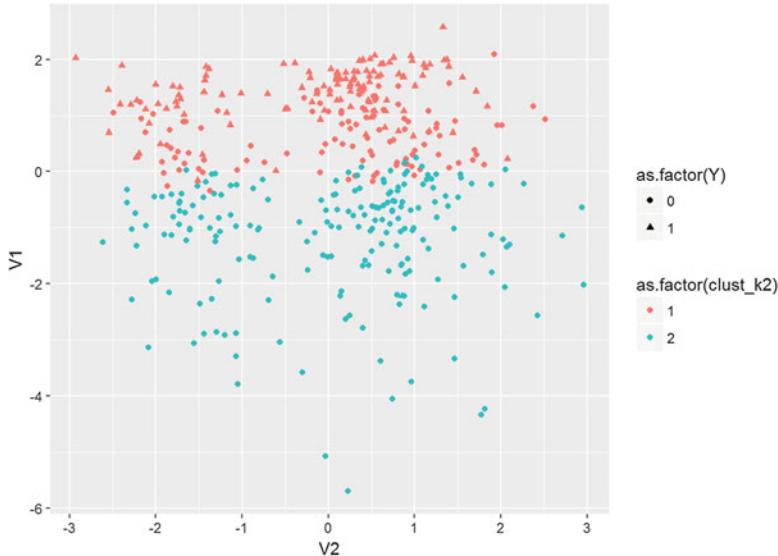


Fig. 21.9 Multi-dimensional scaling plot (2D projection) of the k-means clustering depicting the agreement between testing data labels (glyph shapes) and the predicted class labels (glyph colors)

```
##      P-Value [Acc > NIR] : < 0.000000000000022204
##
##          Kappa : 0.5122558
##  Mcnemar's Test P-Value : < 0.000000000000022204
##
##          Sensitivity : 0.9948980
##          Specificity : 0.5353982
##          Pos Pred Value : 0.6500000
##          Neg Pred Value : 0.9918033
##          Prevalence : 0.4644550
##          Detection Rate : 0.4620853
##  Detection Prevalence : 0.7109005
##          Balanced Accuracy : 0.7651481
##
##      'Positive' Class : 0
```

It achieves 69% accuracy, which is reasonable for unsupervised classification.

Finally, let's visualize the results by superimposing the data into the first two multi-dimensional scaling (MDS) dimensions (Fig. 21.9).

```
library("ggplot2")
mds = as.data.frame(cmdscale(dis, k=2))
mds_temp = cbind(
  mds, as.factor(clust_k2))
names(mds_temp) = c('V1', 'V2', 'cluster k=2')
gp_cluster = ggplot(mds_temp, aes(x=V2, y=V1, color=as.factor(clust_k2))) +
  geom_point(aes(shape = as.factor(Y))) + theme()
gp_cluster
```

21.7.8 Spectral Clustering

Suppose the multivariate dataset is represented as a set of data points A . We can define a similarity matrix $S = s_{(i,j)}$, where $s_{(i,j)}$ represents a measure of the similarity between points $i, j \in A$. Spectral clustering uses the spectrum of the similarity matrix of the high-dimensional data and performs dimensionality reduction for clustering into fewer dimensions. The spectrum of a matrix is the set of its eigenvalues. In general, if $M : \Omega \xrightarrow{\text{linear operator}} \Omega$ maps a vector space Ω into itself, its spectrum is the set of scalars $\lambda = \{\lambda_i\}$ such that $(T - \lambda I)v = 0$, where I is the identity matrix and v are the eigen vectors (or eigen-functions) for the operator T . The **determinant** of the matrix equals the product of its eigenvalues, i.e., $\det(T) = \prod_i \lambda_i$, the **trace** of the matrix $\text{tr}(T) = \sum_i \lambda_i$, and the **pseudo-determinant** for a singular matrix is the product of its nonzero eigenvalues, $\text{pseudo}_{\det}(T) = \prod_{\lambda_i \neq 0} \lambda_i$.

To partition the data into two sets (S_1, S_2) , denote v to be the second-smallest eigenvector of the Laplacian matrix:

$$L = I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$$

of the similarity matrix S , where D is the diagonal matrix $D_{i,i} = \sum_j S_{i,j}$.

This actual (S_1, S_2) partitioning of the cases in the data may be done in different ways. For instance, S_1 may use the median m of the components in v and group all data points whose component in v is greater than m . Then, the remaining cases can be labeled as part of S_2 . This approach may be used iteratively for *hierarchical clustering* by repeatedly partitioning the subsets.

The *specc* method in the *kernlab* package implements a spectral clustering algorithm where the data-clustering is performed by embedding the data into the subspace of the eigenvectors of an affinity matrix.

```
# install.packages("kernlab")
library("kernlab")

# review and choose a dataset (for example the Iris data
data()
#plot(iris)
```

Let's look at a few simple cases of spectral clustering. We are suppressing some of the outputs to save space (e.g., `#plot(my_data, col= data_sc)`).

Iris Petal Data

Let's look at the *iris* dataset we saw in Chap. 3.

```
my_data <- iris; data(my_data)
num_clusters <- 3
data_sc <- specc(my_data, centers= num_clusters)
data_sc
centers(data_sc)
withinss(data_sc)
#plot(my_data, col= data_sc)
```

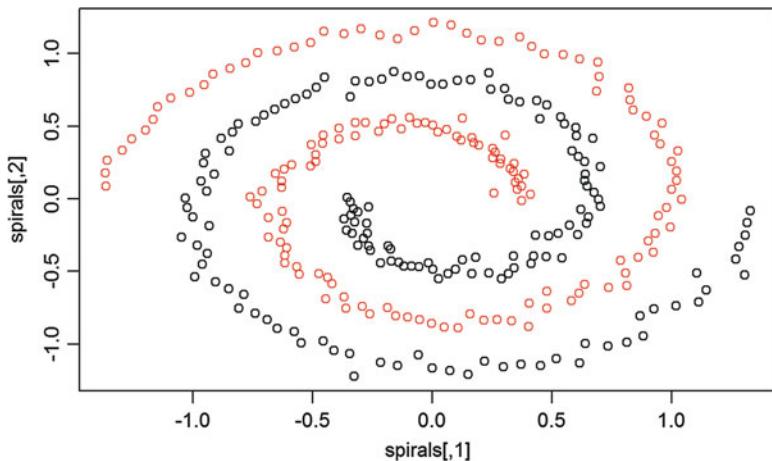


Fig. 21.10 Spirals data spectral clustering results

Spirals Data

Another simple dataset is `kernlab::spirals` (Fig. 21.10).

```

## 
## Within-cluster sum of squares:
## [1] 117.3429096 118.1182272

centers(data_sc)

##           [,1]           [,2]
## [1,] 0.01997200551 -0.1761483316
## [2,] -0.01770984369  0.1775136857

withinss(data_sc)

## [1] 117.3429096 118.1182272

plot(spirals, col= data_sc)

```

Income Data

A customer *income* dataset representing a marketing survey is included in `kernlab::income` (Fig. 21.11).

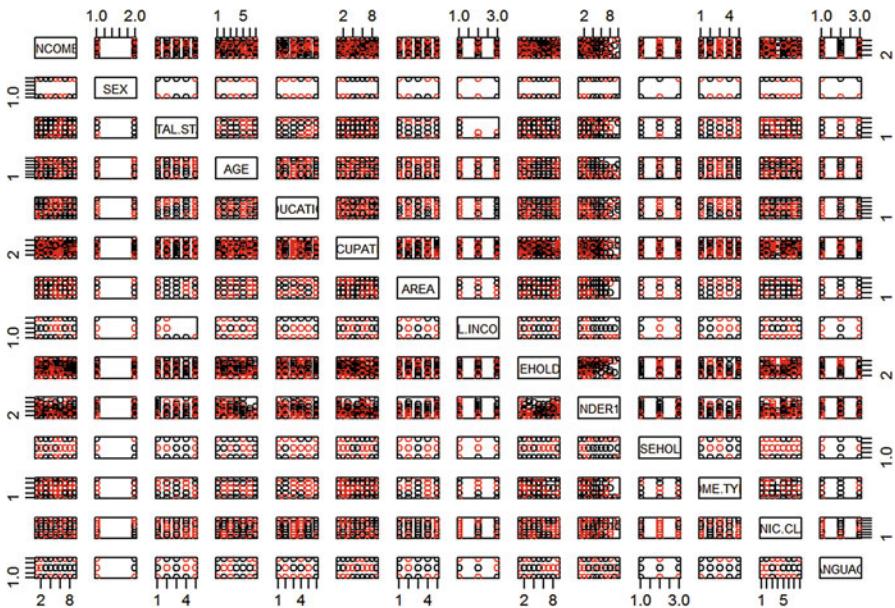


Fig. 21.11 Pair plots of the two-class spectral clustering of the income dataset

```

data(income)
num_clusters <- 2
data_sc <- specc(income, centers= num_clusters)
data_sc

## Spectral Clustering object of class "specc"
##
## Cluster memberships:
##
## 2 1 2 2 2 1 2 1 2 1 1 1 2 1
##
## String kernel function. Type = spectrum
## Hyperparameters : sub-sequence/string length = 4
## Normalized
##
## Cluster size:
## [1] 7 7

centers(data_sc)

##      [,1]
## [1,] NA

withinss(data_sc)

## Logical(0)

plot(income, col= data_sc)

```

21.8 Compare the Results

Now let's compare all eight classifiers (AdaBoost, LDA, QDA, knn, logit, Neural Network, linear SVM and Gaussian SVM) we presented above (Table 21.1).

```

# get AdaBoost CV results
set.seed(123)
cv.out.ada <- crossval::crossval(my.ada, as.data.frame(X), Y, K = 5, B = 1,
negative = neg)

## Number of folds: 5
## Total number of CV fits: 5
##
## Round # 1 of 1
## CV Fit # 1 of 5
## CV Fit # 2 of 5
## CV Fit # 3 of 5
## CV Fit # 4 of 5
## CV Fit # 5 of 5

# get k-Means CV results
my.kmeans <- function (train.x, train.y, test.x, test.y, negative, formula){
  kmeans.fit <- kmeans(scale(test.x), kpp_init(scale(test.x), 2),
  iter.max=100, algorithm='Lloyd')

```

Table 21.1 Comparison of alternative predictive analytic strategies for the PPMI dataset

	acc	sens	spec	ppv	npv	lor
AdaBoost	0.9739336493	0.9933333333	0.9262295082	0.9706840391	0.9826086957	7.5341095473
LDA	0.9617298578	0.9513333333	0.9872950820	0.9945983621	0.8918918919	7.3258499745
QDA	0.9407582938	0.9533333333	0.9098360656	0.9629629630	0.8880000000	5.3285694097
knn	0.6113744076	0.7133333333	0.3606557377	0.7328767123	0.3384615385	0.3391095260
logit	0.9431279621	0.9466666667	0.9344262295	0.9726027397	0.8769230769	5.5331424226
Neural Network	0.9454976303	0.9016393443	0.9633333333	0.9090909091	0.9601328904	5.4841051313
linear SVM	0.9502369668	0.9098360656	0.9666666667	0.9173553719	0.9634551495	5.6789307585
Gaussian SVM	0.9454976303	0.9262295082	0.9533333333	0.8897637795	0.9694915254	5.5470977226
k-Means	0.5331735555	0.5400000000	0.5163934426	0.7330316742	0.3134328358	0.2259399326
Spectral Clustering	0.5592417062	0.5900000000	0.4836065574	0.7375000000	0.3241758242	0.2983680947

```

predict.y <- kmeans.fit$cluster
#count TP, FP, TN, FN, Accuracy, etc.
out <- confusionMatrix(test.y, predict.y, negative = negative)
# negative is the label of a negative "null" sample (default: "control").
  return (out)
}
set.seed(123)
cv.out.kmeans <- crossval::crossval(my.kmeans, as.data.frame(X), Y, K = 5,
B = 2, negative = neg)

## Number of folds: 5
## Total number of CV fits: 10
##
## Round # 1 of 2
## CV Fit # 1 of 10
## CV Fit # 2 of 10
## CV Fit # 3 of 10
## CV Fit # 4 of 10
## CV Fit # 5 of 10
##
## Round # 2 of 2
## CV Fit # 6 of 10
## CV Fit # 7 of 10
## CV Fit # 8 of 10
## CV Fit # 9 of 10
## CV Fit # 10 of 10

# get spectral clustering CV results
my.sc <- function (train.x, train.y, test.x, test.y, negative, formula){
  sc.fit <- specc(scale(test.x), centers= 2)
  predict.y <- sc.fit@Data
  #count TP, FP, TN, FN, Accuracy, etc.
  out <- confusionMatrix(test.y, predict.y, negative = negative)
  # negative is the label of a negative "null" sample (default: "control").
  return (out)
}
set.seed(123)
cv.out.sc <- crossval::crossval(my.sc, as.data.frame(X), Y, K = 5, B = 2,
negative = neg)

## Number of folds: 5
## Total number of CV fits: 10
##
## Round # 1 of 2
## CV Fit # 1 of 10
## CV Fit # 2 of 10
## CV Fit # 3 of 10
## CV Fit # 4 of 10
## CV Fit # 5 of 10
##
## Round # 2 of 2
## CV Fit # 6 of 10
## CV Fit # 7 of 10
## CV Fit # 8 of 10
## CV Fit # 9 of 10
## CV Fit # 10 of 10

```

```

require(knitr)
## Loading required package: knitr

res_tab=rbind(diagnosticErrors(cv.out.ada$stat),diagnosticErrors(
cv.out.Lda$stat),diagnosticErrors(cv.out.qda$stat),diagnosticErrors(
cv.out.knn$stat),diagnosticErrors(cv.out.Logit$stat),diagnosticErrors(
cv.out.nn$stat),diagnosticErrors(cv.out.svml$stat),diagnosticErrors(
cv.out.svmg$stat),diagnosticErrors(cv.out.kmeans$stat),diagnosticErrors(
cv.out.sc$stat))
rownames(res_tab) <- c("AdaBoost", "LDA", "QDA", "knn", "Logit",
"Neural Network", "Linear SVM", "Gaussian SVM", "k-Means",
"Spectral Clustering")
kable(res_tab,caption = "Compare Result")

```

Leaving knn, kmeans and specc aside, the other methods achieve pretty good results. In the PD case study, the reason for suboptimal results in some clustering methods may be rooted in lack of training (e.g., specc and kmeans) or the curse of (high) dimensionality, which we saw in Chap. 7. As the data are rather sparse, predicting from the nearest neighbors may not be too reliable.

21.9 Assignment: 21. Prediction and Internal Statistical Cross-Validation

Demonstrate cross-validation on these two case-studies independently:

- Example 1: ALS (Amyotrophic Lateral Sclerosis)
- Example 2: Quality of Life in Chronic Illness (Case06_QoL_Symptom_ChronicIllness.csv)

Go through the following protocol:

- Review the case-study.
- Choose appropriate dichotomous, polytomous or continuous outcome variables, e.g., use ALSFRS_slope for ALS, CHRONICDISEASESCORE for case 06 and cast them as dichotomous outcomes.
- Apply appropriate data preprocessing.
- Perform regression modeling (e.g., OLS, glmnet, Forward or Backward model selection, etc.) for continuous outcomes.
- Perform classification and prediction using various methods (e.g., LDA, QDA, AdaBoost, SVM, Neural Network, KNN) for discrete outcomes.
- Apply cross-validation on these regression and classification methods, respectively.
- Report standard error for regression approaches.
- Report appropriate quality metrics that can be used to rank the forecasting approaches based on the predictive power of their results.
- Compare the result of model-driven and data-driven (e.g., KNN).

- Compare the sensitivity and specificity.
- Use unsupervised classification methods (*k-Means*) and spectral clustering.
- Evaluate and justify a *k-Means* model and report the agreement of the derived clusters and the real labels.
- Report the classification error of *k-means* and also compare with the result of *k-means++*.

References

- Elder, J, Nisbet, R, Miner, G (eds.) (2009) *Handbook of Statistical Analysis and Data Mining Applications*, Academic Press, ISBN 0080912036, 9780080912035.
- Hastie, T, Tibshirani, R, Friedman, J. (2013) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics*, New York, ISBN 1489905189, 9781489905185.
- Hothorn, T, Everitt, BS. (2014) *A Handbook of Statistical Analyses using R*, CRC Press, ISBN 1482204592, 9781482204599.
- https://en.wikipedia.org/wiki/Coefficient_of_determination
- <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0157077>

Chapter 22

Function Optimization



Most data-driven scientific inference, qualitative, quantitative, and visual analytics involve formulating, understanding the behavior of, and optimizing objective (cost) functions. Presenting the mathematical foundations of representation and interrogation of diverse spectra of objective functions provides mechanisms for obtaining effective solutions to complex big data problems. (Multivariate) *function optimization* (minimization or maximization) is the process of searching for variables $x_1, x_2, x_3, \dots, x_n$ that either minimize or maximize the multivariate cost function $f(x_1, x_2, x_3, \dots, x_n)$. In this chapter, we will specifically discuss (1) constrained and unconstrained optimization; (2) Lagrange multipliers; (3) linear, quadratic and (general) non-linear programming; and (4) data denoising.

22.1 Free (Unconstrained) Optimization

We will start with function optimization without restrictions for the domain of the cost function, $\Omega \ni \{x_i\}$. The extreme value theorem suggests that a solution to the free optimization processes, $\min_{x_1, x_2, x_3, \dots, x_n} f(x_1, x_2, x_3, \dots, x_n)$ or $\max_{x_1, x_2, x_3, \dots, x_n} f(x_1, x_2, x_3, \dots, x_n)$, may be obtained by a gradient vector descent method. This means that we can minimize/maximize the objective function by finding solutions to $\nabla f = \left\{ \frac{df}{dx_1}, \frac{df}{dx_2}, \dots, \frac{df}{dx_n} \right\} = \{0, 0, \dots, 0\}$. Solutions to this equation, x_1, \dots, x_n , will present candidate (local) minima and maxima.

In general, identifying critical points using the gradient or tangent plane, where the partial derivatives are trivial, may not be sufficient to determine the *extrema (minima or maxima)* of multivariate objective functions. Some critical points may represent inflection points, or local extrema that are far from the global *optimum* of the objective function. The eigenvalues of the Hessian matrix, which includes the second order partial derivatives, at the critical points provide clues to pinpoint extrema. For instance, invertible Hessian matrices that (i) are positive definite (i.e.,

all eigenvalues are positive), yield a local minimum at the critical point, (ii) are negative definite (all eigenvalues are negative) at the critical point suggests that the objective function has a local maximum, and (iii) have both positive and negative eigenvalues yield a saddle point for the objective function at the critical point where the gradient is trivial.

There are two complementary strategies to avoid being trapped in *local* extrema. First, we can run many iterations with different initial vectors. At each iteration, the objective function may achieve a (local) maximum/minimum/saddle point. Finally, we select the overall minimal (or maximal) value from all iterations. Another adaptive strategy involves either adjusting the step sizes or accepting solutions *in probability*, e.g., simulated annealing is one example of an adaptive optimization.

22.1.1 Example 1: Minimizing a Univariate Function (Inverse-CDF)

The cumulative distribution function (CDF) of a real-valued random process X , also known as the distribution function of X , represents the probability that the random variable X does not exceed a certain level. Mathematically speaking, the CDF of X is $F_X(x) = P(X \leq x)$. Recall the Chap. 2 discussions of Uniform, Normal, Cauchy, Binomial, Poisson and other discrete and continuous distributions. Also explore the dynamic representations of density and distribution functions included in the Probability Distributome Calculators (<http://distributome.org>).

For each $p \in [0, 1]$, the *inverse distribution function*, also called *quantile function* (e.g., `qnorm`), yields the critical value (x) at which the probability of the random variable is less than or equal to the given probability (p). When the CDF F_X is continuous and strictly increasing, the value of the inverse CDF at p , $F^{-1}(p) = x$, is the unique real number x such that $F(x) = p$.

Below, we will plot the probability density function (PDF) and the CDF for *Normal* distribution in R (Fig. 22.1).

```
par(mfrow=c(1,2), mar=c(3,4,4,2))
z<-seq(-4, 4, 0.1) # points from -4 to 4 in 0.1 steps
q<-seq(0.001, 0.999, 0.001) # probability quantile values from 0.1%
# to 99.9% in 0.1% steps
dStandardNormal <- data.frame(Z=z, Density=dnorm(z, mean=0, sd=1),
Distribution=pnorm(z, mean=0, sd=1))
plot(z, dStandardNormal$Density, col="darkblue", xlab="z",
ylab="Density", type="l", lwd=2, cex=2, main="Standard Normal PDF",
cex.axis=0.8)
# could also do
# xseq<-seq(-4, 4, 0.01); density<-dnorm(xseq, 0, 1); plot (density,
main="Density")
# Compute the CDF
xseq<-seq(-4, 4, 0.01); cumulative<-pnorm(xseq, 0, 1)
# plot (cumulative, main="CDF")
plot(xseq, cumulative, col="darkred", xlab="", ylab="Cumulative
Probability", type="l", lwd=2, cex=2, main="CDF of (Simulated)
Standard Normal", cex.axis=.8)
```

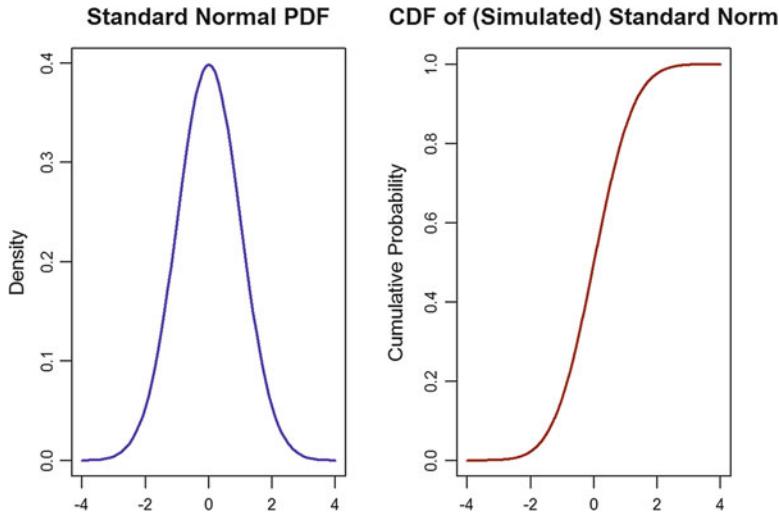


Fig. 22.1 Plots of the density and cumulative distribution functions of the simulated data

Suppose we are interested in computing, or estimating, the inverse-CDF from first principles. Specifically, to invert the CDF, we need to be able to solve the following equation (representing our objective function):

$$CDF(x) - p = 0.$$

The `uniroot` and `stats::nlm` R functions do *non-linear minimization* of a function f using a Newton-Raphson algorithm.

```
set.seed(1234)
x <- rnorm(1000, 100, 20)
pdf_x <- density(x)

# Interpolate the density, the values returned when input x values are outside [min(x): max(x)] should be trivial
f_x <- approxfun(pdf_x$x, pdf_x$y, yleft=0, yright=0)

# Manual computation of the cdf by numeric integration
cdf_x <- function(x){
  v <- integrate(f_x, -Inf, x)$value
  if (v<0) v <- 0
  else if(v>1) v <- 1
  return(v)
}

# Finding the roots of the inverse-CDF function by hand (CDF(x)-p=0)
invcdf <- function(p){
  uniroot(function(x){cdf_x(x) - p}, range(x))$root
  # alternatively, can use
  # nlm(function(x){cdf_x(x) - p}, 0)$estimate
  # minimum - the value of the estimated minimum of f.
  # estimate - the point at which the minimum value of f is obtained.
}
```

```
invcdf(0.5)
## [1] 99.16995
# We can validate that the inverse-CDF is correctly computed: F^{-1}(F(x)) ==
x
cdf_x(invcdf(0.8))
## [1] 0.8
```

The ability to compute exactly, or at least estimate, the inverse-CDF function is important for many reasons. For instance, generating random observations from a specified probability distribution (e.g., normal, exponential, or gamma distribution) is an important task in many scientific studies. One approach for such random number generation from a specified distribution evaluates the inverse CDF at random uniform $u \sim U(0, 1)$ values. Recall that in Chap. 16 we showed an example of generating random uniform samples using atmospheric noise. The key step is ability to quickly, efficiently and reliably estimate the inverse CDF function, of which we just showed one example.

Let's see why inverting the CDF using random uniform data works. Consider the cumulative distribution function (CDF) of a probability distribution from which we are interested in sampling. If the CDF has a closed form analytical expression and is invertible, then we generate a random sample from that distribution by evaluating the inverse CDF at u , where $u \sim U(0, 1)$. This is possible since a continuous CDF, F , is a one-to-one mapping of the domain of the CDF (range of X) into the interval $[0, 1]$. Therefore, if U is a uniform random variable on $[0, 1]$, then $X = F^{-1}(U)$ has the distribution F . Suppose $U \sim Uniform[0, 1]$, then $P(F^{-1}(U) \leq x) = P(U \leq F(x))$, by applying F to both sides of this inequality, since F is monotonic. Thus, $P(F^{-1}(U) \leq x) = F(x)$, since $P(U \leq u) = u$ for uniform random variables.

22.1.2 Example 2: Minimizing a Bivariate Function

Let's look at the function $f(x_1, x_2) = (x_1 - 3)^2 + (x_2 + 4)^2$. We define the function in R and utilize the `optim()` function to obtain the extrema points in the support of the objective function and/or the extrema values at these critical points.

```
require("stats")
f <- function(x) { (x[1] - 3)^2 + (x[2] + 4)^2 }
initial_x <- c(0, -1)
x_optimal <- optim(initial_x, f, method="CG") # performs minimization
x_min <- x_optimal$par
# x_min contains the domain values where the (local) minimum is attained
x_min # critical point/vector
## [1] 3 -4
x_optimal$value # extrema value of the objective function
## [1] 8.450445e-15
```

`optim` allows the use of six candidate optimization strategies:

- **Nelder-Mead**: robust but relatively slow, works reasonably well for non-differentiable functions.
- **BFGS**: quasi-Newton method (also known as a variable metric algorithm), uses function values and gradients to build up a picture of the surface to be optimized.
- **CG**: conjugate gradients method, fragile, but successful in larger optimization problems because it's unnecessary to save large matrix.
- **L-BFGS-B**: allows box constraints.
- **SANN**: a variant of simulated annealing, belonging to the class of stochastic global optimization methods.
- **Brent**: for one-dimensional problems only, useful in cases where `optim()` is used inside other functions where only method can be specified.

22.1.3 Example 3: Using Simulated Annealing to Find the Maximum of an Oscillatory Function

Consider the function $f(x) = 10 \sin(0.3x) \times \sin(1.3x^2) - 0.00002x^4 + 0.3x + 35$. Maximizing $f()$ is equivalent to minimizing $-f()$. Let's plot this oscillatory function, then find and report its critical points and extremum values.

The function `optim` returns two important results:

- `par`: the best set of domain parameters found to optimize the function
- `value`: the extreme values of the function corresponding to `par` (Fig. 22.2).

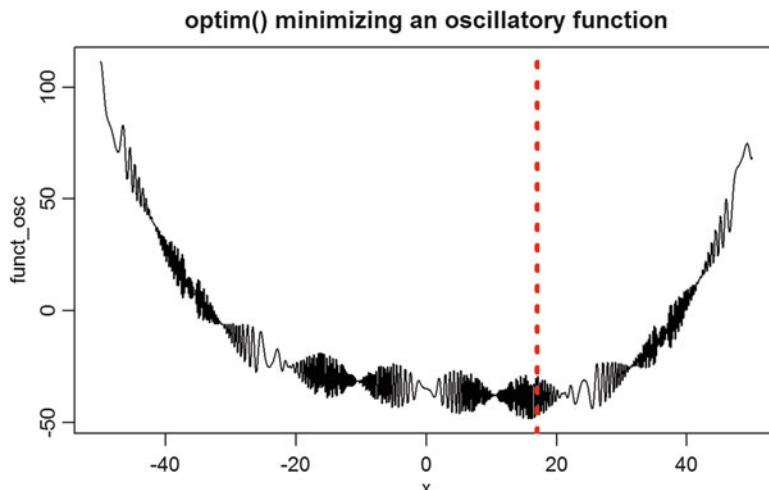


Fig. 22.2 Example of minimizing and oscillatory function, $f(x) = 10 \sin(0.3x) \times \sin(1.3x^2) - 0.00002x^4 + 0.3x + 35$, using `optim`

```

funct_osc <- function (x) { -(10*sin(0.3*x)*sin(1.3*x^2) - 0.00002*x^4 +
0.3*x+35) }
plot(funct_osc, -50, 50, n = 1000, main = "optim() minimizing an oscillatory
function")
abline(v=17, lty=3, lwd=4, col="red")

res <- optim(16, funct_osc, method = "SANN", control = list(maxit = 20000, t
emp = 20, parscale = 20))
res$par
## [1] 15.66197
res$value
## [1] -48.49313

```

22.2 Constrained Optimization

22.2.1 Equality Constraints

When there are support restrictions, dependencies, or other associations between the domain variables x_1, x_2, \dots, x_n , constrained optimization needs to be applied.

For example, we can have k equations specifying these restrictions, which may specify certain model characteristics:

$$\begin{cases} g_1(x_1, x_2, \dots, x_n) = 0 \\ \dots \\ g_k(x_1, x_2, \dots, x_n) = 0 \end{cases}.$$

Note that the right hand sides of these equations may always be assumed to be trivial (0), otherwise we can just move the non-trivial parts within the constraint functions g_i . Linear Programming, Quadratic Programming, and Lagrange multipliers may be used to solve such equality-constrained optimization problems.

22.2.2 Lagrange Multipliers

We can merge the equality constraints within the objective function ($f \rightarrow f^*$). Lagrange multipliers represent a typical solution strategy that turns the *constrained* optimization problem ($\min_x f(x)$ subject to $g_i(x_1, x_2, \dots, x_n), 1 \leq i \leq k$), into an *unconstrained* optimization problem:

$$f^*(x_1, x_2, \dots, x_n; \lambda_1, \lambda_2, \dots, \lambda_k) = f(x_1, x_2, \dots, x_n) + \sum_{i=1}^k \lambda_i g_i(x_1, x_2, \dots, x_n).$$

Then, we can apply traditional unconstrained optimization schemas, e.g., extreme value theorem, to minimize the unconstraint problem:

$$f^*(x_1, x_2, \dots, x_n; \lambda_1, \lambda_2, \dots, \lambda_k) = f(x_1, x_2, \dots, x_n) + \lambda_1 g_1(x_1, x_2, \dots, x_n) + \dots + \lambda_k g_k(x_1, x_2, \dots, x_n).$$

This represents an unconstrained optimization problem using Lagrange multipliers.

The solution of the constrained problem is also a solution to:

$$\nabla f^* = \left[\frac{df}{dx_1}, \frac{df}{dx_2}, \dots, \frac{df}{dx_n}; \frac{df}{d\lambda_1}, \frac{df}{d\lambda_2}, \dots, \frac{df}{d\lambda_k} \right] = [0, 0, \dots, 0].$$

22.2.3 Inequality Constrained Optimization

There are no general solutions for arbitrary inequality constraints; however, partial solutions do exist when some restrictions on the form of constraints are present.

When both the constraints and the objective function are linear functions of the domain variables, then the problem can be solved by Linear Programming.

Linear Programming (LP)

LP works when the objective function is a linear function. The constraint functions are also linear combination of the same variables.

Consider the following elementary (minimization) example:

$$\min_{x_1, x_2, x_3} (-3x_1 - 4x_2 - 3x_3)$$

subject to:

$$\begin{cases} 6x_1 + 2x_2 + 4x_3 \leq 150 \\ x_1 + x_2 + 6x_3 \geq 0 \\ 4x_1 + 5x_2 + 4x_3 = 40 \end{cases}.$$

The exact solution is $x_1 = 0$, $x_2 = 8$, $x_3 = 0$, and can be computed using the package `lpSolveAPI` to set up the constraint problem and the generic `solve()` method to find its solutions.

```

# install.packages("lpSolveAPI")
library(lpSolveAPI)

lps.model <- make.lp(0, 3) # define 3 variables
# add the constraints as a matrix of the Linear coefficients, relations and
# RHS
add.constraint(lps.model, c(6, 2, 4), "<=", 150)
add.constraint(lps.model, c(1, 1, 6), ">=", 0)
add.constraint(lps.model, c(4, 5, 4), "= ", 40)
# set objective function (default: find minimum)
set.objfn(lps.model, c(-3, -4, -3))
# you can save the model to a file
# write.lp(lps.model, 'c:/Users/LPmodel.lp', type='lp')

# these commands define the constraint Linear model
# /* Objective function */
#   min: -3 x1 -4 x2 -3 x3;
#
# /* Constraints */
# +6 x1 +2 x2 +4 x3 <= 150;
# + x1 + x2 +6 x3 >= 0;
# +4 x1 +5 x2 +4 x3 = 40;
#
# writing it in the text file named 'LPmodel.lp'

solve(lps.model)
## [1] 0

# Retrieve the values of the variables from a solved Linear program model
get.variables(lps.model) # check against the exact solution x_1 = 0,
x_2 = 8, x_3 = 0
## [1] 0 8 0
get.objective(lps.model) # get optimal (min) value
## [1] -32

```

In lower dimensional problems, we can also plot the constraints to graphically demonstrate the corresponding support restriction. For instance, here is an example of a simpler 2D constraint and its Venn diagrammatic representation (Fig. 22.3).

$$\begin{cases} x_1 \leq \frac{150 - 2x_2}{6} \\ x_1 \geq -x_2 \end{cases}.$$

```

library(ggplot2)
ggplot(data.frame(x = c(-100, 0)), aes(x = x)) +
  stat_function(fun=function(x) {(150-2*x)/6}, aes(color="Function 1")) +
  stat_function(fun=function(x) { -x }, aes(color = "Function 2")) +
  theme_bw() +
  scale_color_discrete(name = "Function") +
  geom_polygon(

```

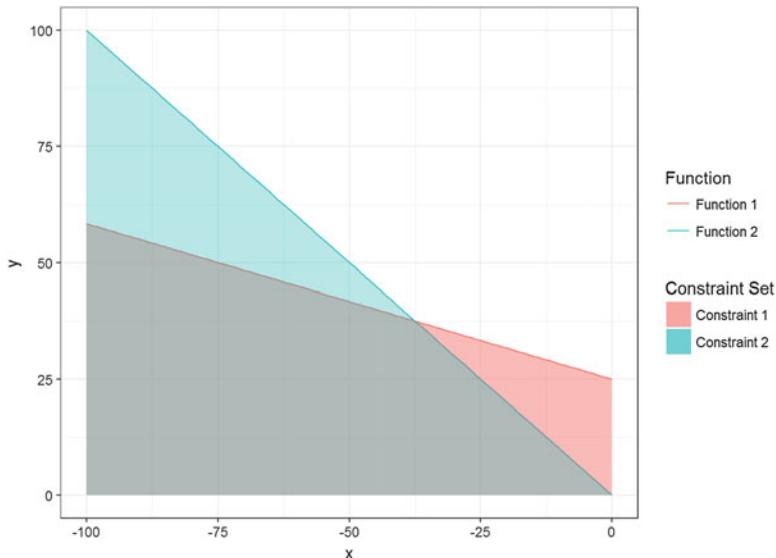


Fig. 22.3 A 2D graphical depiction of the function optimization support restriction constraints

```

data = data.frame(x = c(-100, -100, 0, 0, Inf), y = c(0, 350/6, 150/6,
0, 0)),
aes(x = x, y = y, fill = "Constraint 1"),
inherit.aes = FALSE, alpha = 0.5) +
geom_polygon(
  data = data.frame(x = c(-100, -100, 0, Inf), y = c(0, 100, 0, 0)),
  aes(x = x, y = y, fill = "Constraint 2"),
  inherit.aes = FALSE, alpha = 0.3) +
scale_fill_discrete(name = "Constraint Set") +
scale_y_continuous(limits = c(0, 100))

```

Here is another example of maximization of a trivariate cost function, $f(x_1, x_2, x_3) = 3x_1 + 4x_2 - x_3$, subject to:

$$\begin{cases} -x_1 + 2x_2 + 3x_3 \leq 16 \\ 3x_1 - x_2 - 6x_3 \geq 0 \\ x_1 - x_2 \leq 2 \end{cases} .$$

```
lps.model2 <- make.lp(0, 3)
add.constraint(lps.model2, c(-1, 2, 3), "<=", 16)
add.constraint(lps.model2, c(3, -1, -6), ">=", 0)
add.constraint(lps.model2, c(1, -1, 0), "<=", 2)
set.objfn(lps.model2, c(3, 4, -1), indices = c(1, 2, 3))
lp.control(lps.model2, sense='max')      # changes to max: 3 x1 + 4 x2 - x3

## $anti.degen
## [1] "fixedvars" "stalling"
##
## $basis.crash
## [1] "none"
##
## $bb.depthlimit
## [1] -50
##
## $bb.floorfirst
## [1] "automatic"
##
## $bb.rule
## [1] "pseudononint" "greedy"      "dynamic"      "rcostfixing"
##
## $break.at.first
## [1] FALSE
##
## $break.at.value
## [1] 1e+30
##
## $epsilon
##      epsb      epsd      epsel      epsint  epsperturb  epspivot
##      1e-10    1e-09    1e-12    1e-07    1e-05    2e-07
##
## $improve
## [1] "dualfeas" "thetagap"
##
## $infinite
## [1] 1e+30
##
## $maxpivot
## [1] 250
##
## $mip.gap
## absolute relative
##      1e-11    1e-11
##
## $negrange
## [1] -1e+06
##
## $obj.in.basis
## [1] TRUE
##
## $pivoting
## [1] "devex"    "adaptive"
##
## $presolve
## [1] "none"
```

```

## 
## $scalelimit
## [1] 5
##
## $scaling
## [1] "geometric"    "equilibrate" "integers"
##
## $sense
## [1] "maximize"
##
## $simplextype
## [1] "dual"    "primal"
##
## $timeout
## [1] 0
##
## $verbose
## [1] "neutral"

solve(lps.model2)           # 0 suggests that this solution converges
## [1] 0

get.variables(lps.model2) # get point of maximum
## [1] 20 18  0

get.objective(lps.model2) # get optimal (max) value
## [1] 132

```

In 3D, we can utilize the `rgl::surface3d()` method to display the constraints. This output is suppressed, as it can only be interpreted via the pop-out 3D rendering window.

```

library("rgl")
n <- 100
x <- y <- seq(-500, 500, Length = n)
region <- expand.grid(x = x, y = y)

z1 <- matrix(((150 -2*region$x -4*region$y)/6), n, n)
z2 <- matrix(-region$x + 6*region$y, n, n)
z3 <- matrix(40 -5*region$x - 4*region$y, n, n)

surface3d(x, y, z1, back = 'line', front = 'line', col = 'red', Lwd = 1.5,
alpha = 0.4)
surface3d(x, y, z2, back = 'line', front = 'line', col = 'orange', Lwd =
1.5, alpha = 0.4)
surface3d(x, y, z3, back = 'line', front = 'line', col = 'blue', Lwd = 1.5,
alpha = 0.4)
axes3d()

```

It is possible to restrict the domain type to contain only solutions that are:

- *integers*, which makes it an Integer Linear Programming (ILP),
- *binary/boolean* values (BLP), or
- *mixed* types, Mixed Integer Liner Programming (MILP).

Some examples are included below.

Mixed Integer Linear Programming (MILP)

Let's demonstrate MILP with an example where the type of x_1 is unrestricted, x_2 is dichotomous (binary), and x_3 is restricted to be an integer.

```

lps.model <- make.lp(0, 3)
add.constraint(lps.model, c(6, 2, 4), "<=", 150)
add.constraint(lps.model, c(1, 1, 6), ">=", 0)
add.constraint(lps.model, c(4, 5, 4), "=", 40)
set.objfn(lps.model, c(-3, -4, -3))

set.type(lps.model, 2, "binary")
set.type(lps.model, 3, "integer")
get.type(lps.model) # This is Mixed Integer Linear Programming (MILP)

## [1] "real"    "integer"  "integer"

set.bounds(lps.model, Lower=-5, upper=5, columns=c(1))

# give names to columns and restrictions
dimnames(lps.model) <- list(c("R1", "R2", "R3"), c("x1", "x2", "x3"))

print(lps.model)

## Model name:
##          x1    x2    x3
## Minimize -3    -4    -3
## R1        6     2     4  <= 150
## R2        1     1     6  >= 0
## R3        4     5     4   = 40
## Kind     Std   Std   Std
## Type     Real  Int   Int
## Upper    5     1     Inf
## Lower   -5     0     0

solve(lps.model)

## [1] 0

get.objective(lps.model)

## [1] -30.25

get.variables(lps.model)

## [1] 4.75 1.00 4.00

get.constraints(lps.model)

## [1] 46.50 29.75 40.00

```

The next example limits all three variable to be dichotomous (binary).

```

lps.model <- make.lp(0, 3)
add.constraint(lps.model, c(1, 2, 4), "<=", 5)
add.constraint(lps.model, c(1, 1, 6), ">=", 2)
add.constraint(lps.model, c(1, 1, 1), "=", 2)
set.objfn(lps.model, c(2, 1, 2))

set.type(lps.model, 1, "binary")
set.type(lps.model, 2, "binary")
set.type(lps.model, 3, "binary")

print(lps.model)

## Model name:
##          C1   C2   C3
## Minimize  2    1    2
## R1        1    2    4  <=  5
## R2        1    1    6  >=  2
## R3        1    1    1   =  2
## Kind     Std  Std  Std
## Type     Int  Int  Int
## Upper    1    1    1
## Lower    0    0    0

solve(lps.model)
## [1] 0

get.variables(lps.model)
## [1] 1 1 0

```

22.2.4 Quadratic Programming (QP)

QP can be used for second order (quadratic) objective functions, but the constraint functions are still linear combinations of the domain variables.

A matrix formulation of the problem can be expressed as minimizing an objective function:

$$f(X) = \frac{1}{2} X^T D X - d^T X,$$

where X is a vector $[x_1, x_2, \dots, x_n]^T$, D is the matrix of weights of each association pair, x_i , x_j , and d are the weights for each individual feature, x_i . The $\frac{1}{2}$ coefficient ensures that the weights matrix D is symmetric and each x_i , x_j pair is not double-counted. This cost function is subject to the constraints:

$$A^T X [= | \geq] b,$$

where the first k constraints may represent equalities ($=$) and the remaining ones are inequalities (\geq), and b is the constraints right hand size (RHS) constant vector.

Here is an example of a QP objective function and its R optimization:

$$f(x_1, x_2, x_3) = 2x_1^2 - x_1x_2 - 2x_2^2 + x_2x_3 + 2x_3^2 - 5x_2 + 3x_3.$$

Subject to the following constraints:

$$\begin{aligned} -4x_1 + -3x_2 &= -8 \\ 2x_1 + x_2 &= 2 \\ -2x_2 + x_3 &\geq 0 \end{aligned}$$

```
library(quadprog)

Dmat      <- matrix(c( 2, -1, 0,
                      -1, 2, -1,
                      0, -1, 2), 3, 3)
dvec      <- c(0, -5, 3)
Amat      <- matrix(c(-4, -3, 0,
                      2, 1, 0,
                      0, -2, 1), 3, 3)
bvec      <- c(-8, 2, 0)
n.eqs     <- 2 # the first two constraints are equalities
sol <- solve.QP(Dmat, dvec, Amat, bvec=bvec, meq=2)
sol$solution # get the (x1, x2, x3) point of minimum
## [1] -1 4 8
sol$value # get the actual cost function minimum
## [1] 49
```

The minimum value, 49, of the QP solution is attained at $x_1 = -1, x_2 = 4, x_3 = 8$.

When D is a positive definitive matrix, i.e., $X^TDX > 0$, for all non-zero X , the QP problem may be solved in polynomial time. Otherwise, the QP problem is NP-hard. In general, even if D has only one negative eigenvalue, the QP problem is still NP-hard.

The QP function `solve.QP()` expects a positive definitive matrix D .

22.3 General Non-linear Optimization

The package `Rsolnp` provides a special function `solnp()`, which solves the general non-linear programming problem:

$$\min_x f(x)$$

subject to:

$$\begin{aligned} g(x) &= 0 \\ l_h &\leq h(x) \leq u_h \\ l_x &\leq x \leq u_x, \end{aligned}$$

where $f(x), g(x), h(x)$ are all smooth functions.

22.3.1 *Dual Problem Optimization*

Duality in math really just means having two complementary ways to think about an optimization problem. The *primal problem* represents an optimization challenge in terms of the original decision variable x . The *dual problem*, also called *Lagrange dual*, searches for a lower bound of a minimization problem or an upper bound for a maximization problem. In general, the primal problem may be difficult to analyze, or solve directly, because it may include non-differentiable penalty terms, e.g., l_1 norms, recall LASSO/Ridge regularization in Chap. 18. Hence, we turn to the corresponding *Lagrange dual problem* where the solutions may be more amenable, especially for convex functions, that satisfy the following inequality:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Motivation

Suppose we want to borrow money, x , from a bank, or lender, and $f(x)$ represents the borrowing cost to us. There are natural “design constraints” on money lending. For instance, there may be a cap in the interest rate, $h(x) \leq b$, or we can have many other constraints on the loan duration. There may be multiple lenders, including self-funding, that may “charge” us $f(x)$ for lending us x . *Lenders goals are to maximize profits*. Yet, they can’t charge you more than the prime interest rate, plus some premium based on your credit worthiness. Thus, for a given fixed λ , a lender may make us an offer to lend us x aiming to minimize

$$f(x) + \lambda \times h(x).$$

If this cost is not optimized, i.e., minimized, you may be able to get another loan y at lower cost $f(y) < f(x)$, and the funding agency loses your business. If the cost/objective function is minimized, the lender may maximize their profit by varying λ and still get us to sign on the loan.

The customer’s strategy represents a *game theoretic interpretation* of the **primal problem**, whereas the **dual problem** corresponds to the strategy of the lender.

In solving complex optimization problems, *duality* is equivalent to existence of a saddle point of the Lagrangian. For convex problems, the double-dual is *equivalent* to the primal problem. In other words, applying the convex conjugate (Fenchel transform) twice returns the *convexification* of the original objective function, which in most situations is the same as the original function.

The *dual of a vector space* is defined as the space of all continuous linear functionals on that space. Let $X = R^n$, $Y = R^m$, $f: X \rightarrow R$, and $h: X \rightarrow Y$. Consider the following optimization problem:

$$\min_x f(x)$$

subject to

$$x \in X$$

$$h(x) \leq 0.$$

Then, this *primal problem* has a corresponding *dual problem*:

$$\min_{\lambda} \inf_{x \in X} (f(x) + \langle \lambda, h(x) \rangle)$$

subject to

$$\lambda_i \geq 0, \forall 0 \leq i \leq m.$$

The parameter $\lambda \in R^m$ is an element of the dual space of Y , i.e., Y^* , since the inner product $\langle \lambda, h(x) \rangle$ is a *continuous linear functional on Y* . Here Y is finite dimensional and by the Riesz representation theorem Y^* is isomorphic to Y . Note that in general, for infinite dimensional spaces, Y and Y^* are not guaranteed to be isomorphic.

Example 1: Linear Example

Minimize $f(x, y) = 5x - 3y$, constrained by $x^2 + y^2 = 136$, which has a minimum value of -68 attained at $(-10, 6)$. We will use the `Rsolnp::solnp()` method in this example.

```
# install.packages("Rsolnp")
library(Rsolnp)

fn1 <- function(x) { # f(x, y) = 5x-3y
  5*x[1] - 3*x[2]
}

# constraint z1: x^2+y^2=136
eqn1 <- function(x) {
  z1=x[1]^2 + x[2]^2
  return(c(z1))
}
constraints = c(136)

x0 <- c(1, 1) # setup initial values
sol1 <- solnp(x0, fun = fn1, eqfun = eqn1, eqB = constraints)

## Iter: 1 fn: 37.4378  Pars: 30.55472 38.44528
## Iter: 2 fn: -147.9181  Pars: -6.57051 38.35517
```

```

## Iter: 3 fn: -154.7345      Pars: -20.10545 18.06907
## Iter: 4 fn: -96.4033      Pars: -14.71366 7.61165
## Iter: 5 fn: -72.4915      Pars: -10.49919 6.66517
## Iter: 6 fn: -68.1680      Pars: -10.04485 5.98124
## Iter: 7 fn: -68.0006      Pars: -9.99999 6.00022
## Iter: 8 fn: -68.0000      Pars: -10.00000 6.00000
## Iter: 9 fn: -68.0000      Pars: -10.00000 6.00000
## solnp--> Completed in 9 iterations

sol1$values[10] # sol1$values contains all steps of the iteration algorithm
and the last value is the min value

## [1] -68

sol1$pars

## [1] -10 6

```

Example 2: Quadratic Example

Minimize $f(x, y) = 4x^2 + 10y^2 + 5$ subject to the inequality constraint $0 \leq x^2 + y^2 \leq 4$, which has a minimum value of 5 attained at the origin $(0, 0)$.

```

fn2 <- function(x) { # f(x, y) = 4x^2 + 10y^2 + 5
  4*x[1]^2 + 10*x[2]^2 + 5
}

# constraint z1: x^2+y^2 <= 4
ineq2 <- function(x) {
  z1=x[1]^2 + x[2]^2
  return(c(z1))
}

Lh <- c(0)
uh <- c(4)

x0 = c(1, 1) # setup initial values
sol2 <- solnp(x0, fun = fn2, ineqfun = ineq2, ineqLB = Lh, ineqUB=uh)

##
## Iter: 1 fn: 7.8697      Pars: 0.68437 0.31563
## Iter: 2 fn: 5.6456      Pars: 0.39701 0.03895
## Iter: 3 fn: 5.1604      Pars: 0.200217 0.002001
## Iter: 4 fn: 5.0401      Pars: 0.10011821 0.00005323
## Iter: 5 fn: 5.0100      Pars: 0.0500592618 0.0000006781
## Iter: 6 fn: 5.0025      Pars: 0.02502983706 -0.00000004425
## Iter: 7 fn: 5.0006      Pars: 0.01251500215 -0.00000005034
## Iter: 8 fn: 5.0002      Pars: 0.00625757145 -0.00000005045
## Iter: 9 fn: 5.0000      Pars: 0.00312915970 -0.00000004968
## Iter: 10 fn: 5.0000      Pars: 0.00156561388 -0.00000004983
## Iter: 11 fn: 5.0000      Pars: 0.0007831473 -0.0000000508
## Iter: 12 fn: 5.0000      Pars: 0.00039896484 -0.00000005045
## Iter: 13 fn: 5.0000      Pars: 0.00021282342 -0.00000004897
## Iter: 14 fn: 5.0000      Pars: 0.00014285437 -0.00000004926
## Iter: 15 fn: 5.0000      Pars: 0.00011892066 -0.00000004976
## solnp--> Completed in 15 iterations

```

```

sol2$values
## [1] 19.000000 7.869675 5.645626 5.160388 5.040095 5.010024
5.002506
## [8] 5.000626 5.000157 5.000039 5.000010 5.000002 5.000001
5.000000
## [15] 5.000000 5.000000
sol2$pars
## [1] 1.189207e-04 -4.976052e-08

```

There are a number of parameters that control the `solnp` procedure. For instance, `TOL` defines the tolerance for optimality (which impacts the convergence) and `trace=0` turns off the printing of the results at each iteration.

```

ctrl <- List(TOL=1e-15, trace=0)
sol2 <- solnp(x0, fun = fn2, ineqfun = ineq2, ineqLB = lh, ineqUB=uh, control=ctrl)
sol2$pars
## [1] 1.402813e-08 -5.015532e-08

```

Example 3: More Complex Non-linear Optimization

Let's try to minimize

$$f(X) = -x_1 x_2 x_3$$

subject to

$$4x_1x_2 + 2x_2x_3 + 2x_3x_1 = 100 \\ 1 \leq x_i \leq 10, i = 1, 2, 3.$$

```

fn3 <- function(x, ...){
  -x[1]*x[2]*x[3]
}

eqn3 <- function(x, ...){
  4*x[1]*x[2]+2*x[2]*x[3]+2*x[3]*x[1]
}
constraints3 = c(100)

Lx <- rep(1, 3)
ux <- rep(10, 3)

pars <- c(2, 1, 7) # setup: Try alternative starting-parameter vector (pars)
ctrl <- List(TOL=1e-6, trace=0)
sol3 <- solnp(pars, fun=fn3, eqfun=eqn3, eqB = constraints3, LB=Lx, UB=ux, control=ctrl)
sol2$values

## [1] 19.000000 7.869675 5.645626 5.160388 5.040095 5.010024 5.002506
## [8] 5.000626 5.000157 5.000039 5.000010 5.000002 5.000001 5.000000
## [15] 5.000000 5.000000 5.000000 5.000000 5.000000 5.000000 5.000000
## [22] 5.000000 5.000000 5.000000 5.000000 5.000000 5.000000 5.000000
## [29] 5.000000

sol3$pars
## [1] 2.886751 2.886751 5.773505

```

The non-linear optimization is sensitive to the initial parameters (pars), especially when the objective function is not smooth or if there are many local minima. The function `gosolnp()` may be employed to generate initial (guesstimates of the) parameters.

Example 4: Another Linear Example

Let's try another minimization of a linear objective function $f(x, y, z) = 4y - 2z$ subject to

$$\begin{aligned} 2x - y - z &= 2 \\ x^2 + y^2 &= 1. \end{aligned}$$

```
fn4 <- function(x)  # f(x, y, z) = 4y-2z
{
  4*x[2] - 2*x[3]
}

# constraint z1: 2x-y-z = 2
# constraint z2: x^2+y^2 = 1
eqn4 <- function(x){
  z1=2*x[1] - x[2] - x[3]
  z2=x[1]^2 + x[2]^2

  return(c(z1, z2))
}
constraints4 <- c(2, 1)

x0 <- c(1, 1)
ctrl <- List(trace=0)
sol4 <- solnp(x0, fun = fn4, eqfun = eqn4, eqB = constraints4, control=ctrl)
sol4$values

## [1] 2.000000 -5.078795 -11.416448 -5.764047 -3.584894 -3.224531
## [7] -3.211165 -3.211103 -3.211103

sol4$pars

## [1] 0.55470019 -0.83205030 -0.05854932
```

The materials in the linear algebra and matrix computing, Chap. 5, and the regularized parameter estimation, Chap. 18, provide additional examples of least squares parameter estimation, regression, and regularization.

22.4 Manual Versus Automated Lagrange Multiplier Optimization

Let's manually implement the Lagrange Multipliers procedure and then compare the results to some optimization examples obtained by automatic R function calls. The latter strategies may be more reliable, efficient, flexible, and rigorously validated.

The manual implementation provides a more direct and explicit representation of the actual optimization strategy.

We will test a simple example of an objective function:

$$f(x, y, z) = 4y - 2z + x^2 + y^2,$$

subject to two constraints:

$$\begin{aligned} 2x - y - z &= 2 \\ x^2 + y^2 + z &= 1. \end{aligned}$$

The R package `numDeriv` may be used to calculate numerical approximations of partial derivatives.

```
# define the main Lagrange Multipliers Optimization strategy from scratch
require(numDeriv)

Lagrange_multipliers <- function(x, f, g) { # Objective/cost function,
f, and constraints, g
  k <- length(x)
  l <- length(g(x))

  # Compute the derivatives
  grad_f <- function(x) { grad(f, x) }

  # g, representing multiple constraints, is a vector-valued function:
  # its first derivative is a matrix
  grad_g <- function(x) { jacobian(g, x) }

  # The Lagrangian is a scalar-valued function:
  # L(x, lambda) = f(x) - lambda * g(x)
  # whose first derivative roots give the optimal solutions
  # h(x, lambda) = c( f'(x) - lambda * g'(x), - g(x) ).
  h <- function(y) {
    c(grad_f(y[1:k]) - t(y[-(1:k)])) %*% grad_g(y[1:k]), -g(y[1:k]))
  }

  # To find the roots of the first derivative, we can use Newton's method:
  # iterate y <- y - h'(y)^{-1} h(y) until certain convergence criterion
  # is met # e.g., (\delta <= 1e-6)
  grad_h <- function(y) { jacobian(h, y) }

  y <- c(x, rep(0, l))
  previous <- y + 1
  while(sum(abs(y-previous)) > 1e-6) {
    previous <- y
    y <- y - solve( grad_h(y), h(y) )
  }
  y[1:k]
}

x <- c(0, 0, 0)

# Define the objective cost function
fn4 <- function(x) # f(x, y, z) = 4y-2z + x^2+y^2
```

```

{
  4*x[2] - 2*x[3] + x[1]^2+ x[2]^2
  #sum(x^2)
}
# check the derivative of the objective function
grad(fn4, x)

## [1] 0 4 -2

# define the domain constraints of the problem
# constraint z1: 2x-y-z = 2
# constraint z2: x^2+y^2 +z = 1
eqn4 <- function(x){
  z1=2*x[1] - x[2] - x[3] -2
  z2=x[1]^2 + x[2]^2 + x[3] -1
  return(c(z1, z2))
}

# Check the Jacobian of the constraints
jacobian(eqn4, x)

##      [,1] [,2] [,3]
## [1,]    2   -1   -1
## [2,]    0     0    1

# Call the Lagrange-multipliers solver

# check one step of the algorithm
k <- length(x)
L <- length(eqn4(x));
h <- function(x) {
  c(grad(fn4, x[1:k]) - t(-x[(1:2)])) %*% jacobian(eqn4, x[1:k]),
  -eqn4(x[1:k]))
}
jacobian(h, x)

##      [,1] [,2]      [,3]
## [1,]    4     0  0.000000e+00
## [2,]   -1     2  5.482583e-15
## [3,]   -1     1  0.000000e+00
## [4,]   -2     1  1.000000e+00
## [5,]    0     0 -1.000000e+00

# Lagrange-multipliers solver for f(x, y, z) subject to g(x, y, z)
Lagrange_multipliers(x, fn4, eqn4)
## [1] 0.3416408 -1.0652476 -0.2514708

```

Now, let's double-check the above manual optimization results against the automatic `solnp` solution minimizing

$$f(x, y, z) = 4y - 2z + x^2 + y^2$$

subject to:

$$\begin{aligned} 2x - y - z &= 2 \\ x^2 + y^2 &= 1. \end{aligned}$$

```

library(Rsolnp)
fn4 <- function(x) # f(x, y, z) = 4y-2z + x^2+y^2
{
  4*x[2] - 2*x[3] + x[1]^2+ x[2]^2
}

# constraint z1: 2x-y-z = 2
# constraint z2: x^2+y^2 +z = 1
eqn4 <- function(x){
  z1=2*x[1] - x[2] - x[3]
  z2=x[1]^2 + x[2]^2 + x[3]
  return(c(z1, z2))
}
constraints4 <- c(2, 1)

x0 <- c(1, 1, 1)
ctrl <- list(trace=0)
sol4 <- solnp(x0, fun = fn4, eqfun = eqn4, eqB = constraints4, control=ctrl)
sol4$values

## [1] 4.0000000 -0.1146266 -5.9308852 -3.7035124 -2.5810141 -2.5069444
## [7] -2.5065779 -2.5065778 -2.5065778

```

The results of both (manual and automated) experiments identifying the optimal (x, y, z) coordinates minimizing the objective function $f(x, y, z) = 4y - 2z + x^2 + y^2$ are in agreement.

- *Manual* optimization: `lagrange_multipliers(x, fn4, eqn4): 0.3416408 -1.0652476 -0.2514708.`
- *Automated* optimization: `solnp(x0, fun = fn4, eqfun = eqn4, eqB = constraints4, control = ctrl): 0.3416408 -1.0652476 -0.2514709.`

22.5 Data Denoising

Suppose we are given x_{noisy} with n noise-corrupted data points. The noise may be additive ($x_{noisy} \sim x + \epsilon$) or not additive. We may be interested in denoising the signal and recovering a version of the original (unobserved) dataset x , potentially as a smoothed representation of the original (uncorrupted) process. Smoother signals suggest less (random) fluctuations between neighboring data points.

One objective function we can design to denoise the observed signal, x_{noisy} , may include a *fidelity term* and a *regularization term*; see the regularized linear modeling in Chap. 18.

Total variation denoising assumes that for each time point t , the observed noisy data

$$\underbrace{x_{noisy}(t)}_{\text{observed signal}} \sim \underbrace{x(t)}_{\text{native signal}} + \underbrace{\epsilon(t)}_{\text{random noise}} .$$

To recover the *native signal*, $x(t)$, we can optimize ($\text{argmin}_x f(x)$) the following objective cost function:

$$f(x) = \underbrace{\frac{1}{2} \sum_{t=1}^{n-1} \| y(t) - x_{noisy}(t) \|^2}_{\text{fidelity term}} + \underbrace{\lambda \sum_{t=2}^{n-1} |x(t) - x(t-1)|}_{\text{regularization term}},$$

where λ is the regularization smoothness parameter, $\lambda \rightarrow 0 \Rightarrow y \rightarrow x_{noisy}$. Minimizing $f(x)$ provides a minimum total-variation solution to the data denoising problem.

Below is an example illustrating total variation (TV) denoising using a simulated noisy dataset. We start by generating an oscillatory noisy signal. Then, we compute several smoothed versions of the noisy data, plot the initial and smoothed signals, define and optimize the TV denoising objective function, which is a mixture of a fidelity term and a regularization term (Fig. 22.4).

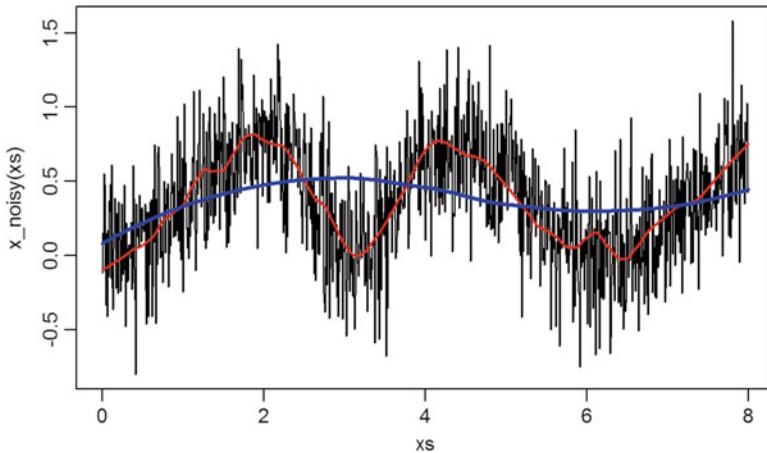


Fig. 22.4 Denoising by smoothing, raw noisy data and two smoothed models (**loess**)

```

n <- 1000
x <- rep(0, n)
xs <- seq(0, 8, len=n) # seq(from = 1, to = 1, length)
noise_level = 0.3 # sigma of the noise, try varying this noise -level

# here is where we add the zero-mean noise
set.seed(1234)

x_noisy <- function (x) {
  sin(x)^2/(1.5+cos(x)) + rnorm(length(x), 0, noise_level)
}

# initialize the manual denoised signal
x_denoisedManu <- rep(0, n)

df <- as.data.frame(cbind(xs, x_noisy(xs)))
# loess fit a polynomial surface determined by numerical predictors,
# using local fitting
poly_model1 <- loess(x_noisy(xs) ~ xs, span=0.1, data=df) # tight model
poly_model2 <- loess(x_noisy(xs) ~ xs, span=0.9, data=df) # smoother model
# To see some of numerical results of the model -fitting:
# View(as.data.frame(cbind(xs, x_noisy, predict (poly_model1))))
```

```

plot(xs, x_noisy(xs), type='l')
Lines(xs, poly_model1$fitted, col="red", lwd=2)
Lines(xs, poly_model2$fitted, col="blue", lwd=3)

```

Next, let's initiate the parameters, define the objective function and optimize it, i.e., estimate the parameters that minimize the cost function as a mixture of fidelity and regularization terms (Fig. 22.5).

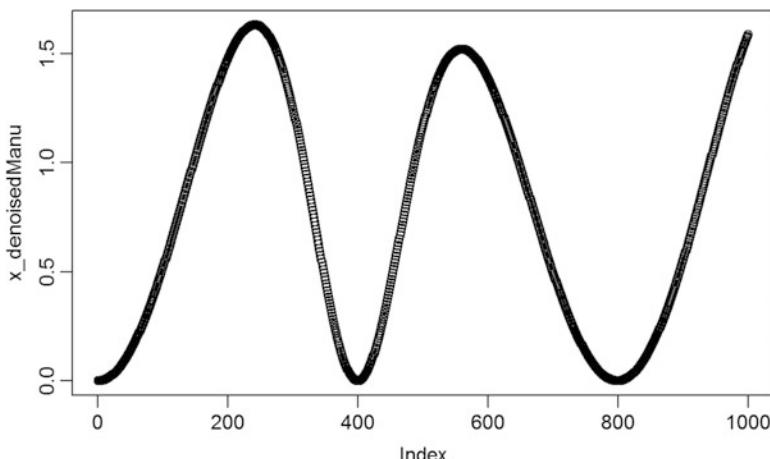


Fig. 22.5 Manual denoising signal recovery using non-linear constraint optimization (**solnp**)

```

# initialization of parameters
betas_0 <- c(0.3, 0.3, 0.5, 1)
betas <- betas_0

# Denoised model
x_denoised <- function(x, betas) {
  if (length(betas) != 4) {
    print(paste0("Error!!! Length(betas)=", length(betas), " != 4!!! Exiting
..."))
    break();
  }
  # print(paste0(".... betas = ", betas, "...\\n"))
  # original noise function definition: sin(x)^2/(1.5+cos(x))
  return((betas[1]*sin(betas[2]*x)^2)/(betas[3]+cos(x)))
}

library(Rsolnp)

fidelity <- function(x, y) {
  sqrt((1/length(x)) * sum((y - x)^2))
}

regularizer <- function(betas) {
  reg <- 0
  for (i in 1:(length(betas-1))) {
    reg <- reg + abs(betas[i])
  }
  return(reg)
}

# Objective Function
objective_func <- function(betas) {
#  $f(x) = 1/2 * \sum_{t=1}^{n-1} \{ |y(t) - x_{noisy}(t)|^2 \} + \lambda * \sum_{t=2}^{n-1} |x(t) - x(t-1)|$ 
  fid <- fidelity(x_noisy(xs), x_denoised(xs, betas))
  reg <- abs(betas[4])*regularizer(betas)
  error <- fid + reg
  # uncomment to track the iterative optimization state
  # print(paste0(".... Fidelity =", fid, "... Regularizer = ", reg, "... TotalError=", error))
  # print(paste0(".... betas=(", betas[1], ",", betas[2], ",", betas[3], ",", betas[4], ")"))
  return(error)
}

# inequality constraint forcing regularization parameter lambda=beta[4]>0
inequalConstr <- function(betas){
  betas[4]
}
inequalLowerBound <- 0; ineqaulUpperBound <- 100

# should we list the value of the objective function and the parameters at
# every iteration (default trace=1; trace=0 means no interim reports)
# constraint problem
# ctrl <- list(trace=0, tol=1e-5) ## could specify: outer.iter=5,
# inner.iter=9)
set.seed(121)

```

```

sol_Lambda <- solnp(betas_0, fun = objective_func, ineqfun = inequalConstr,
ineqLB = unequalLowerBound, ineqUB = unequalUpperBound, control=ctrl)

# unconstraint optimization
# ctrl <- list(trace=1, tol=1e-5) ## could specify: outer.iter=5,
inner.iter=9)
# sol_lambda <- solnp(betas_0, fun = denoising_func, control=ctrl)

# suppress the report of the the functional values (too many)
# sol_lambda$values

# reprot the optimal parameter estimates (betas)
sol_Lambda$pars

## [1] 2.5649689 0.9829681 1.7605481 0.9895268

# Reconstruct the manually-denoised signal using the optimal betas
betas <- sol_Lambda$pars
x_denoisedManu <- x_denoised(xs, betas)
print(paste0("Final Denoised Model:", betas[1], "*sin(", betas[2],
"*x)^2/", betas[3], "+cos(x))))")

## [1] "Final Denoised Model:2.56496893433154*sin(0.982968123322892*x)^2/(1.
76054814253387+cos(x)))"

plot(x_denoisedManu)

```

Finally, we can validate our manual denoising protocol against the automated TV denoising using the R package tvd (Fig. 22.6).

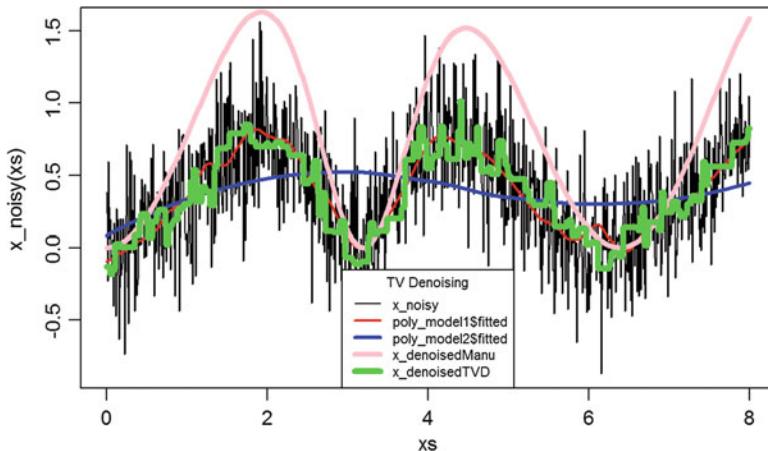


Fig. 22.6 Plot of the observed noisy data and four alternative denoised reconstructions

```

# install.packages("tvd")
library("tvd")

Lambda_0 <- 0.5
x_denoisedTVD <- tvd1d(x_noisy(xs), Lambda_0, method = "Condat")
# lambda_0 is the total variation penalty coefficient
# method is a string indicating the algorithm to use for denoising.
# Default method is "Condat"

# plot(xs, x_denoisedTVD, type='l')
plot(xs, x_noisy(xs), type='l')
Lines(xs, poly_model1$fitted, col="red", lwd=2)
Lines(xs, poly_model2$fitted, col="blue", lwd=3)
Lines(xs, x_denoisedManu, col="pink", lwd=4)
Lines(xs, x_denoisedTVD, col="green", lwd=5)
# add a legend
Legend("bottom", c("x_noisy", "poly_model1$fitted", "poly_model2$fitted",
  "x_denoisedManu", "x_denoisedTVD"), col=c("black", "red", "blue", "pink",
  "green"), lty=c(1,1, 1,1), cex=0.7, lwd= c(1,2,3,4,5), title="TV Denoising")

```

22.6 Assignment: 22. Function Optimization

22.6.1 Unconstrained Optimization

Apply `optim()` to solve the following unconstrained optimization problems:

1. $\min_x f(x) = x^4$.
2. $\max_x \left(2 \sin x - \frac{x^2}{10}\right)$.
3. $\max_{x, y} (2xy + 2x - x^2 - 2y^2)$.

22.6.2 Linear Programming (LP)

Solve the following LP problem:

$$\max_{x_1, x_2, x_3, x_4} (x_1 + 2x_2 + 3x_3 + 4x_4 + 5)$$

subject to:

$$\begin{cases} 4x_1 + 3x_2 + 2x_3 + x_4 & \leq 10 \\ x_1 - x_3 + 2x_4 & = 2 \\ x_1 + x_2 + x_3 + x_4 & \geq 1 \\ x_1 \geq 0, x_3 \geq 0, x_4 & \geq 0 \end{cases}.$$

Apply `lpSolveAPI` and `Rsolnp` and compare the results.

22.6.3 Mixed Integer Linear Programming (MILP)

Apply `lpSolveAPI` to solve the following MILP problem:

$$\min_{x_1, x_2} 4x_1 + 6x_2$$

subject to:

$$\begin{cases} 2x_1 + 2x_2 & \geq 5 \\ x_1 - x_2 & \leq 1 \\ x_1, x_2 & \geq 0 \\ x_1, x_2 & \in \text{integers} \end{cases}.$$

22.6.4 Quadratic Programming (QP)

Solve the following QP problem:

$$\min_{x_1, x_2} 2x_1^2 + x_2^2 + x_1x_2 + x_1 + x_2$$

subject to:

$$\begin{cases} x_1 + x_2 & = 1 \\ x_1, x_2 & \geq 0 \end{cases}.$$

- Apply `quadprog` to solve the QP.
- Use `Rsolnp` to solve the QP.
- Write the Lagrange multiplier form.
- Apply `numDeriv` to solve this Lagrange multiplier optimization manually.
- Compare the three versions of the results above.

22.6.5 Complex Non-linear Optimization

Solve the following nonlinear problem:

$$\min_{x_1, x_2} \left(100(x_2 - x_1^2)^2 + (1 - x_1)^2 \right)$$

subject to $x_1, x_2 \geq 0$.

22.6.6 Data Denoising

Based on the signal denoising example presented in this chapter, try to change the noise level, replicate the denoising process, and report your findings.

References

- Cortez, P. (2014) *Modern Optimization with R*, Springer, ISBN 3319082639, 9783319082639.
CRAN Optimization & Math Programming Site provides details about a broad range of R optimization functions.
Vincent Zoonekynd's Optimization Blog http://zoonek.free.fr/blosxom/R/2012-06-01_Optimization.html.

Chapter 23

Deep Learning, Neural Networks



Deep learning is a special branch of machine learning using a collage of algorithms to model high-level data motifs. Deep learning resembles the biological communications of systems of brain neurons in the central nervous system (CNS), where synthetic graphs represent the CNS network as nodes/states and connections/edges between them. For instance, in a simple synthetic network consisting of a pair of connected nodes, an output sent by one node is received by the other as an input signal. When more nodes are present in the network, they may be arranged in multiple levels (like a multiscale object) where the i th layer output serves as the input of the next $(i + 1)$ st layer. The signal is manipulated at each layer, sent as a layer output downstream, interpreted as an input to the next, $(i + 1)$ st layer, and so forth. Deep learning relies on multipler layers of nodes and many edges linking the nodes forming input/output (I/O) layered grids representing a multiscale processing network. At each layer, linear and non-linear transformations are converting inputs into outputs.

In this chapter, we explore the R deep learning package MXNet and demonstrate state-of-the-art deep learning models utilizing CPU and GPU for fast training (learning) and testing (validation). Other powerful deep learning frameworks include *TensorFlow*, *Theano*, *Caffe*, *Torch*, *CNTK* and *Keras*.

Neural Networks vs. Deep Learning: Deep Learning is a machine learning strategy that learns a deep multi-level hierarchical representation of the affinities and motifs in the dataset. Machine learning Neural Nets tend to use shallower network models. Although there are no formal restrictions on the depth of the layers in a Neural Net, few layers are commonly utilized. Recent methodological, algorithmic, computational, infrastructure, and service advances overcome previous limitations. In addition, the rise of *Big Data* accelerated the evolution of *classical Neural Nets* to *Deep Neural Nets*, which can now handle lots of layers and many hidden nodes per layer. The former is a precursor to the latter; however, there are also *non-neural* deep learning techniques, for example, *syntactic pattern recognition methods* and *grammar induction discover hierarchies*.

23.1 Deep Learning Training

Review Chap. 11 (Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines) prior to proceeding.

23.1.1 Perceptrons

A **perceptron** is an artificial analogue of a neuronal brain cell that calculates a *weighted sum of the input values* and *outputs a thresholded version of that result*. For a bivariate perceptron, P , having two inputs, (X, Y) , we can denote the weights of the inputs by A and B , respectively. Then, the weighted sum could be represented as:

$$W = AX + BY.$$

At each layer l , the weight matrix, $W^{(l)}$, has the following properties:

- The number of rows of $W^{(l)}$ equals the number of nodes/units in the previous $(l - 1)$ st layer, and
- The number of columns of $W^{(l)}$ equals the number of units in the next $(l + 1)$ st layer.

Neuronal cells fire depending on the presynaptic inputs to the cell, which causes constant fluctuations of the neuronal membrane - depolarizing or hyperpolarizing, i.e., the cell membrane potential rises or falls. Similarly, perceptrons rely on thresholding of the weight-averaged input signal, which for biological cells corresponds to voltage increases passing a critical threshold. Perceptrons output non-zero values only when the weighted sum exceeds a certain threshold C . In terms of its input vector, (X, Y) , we can describe the output of each perceptron (P) by:

$$\text{Output}(P) = \begin{cases} 1, & \text{if } AX + BY > C \\ 0, & \text{if } AX + BY \leq C \end{cases}.$$

Feed-forward networks are constructed as layers of perceptrons where the first layer ingests the inputs and the last layer generates the network outputs. The intermediate (internal) layers are not directly connected to the external world, and are called hidden layers. In *fully connected networks*, each perceptron in one layer is connected to every perceptron on the next layer enabling information “fed forward” from one layer to the next. There are no connections between perceptrons in the same layer.

Multilayer perceptrons (fully-connected feed-forward neural network) consist of several fully-connected layers representing an input matrix $X_{n \times m}$ and a generated output matrix $Y_{n \times k}$. The input $X_{n, m}$ is a matrix encoding the n cases and m features per case. The weight matrix $W_{m, k}^{(l)}$ for layer l has rows (i) corresponding to

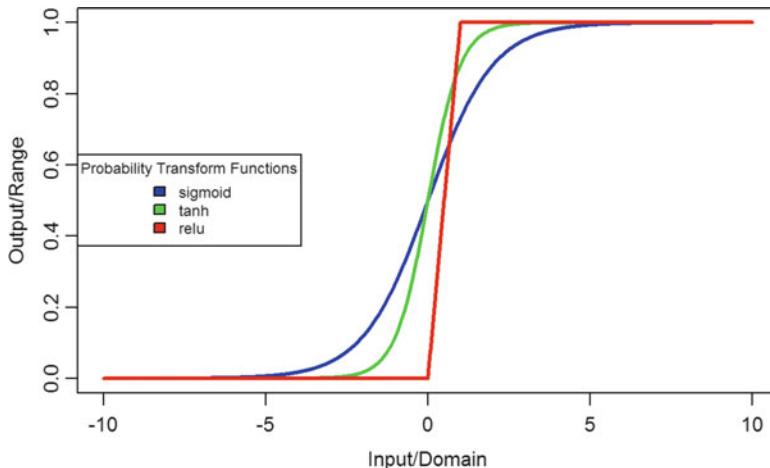


Fig. 23.1 Graphical representation of three alternative activation functions

the weights leading from all the units i in the previous layer to all of the units j in the current layer. The product matrix $X \times W$ has dimensions $n \times k$.

The hidden size parameter k , the weight matrix $W_{m \times k}$, and the bias vector $b_{n \times 1}$ are used to compute the outputs at each layer:

$$Y_{n \times k}^{(l)} = f_k^{(l)}(X_{n \times m} W_{m \times k} + b_{k \times 1}).$$

The role of the bias parameter is similar to the intercept term in linear regression and helps improve the accuracy of prediction by shifting the decision boundary along Y axis. The outputs are fully-connected layers that feed into an activation layer to perform element-wise operations. Examples of **activation functions** that transform real numbers to probability-like values include (Fig. 23.1):

- The sigmoid function, a special case of the logistic function, which converts real numbers to probabilities,
- The rectifier (relu, Rectified Linear Unit) function, which outputs the $\max(0, input)$,
- The tanh (hyperbolic tangent function).

The final fully-connected layer may be hidden of size equal to the number of classes in the dataset and may be followed by a softmax layer mapping the input into a probability score. For example, if a size $n \times m$ input is denoted by $X_{n \times m}$, then the probability scores may be obtained by the softmax transformation function, which maps real valued vectors to vectors of probabilities:

$$\left(\frac{e^{x_{i,1}}}{\sum_{j=1}^m e^{x_{i,j}}}, \dots, \frac{e^{x_{i,m}}}{\sum_{j=1}^m e^{x_{i,j}}} \right).$$

Fig. 23.2 A schematic of a fully-connected feed-forward neural network with two hidden layers

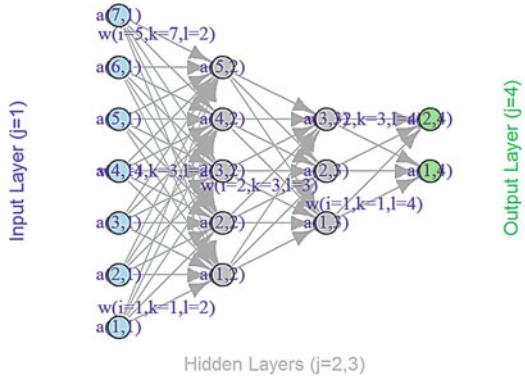


Figure 23.2 shows a schematic of fully-connected feed-forward neural network of nodes:

$$\{a_{j=\text{node index}, l=\text{layer index}}\}_{j=1, l=1}^{n_j, 4}.$$

The plot above illustrates the key elements in the action potential, or activation function, and the calculations of the corresponding training parameters:

$$a_{node=k, layer=l} = f\left(\sum_i w_{k,i}^l \times a_i^{l-1} + b_k^l\right),$$

where:

- f is the *activation function*, e.g., logistic function $f(x) = \frac{1}{1+e^{-x}}$. It converts the aggregate weights at each node to probability values,
- $w_{k,i}^l$ is the weight carried from the i th element of the $(l-1)$ th layer to the k th element of the current l th layer,
- b_k^l is the (residual) bias present in the k th element in the l th layer. This is effectively the information not explained by the training model.

These parameters may be estimated using different techniques (e.g., using least squares, or stochastically using steepest decent methods) based on the training data.

23.2 Biological Relevance

There are parallels between biology (neuronal cells) and the mathematical models (perceptrons) for neural network representation. The human brain contains about 10^{11} neuronal cells connected by approximately 10^{15} synapses forming the basis of our

functional phenotypes. Figure 23.3 illustrates some of the parallels between brain biology and the mathematical representation using synthetic neural nets. Every neuronal cell receives multi-channel (afferent) input from its dendrites, generates output signals, and disseminates the results via its (efferent) axonal connections and synaptic connections to dendrites of other neurons.

The perceptron is a mathematical model of a neuronal cell that allows us to explicitly determine algorithmic and computational protocols transforming input signals into output actions. For instance, a signal arriving through an axon x_0 is modulated by some prior weight, e.g., synaptic strength, $w_0 \times x_0$. Internally, within the neuronal cell, this input is aggregated (summed, or weight-averaged) with inputs from all other axons. Brain plasticity suggests that synaptic strengths (weight coefficients w) are strengthened by training and prior experience. This learning process controls the direction and strength of influence of neurons on other neurons. Either excitatory ($w > 0$) or inhibitory ($w \leq 0$) influences are possible. Dendrites and axons carry signals to and from neurons, where the aggregate responses are computed and transmitted downstream. Neuronal cells only fire if action potentials exceed a certain threshold. In this situation, a signal is transmitted downstream through its axons. The neuron remains silent, if the summed signal is below the critical threshold.

Timing of events is important in biological networks. In the computational perceptron model, a first order approximation may ignore the timing of neuronal firing (spike events) and only focus on the frequency of the firing. The firing rate of a neuron with an activation function f represents the frequency of the spikes along the axon. We saw some examples of activation functions earlier.

Figure 23.3 illustrates the parallels between the brain network-synaptic organization and an artificial synthetic neural network.

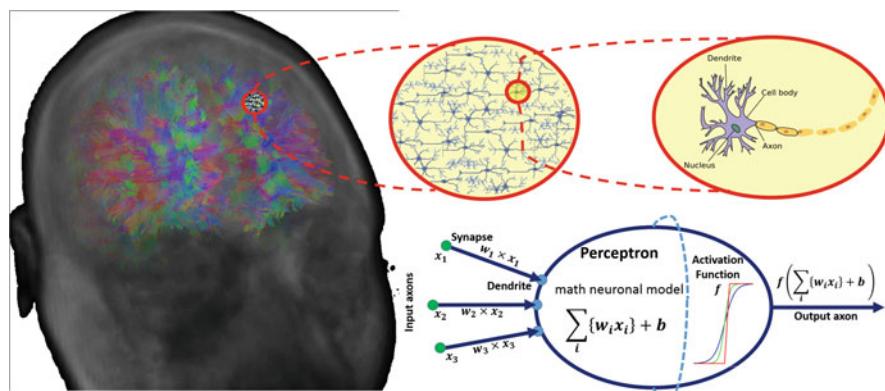


Fig. 23.3 A depiction of the parallels between a biological central nervous system network organization (human brain) and a synthetic neural network employed in deep machine learning

23.3 Simple Neural Net Examples

Before we look at some examples of deep learning algorithms applied to model observed natural phenomena. Specifically, we will develop a couple of simple networks for computing fundamental Boolean operations.

23.3.1 Exclusive OR (XOR) Operator

The exclusive OR (XOR) operator works as a bivariate binary-outcome function, mapping pairs of false (0) and true (1) values to dichotomous false (0) and true (1) outcomes.

We can design a simple two-layer neural network that calculates XOR. The **values listed within each neuron represent its explicit threshold**, which can be normalized so that all neurons utilize the same threshold (typically 1). The **value labels associated with network connections (edges) represent the weights of the inputs**. When the threshold is not reached, the output is 0, and when the threshold is reached, the output is correspondingly 1 (Fig. 23.4).

Let's work out manually the four possibilities (Table 23.1):

We can validate that this network indeed represents an XOR operator by plugging in all four possible input combinations and confirming the expected results at the end (Fig. 23.5).

Fig. 23.4 A neural network representation corresponding to the XOR binary function

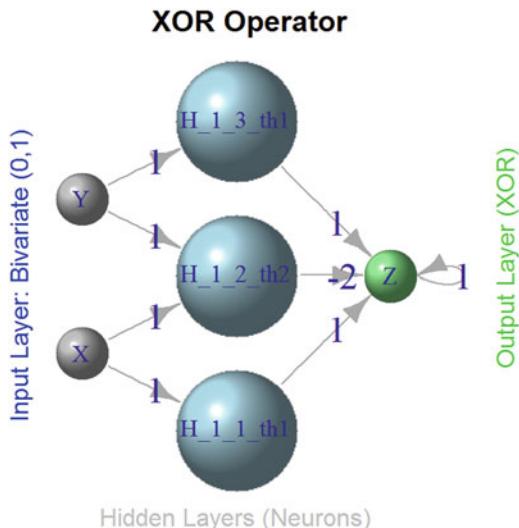


Table 23.1 Exact XOR binary operator

InputX	InputY	XOR output(Z)
0	0	0
0	1	1
1	0	1
1	1	0

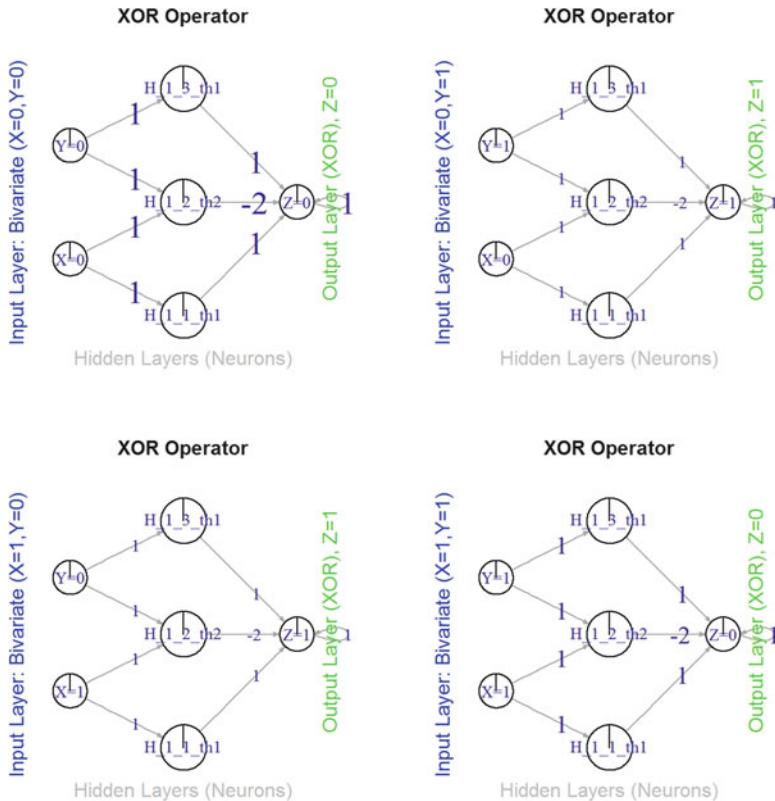


Fig. 23.5 Validation of the explicit neural network calculation of the XOR operator

23.3.2 NAND Operator

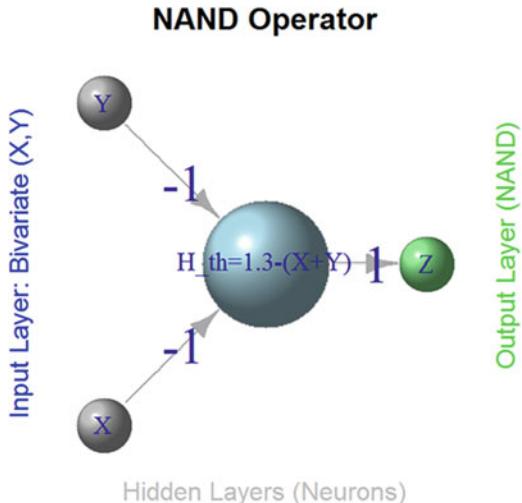
Another binary operator is NAND (negative AND, Sheffer stroke), which produces a false (0) output if and only if both of its operands are true (1), and which generates true (1), otherwise. Below is the NAND input-output table (Table 23.2).

Similarly to the XOR operator, we can design a one-layer neural network that calculates NAND. Again, the **values listed within each neuron represent its explicit threshold**, which can be normalized so that all neurons utilize the same threshold (typically 1). The **value labels associated with network connections (edges) represent the weights of the inputs**. When the threshold is not reached, the output

Table 23.2 Exact NAND binary operator

InputX	InputY	NAND output(Z)
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 23.6 A neural network representation corresponding to the NAND binary function



is trivial (0), and when the threshold is reached, the output is correspondingly 1. Here is a shorthand analytic expression for the NAND calculation:

$$NAND(X, Y) = 1.3 - (1 \times X + 1 \times Y).$$

Check that $NAND(X, Y) = 0$ if and only if $X = 1$ and $Y = 1$, otherwise it equals 1 (Fig. 23.6).

23.3.3 Complex Networks Designed Using Simple Building Blocks

Observe that stringing together some of these primitive networks of Boolean operators, or/and increasing the number of hidden layers, allows us to model problems with exponentially increasing complexity. For instance, constructing a 4-input NAND function would simply require repeating several of our 2-input NAND operators. This will increase the space of possible outcomes from 2^2 to 2^4 . Of course, introducing more depth in the **hidden layers** further expands the complexity of the problems that can be modeled using neural nets.

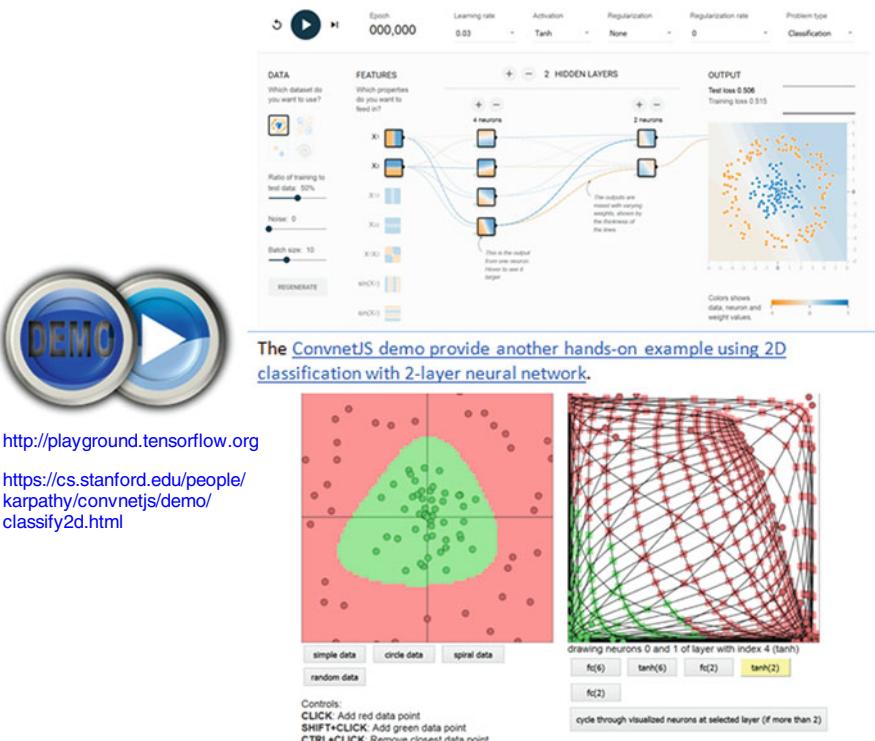


Fig. 23.7 Live Demo: TensorFlow and ConvnetJS deep neural network webapps

You can interactively manipulate Google's TensorFlow Deep Neural Network Webapp to gain additional intuition and experience with the various components of deep learning networks.

The ConvnetJS demo provide another hands-on example using 2D classification with 2-layer neural network (Fig. 23.7).

23.4 Classification

In MXNet, a Multilayer perceptron (MLP) may be defined by:

- Creating a place holder variable for the input data, `data = mx.sym.Variable('data')`
- Flattening the data from 4D shape space (width, height, batch_size, num_channel) into 2D (num_channel*width*height, batch_size), `'data = mx.sym.Flatten(data=data)'`
- And iterating over the fully-connected layers:

- First layer, `fc1 = mx.sym.FullyConnected(data=data, name='fc1', num_hidden=128)`
- Apply `relu` function to the output of the first fully-connected layer, `act1 = mx.sym.Activation(data=fc1, name='relu1', act_type="relu")`
- Generate the second fully-connected layer and apply the activation function, `fc2 = mx.sym.FullyConnected(data=act1, name='fc2', num_hidden = 64); act2 = mx.sym.Activation(data=fc2, name='relu2', act_type="relu")`
- Generate the third/final fully-connected layer, with a hidden size $k = 10$, which in digit recognition tasks corresponds to the number of unique digits $0 : 9$, `fc3 = mx.sym.FullyConnected(data=act2, name='fc3', num_hidden=10)`
- Finally, mapping the input into a probability score using the softmax and loss layer, `mlp = mx.sym.SoftmaxOutput(data=fc3, name='softmax')`. See the `mlp` R source code here.

23.4.1 Sonar Data Example

Let's load the `mlbench` and `mlbench` packages and demonstrate the basic invocation of `mxnet`. The Sonar data `mlbench::Sonar` includes sonar signals bouncing off a metal cylinder or a roughly cylindrical rock. Each of 208 patterns includes a set of 60 numbers (features) in the range 0.0–1.0, and a label M (metal) or R (rock). Each feature represents the energy within a particular frequency band, integrated over a certain period of time. The M and R labels associated with each observation classify the record as rock or mine (metal) cylinder. The numbers in the labels are in increasing order of aspect angle, but they do not encode the angle directly.

```
# Load the required packages: mlbench and mxnet
# install.packages("mlbench"); install.packages("mxnet")
# Note mxnet requires "visNetwork"
# If it doesn't work, you may need the following lines:
# install.packages("drat", repos="https://cran.rstudio.com")
# drat:::addRepo("dmlc")
# install.packages("mxnet")

require(mlbench)
require(mxnet)

## Init Rcpp

data(Sonar, package="mlbench")

table(Sonar[, 61])
```

```

## 
##   M   R
## 111  97

Sonar[,61] = as.numeric(Sonar[,61])-1 # R = "1", "M" = "0"
set.seed(123)
train.ind = sample(1:nrow(Sonar), 0.7*nrow(Sonar))

train.x = data.matrix(Sonar[train.ind, 1:60])
train.y = Sonar[train.ind, 61]
test.x = data.matrix(Sonar[-train.ind, 1:60])
test.y = Sonar[-train.ind, 61]

```

Let's start by using a **multi-layer perceptron** as a classifier. The `mxnet` function `mx.mlp` builds a general multi-layer neural network that can be utilized to do classification or regression graph modeling. It relies on the following parameters:

- Training data and labels
- Number of hidden nodes in each hidden layers
- Number of nodes in the output layer
- Type of activation
- Type of output loss
- The device to train (GPU or CPU)
- Additional optional parameters, see `mx.model.FeedForward.create`

Here is one example using the *training* and *testing* data we defined above:

```

mx.set.seed(1234)
model.mx <- mx.mlp(train.x, train.y, hidden_node=8, out_node=2,
out_activation="softmax",
  num.round=200, array.batch.size=15, Learning.rate=0.1, momentum=0.9,
  eval.metric=mx.metric.accuracy, verbose=F)
#calculate Prediction sensitivity & specificity
preds = predict(model.mx, test.x) # these are probabilities

# You can inspect the test labels vs. assigned probabilities by
# View(data.frame(test.y, preds[2,]))
preds1 <- ifelse(preds[2,] <= 0.5, 0, 1) # dichotomize to labels
table(preds1)

## preds1
##   0   1
## 35 28

pred.Label = t(preds1)
table(pred.Label, test.y)

##           test.y
## pred.Label 0 1
##             0 28 7
##             1  6 22

library("caret")
sensitivity(factor(preds1), factor(as.numeric(test.y)),positive = 1)
## [1] 0.7586207
specificity(factor(preds1), factor(as.numeric(test.y)),negative = 0)
## [1] 0.8235294

```

We can also use `crossval::diagnosticErrors()` and `crossval::confusionMatrix()` to get more detailed evaluations. Similar to using the `sensitivity()` and `specificity()` methods, we should specify the *negative* and *positive* labels.

Note that you have to specify `crossval::confusionMatrix()` if you also have the `caret` package loaded, as `caret` also has a function called `confusionMatrix()`.

```
library("crossval")
diagnosticErrors(crossval::confusionMatrix(preds1, test.y, negative = 0))
##      acc      sens      spec      ppv      npv      Lor
## 0.7936508 0.7857143 0.8000000 0.7586207 0.8235294 2.6855773
## attr(", "negative")
## [1] 0
```

Now, we compare the results of different number of rounds, or epochs, representing the number of full (training-phase) passes through the data (cf. num. `round=n`).

```
mx.set.seed(1234)
get_pred = function(n){
  model.mx <- mx.mlp(train.x, train.y, hidden_node=8, out_node=2, out_activation="softmax",
    num.round=n, array.batch.size=15, learning.rate=0.1, momentum=0.9,
    eval.metric=mx.metric.accuracy, verbose=F)
  preds = predict(model.mx, test.x)
}
preds100 = get_pred(100)
preds50 = get_pred(50)
preds10 = get_pred(10)
```

We can plot the ROC curve and calculate the AUC (Area under the curve) (Fig. 23.8):

```
# install.packages("pROC"); install.packages("plotROC"); install.packages("reshape2")
library(pROC); library(plotROC); library(reshape2);

# compute AUC
get_roc = function(preds){
  roc_obj <- roc(test.y, preds[2,])
  auc(roc_obj)
}
get_roc(preds)

## Area under the curve: 0.9249

get_roc(preds100)
```

```

## Area under the curve: 0.9209
get_roc(preds50)

## Area under the curve: 0.8824
get_roc(preds10)

## Area under the curve: 0.8022

#plot roc
dt <- data.frame(test.y, preds[2,], preds100[2,], preds50[2,], preds10[2,])
colnames(dt) <- c("D", "rounds=200", "rounds=100", "rounds=50", "rounds=10")
dt <- melt(dt, id.vars = "D")

basicplot <- ggplot(dt, aes(d = D, m = value, colour=variable)) +
  geom_roc(labels = FALSE, size = 0.5, alpha.line = 0.6, linejoin = "mitre") +
  theme_bw() + coord_fixed(ratio = 1) + style_roc() + ggtitle("ROC CURVE") +
  annotate("rect", xmin = 0.4, xmax = 1, ymin = 0.2, ymax = 0.75,
           alpha = .2) +
  annotate("text", x = 0.7, y = 0.5, size = 3,
           label = "AUC: \n rounds=200: 0.9209\n rounds=100: 0.9128\n
rounds=50: 0.8824\n rounds=10: 0.8022\n ")
basicplot

```

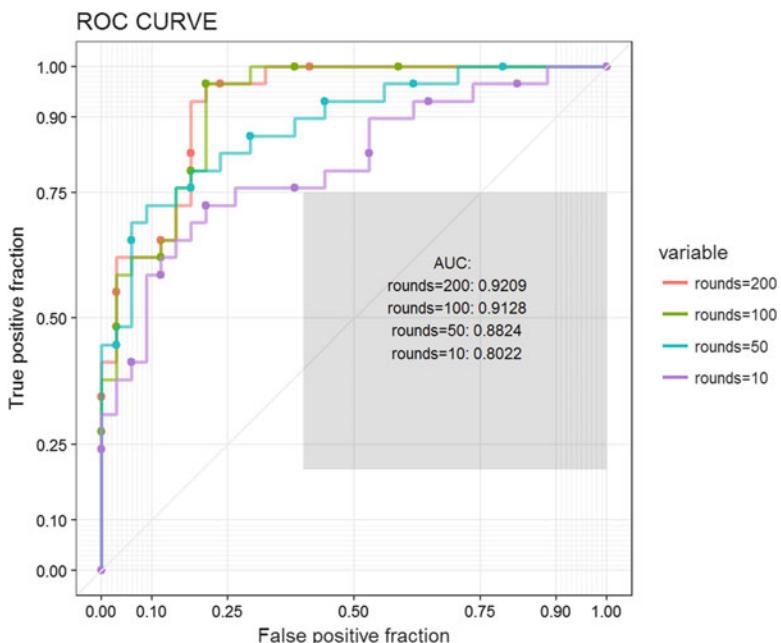


Fig. 23.8 ROC curves of multi-layer perceptron predictions (`mx.mlp`), using out-of-bag test-data, corresponding to different number of iterations, see Chap. 14

The plot suggests that the results stabilize after 100 training (epoch) iterations. Let's look at some visualizations of the real labels of the test data (`ttest.y`) and their corresponding ML-derived classification labels (`preds[2,]`) using 200 iterations (Figs. 23.9, 23.10, 23.11, 23.12, and 23.13).

```
graph.viz(model.mx$symbol)
hist(preds10[2,],main = "rounds=10")
hist(preds50[2,],main = "rounds=50")
hist(preds100[2,],main = "rounds=100")
hist(preds[2,],main = "rounds=200")
```

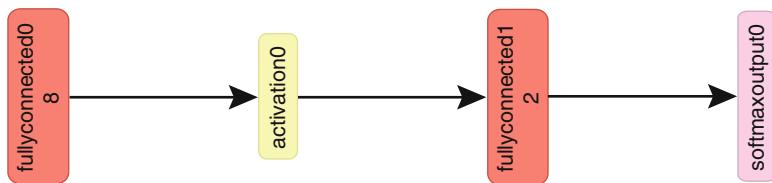


Fig. 23.9 MLP model structure (the plot is rotated 90-degrees to save space)

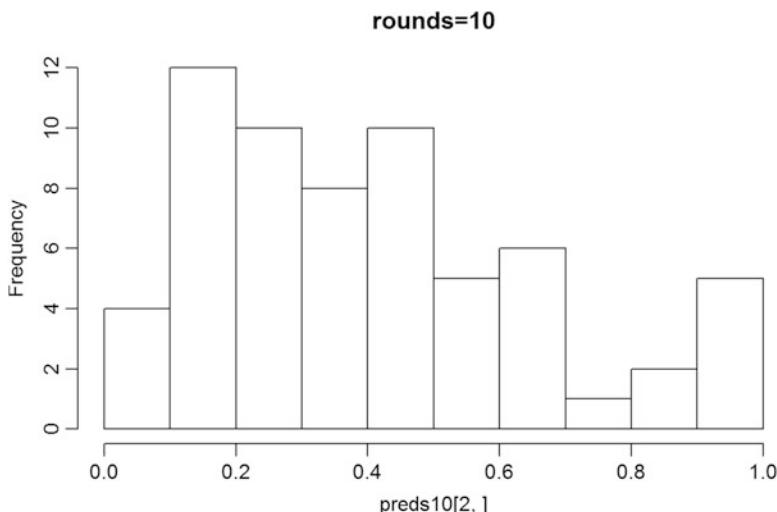


Fig. 23.10 Frequency plot of the predicted probabilities using ten epochs corresponding to ten full (training-phase) passes through the data (cf. `num.round=n`)

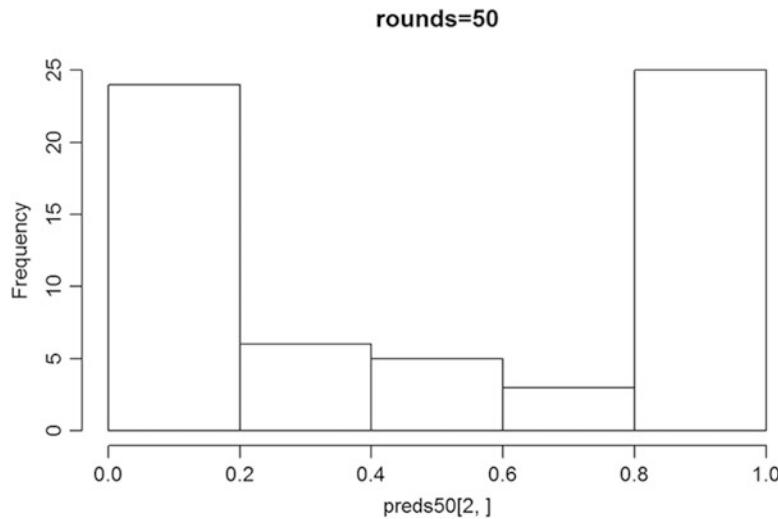


Fig. 23.11 Frequency plot of the predicted probabilities using 50 epochs, compare to Fig. 23.10

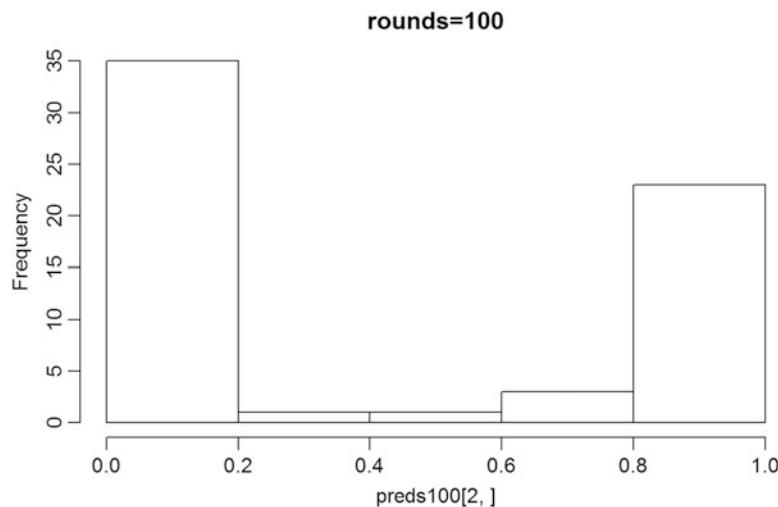


Fig. 23.12 Frequency plot of the predicted probabilities using 100 epochs, compare to Fig. 23.11

We see a significant bimodal trend when the number of rounds increases. Another plot shows more details about the agreement between the real labels and their predicted class counterparts (Fig. 23.14):

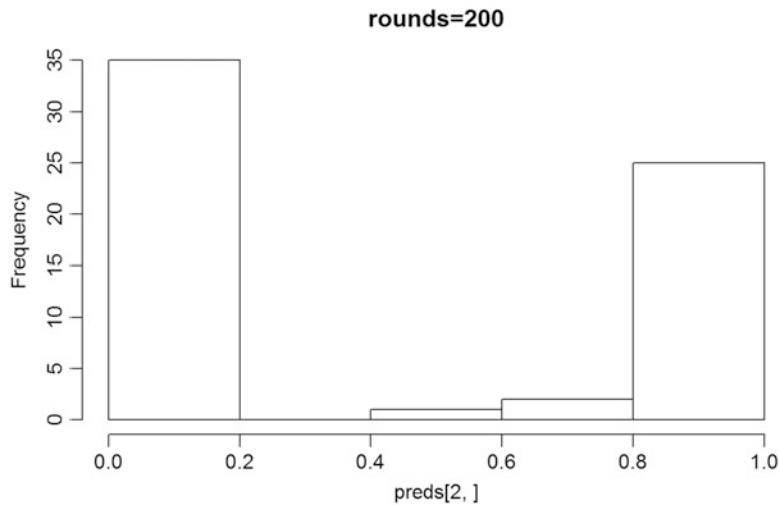


Fig. 23.13 And finally, the plot of the predicted probabilities using 200 epochs; compare to Fig. 23.12

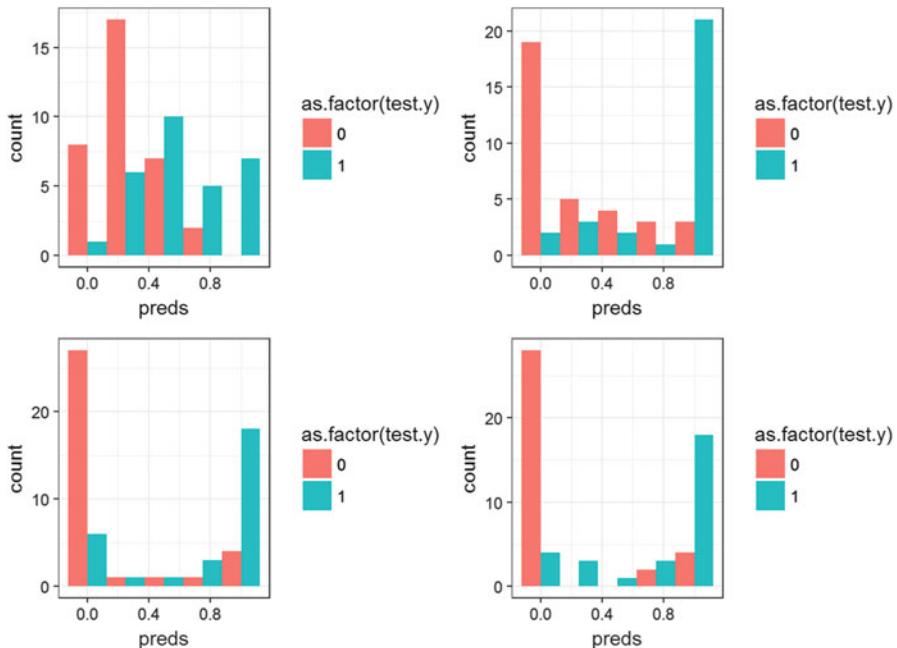


Fig. 23.14 Summary plots illustrating the progression of the neural network learning from 10 to 200 epochs, corresponding with improved binary classification results (testing data)

```

library(ggplot2)
get_gghist = function(preds){
  ggplot(data.frame(test.y, preds),
         aes(x=preds, group=test.y, fill=as.factor(test.y)))+
    geom_histogram(position="dodge", binwidth=0.25)+theme_bw()
}
df = data.frame(preds[2,],preds100[2,],preds50[2,],preds10[2,])
p <- lapply(df, get_gghist)
require(gridExtra) # used for arrange ggplots
grid.arrange(p$preds10.2..., p$preds50.2..., p$preds100.2..., p$preds.2...)

```

23.4.2 MXNet Notes

- The `mx.mlp()` function is a proxy to the more complex and laborious process of defining a neural network by using MXNet's `Symbol`. For instance, this call `model.mx <- mx.mlp(train.x, train.y, hidden_node=8, out_node=2, out_activation="softmax", num.round=20, array.batch.size=15, learning.rate=0.1, momentum=0.9, eval.metric=mx.metric.accuracy)` would be equivalent to a symbolic network definition like: `data <- mx.symbol.Variable ("data"); fc1 <- mx.symbol.FullyConnected(data, num_hidden=128) act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu"); fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64); act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu"); fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=2); lro <- mx.symbol.SoftmaxOutput(fc3, name="sm"); model2 <- mx.model.FeedForward.create(lro, X=train.x, y=train.y, ctx=mx.cpu(), num.round=100, array.batch.size=15, learning.rate=0.07, momentum=0.9)` (see example with linear regression below).
- Layer-by-layer definitions translate inputs into outputs. At each level, the network allows for a different number of neurons and alternative activation functions. Other options can be specified by using `mx.symbol`:
- `mx.symbol.Convolution` applies convolution to the input and then adds a bias. It can create convolutional neural networks.
- `mx.symbol.Deconvolution` does the opposite and can be used in segmentation networks along with `mx.symbol.UpSampling`, e.g., to reconstruct the pixel-wise classification of an image.
- `mx.symbol.Pooling` reduces the data by selecting signals with the highest response.
- `mx.symbol.Flatten` links convolutional and pooling layers to form a fully connected network.
- `mx.symbol.Dropout` attempts to cope with the overfitting problem.

The function `mx.mlp()` is a wrapper for quick design of standard multi-layer perceptrons. For more extensive experiments, customized symbolic representation can be explicitly specified using combinations of the above methods.

```
# Example of **LeNet** network for recognizing handwritten digits:
data <- mx.symbol.Variable('data')
conv1 <- mx.symbol.Convolution(data=data, kernel=c(5,5), num_filter=20)
tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2,2),
    stride=c(2,2))
conv2 <- mx.symbol.Convolution(data=pool1, kernel=c(5,5), num_filter=50)
tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2,2),
    stride=c(2,2))
flatten <- mx.symbol.Flatten(data=pool2)
fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
fc2 <- mx.symbol.FullyConnected(data=tanh3, num_hidden=10)
lenet <- mx.symbol.SoftmaxOutput(data=fc2)
model <- mx.model.FeedForward.create(lenet, X=train.x, y=train.y, ctx=device
.cpu, num.round=5, array.batch.size=100, Learning.rate=0.05, momentum=0.9)
```

To allow smooth, fast, and consistent operation on CPU and GPU, in `in` `mxnet`, the generic R function controlling the reproducibility of stochastic results is overwritten by `mx.set.seed`. So can use `mx.set.seed()` to control random numbers in **MXNet**.

To examine the accuracy of the `model.mx` learner (trained on the training data), we can make prediction (on testing data) and evaluate the results using the provided testing labels (report the confusion matrix).

```
preds = predict(model.mx, test.x)
pred.Label = max.col(t(preds))-1
table(pred.Label, test.y)

##           test.y
## pred.Label  0  1
##             0 28  7
##             1  6 22
```

For multi-class predictions, `mxnet` outputs n (*class*) \times m (*examples*) confusion matrices where each row corresponds to probability of the corresponding (column-defined) class.

23.5 Case-Studies

Let's demonstrate *regression* and *prediction* deep learning examples using several complementary datasets.

23.5.1 ALS Regression Example

Let's first demonstrate a deep learning regression using the ALS data to predict ALSFRS_slope, Figs. 23.15 and 23.16.

```
als <- read.csv("https://umich.instructure.com/files/1789624/download?downlo
ad_frd=1")
als <- scale(als[,-c(1,7)])
train.ind = sample(1:nrow(als), 0.7*nrow(als))
train.x = data.matrix(als[train.ind,-c(1,7)])
train.y = als[train.ind,7]
test.x = data.matrix(scale(als[-train.ind,-c(1,7)]))
test.y = als[-train.ind,7]

# Define the input data
data <- mx.symbol.Variable("data")
# A fully connected hidden layer
# data: input source
# num_hidden: number of neurons in this hidden layer
fc1 <- mx.symbol.FullyConnected(data, num_hidden=1)
# Use linear regression for the output layer
lro <- mx.symbol.LinearRegressionOutput(fc1)

mx.set.seed(1234)
# Create a MXNet Feedforward neural net model with the specified training.
model <- mx.model.FeedForward.create(lro, X=train.x, y=train.y,
  ctx=mx.cpu(), num.round=1000, array.batch.size=20,
  Learning.rate=2e-6, momentum=0.9, eval.metric=mx.metric.rmse, verbose=F)
```

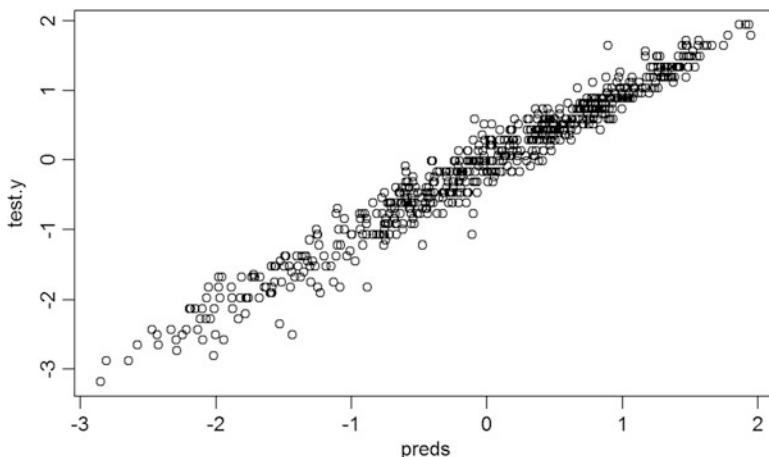
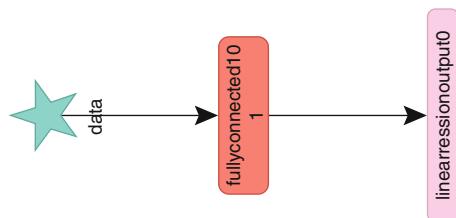


Fig. 23.15 The strong linear relation between the out-of-bag testing data continuous outcome variable (y-axis) and the corresponding predicted regression values (x-axis) suggests a good network prediction performance

Fig. 23.16 Computational graph of the neural network



The option `verbose = F` can suppress messages, including training accuracy reports, in each iteration. You may rerun the code with `verbose = T` to examine the rate of decrease of *train error* against the *number of iterations*.

You must `scale` data before inputting it into MXnet, which expects that the training and testing sets are normalized to the **same scale**. There are two strategies to scale the data.

- Either scaling the complete data simultaneously and then splitting them into train data and test data, or
- Alternatively, scaling only the training dataset to enable model-training, but saving your protocol for data normalization, as new data (testing, validation) will need to be (pre)processed the same way as the training data.

Have a look at the Google TensorFlow API. It shows the importance of *learning rate* and the *number of rounds*. You should test different sets of parameters.

- Too small *learning rate* may lead to long computations.
- Too large *learning rate* may cause the algorithm to fail to converge, as large step size (learning rate) may by-pass the optimal solution and then oscillate or even diverge.

```

preds = predict(model, test.x)
sqrt(mean((preds-test.y)^2))
## [1] 0.2171032
range(test.y)
## [1] -3.181499  1.943890
# plot the correlation between testdata.y and testdata.predicted.y
plot(preds, test.y)
  
```

We can see that the *RMSE* on the test set is pretty small. To get a visual representation of the deep learning network we can also display this relatively simple computation graph (Fig. 23.16):

```
graph.viz(model$symbol)
```

23.5.2 *Spirals 2D Data*

We can again use the `mx.mlp` wrapper to construct the learning network, but we can also use a more flexible way to construct and configure the multi-layer network in `mxnet`. This configuration is done by using the `Symbol` call, which specifies the links among network nodes, the activation function, dropout ratio, and so on:

Below we show the configuration of a perceptron with one hidden layer.

```
##### Network configuration
# variables
act <- mx.symbol.Variable("data")
# affine transformation
fc <- mx.symbol.FullyConnected(act, num.hidden = 10)
# non-linear activation
act <- mx.symbol.Activation(data = fc, act_type = "relu")
# affine transformation
fc <- mx.symbol.FullyConnected(act, num.hidden = 2)
# softmax output and cross-
mlp <- mx.symbol.SoftmaxOutput(fc)

#####Preparing data
set.seed(2235)

##### spirals dataset
s <- sample(x = c("train", "test"), size = 1000, prob = c(.8,.2), replace = TRUE)
dta <- mlbench.spirals(n = 1000, cycles = 1.2, sd = .03)
dta <- cbind(dta[["x"]], as.integer(dta[["classes"]]) - 1)
colnames(dta) <- c("x", "y", "Label")
##### train, validate, test
dta.train <- dta[s == "train",]
dta.test <- dta[s == "test",]
```

Let's display the data and examine its structure (Fig. 23.17).

```
dt <- as.data.frame(dta);dt[,3] <- as.factor(dt[,3])
dt.train <- dt[s == "train",]
dt.test <- dt[s == "test",]
p1 <- ggplot(dt,aes(x = x,y = y,color=Label))+geom_point()+ggtitle("Whole
data structure")
p2 <- ggplot(dt.train,aes(x = x,y =
y,color=Label))+geom_point()+ggtitle("Train data structure")
p3 <- ggplot(dt.test,aes(x = x,y =
y,color=Label))+geom_point()+ggtitle("Test data structure")
grid.arrange(p1,p2,p3,nrow=3)
```

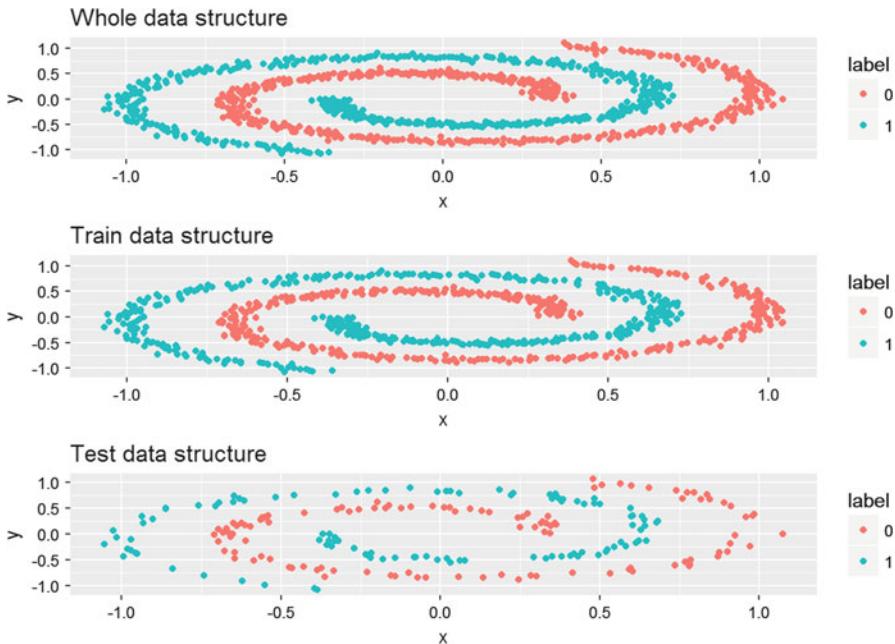


Fig. 23.17 Original spirals data structure (whole, training and testing sets)

```

# Network training
# Feed-forward networks may be trained using iterative gradient descent algorithms. A **batch** is a subset of data that is used during single forward pass of the algorithm. An **epoch** represents one step of the iterative process that is repeated until all training examples are used.

##### basic spiral-data training
mx.set.seed(2235)
model <- mx.model.FeedForward.create(
  symbol = mlp,
  X = dta.train[, c("x", "y")],
  y = dta.train[, c("Label")],
  num.round = 500,
  array.layout = "rowmajor",
  learning.rate = 1,
  eval.metric = mx.metric.accuracy, verbose = F)

preds = predict(model, dta.test[,c(1:2)])
pred.Label = max.col(t(preds))-1; table(pred.Label, dta.test[,3])

## pred.Label  0  1
##          0 90 30
##          1 22 73

```

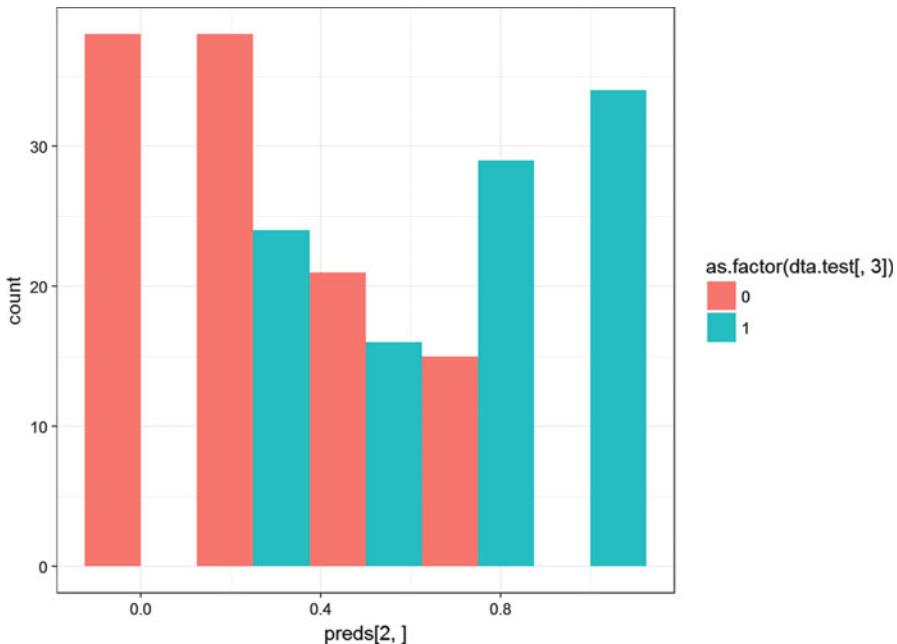


Fig. 23.18 Frequency of feed-forward neural network prediction probabilities (x-axis) for the spirals data relative to testing set labels (colors)

The prediction result is close to perfect, and we can inspect deeper the results using `crossval::confusionMatrix` (Fig. 23.18).

```
library("crossval")
diagnosticErrors(crossval::confusionMatrix(pred.Label, dta.test[, 3], negative = 0))
##      acc      sens      spec      ppv      npv      lor
## 0.7581395 0.7684211 0.7500000 0.7087379 0.8035714 2.2980293
## attr(", "negative")
## [1] 0

ggplot(data.frame(dta.test[, 3], preds[2, ]), 
       aes(x=preds[2, ], group=dta.test[, 3], fill=as.factor(dta.test[, 3])))+
  geom_histogram(position="dodge", binwidth=0.25)+theme_bw()
```

Once we fit a model (like the binary label classification below), we can:

- Visually inspect the quality of the ML classification.
- Display the structure of the labeled test-data objects (Fig. 23.19).

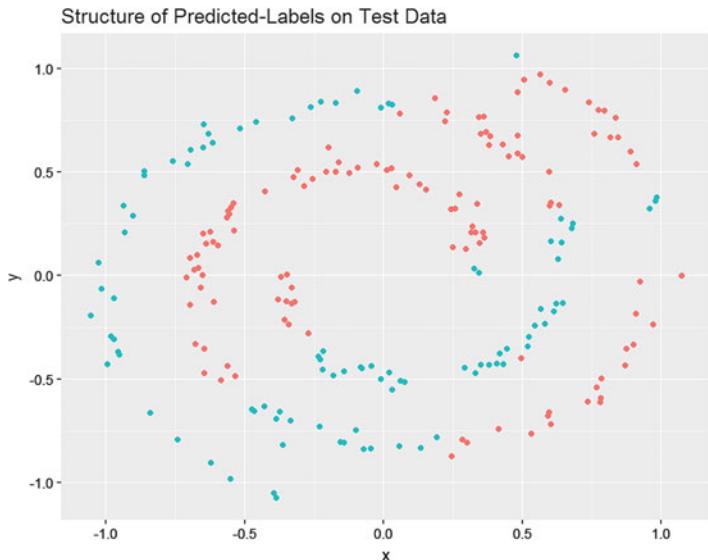


Fig. 23.19 Testing-data validation of neural network model (spirals)

```
# define a custom call-back, which stops the process of training when the pr
ogress in accuracy is below certain level of tolerance. It call is made afte
r every epoch to check the status of convergence of the algorithm.

mx.callback.train.stop <- function(tol = 1e-3, mean.n = 1e2, period = 100, m
in.iter = 100) {
  function(iteration, nbatch, env, verbose = TRUE) {
    if (nbatch == 0 & !is.null(env$metric)) {
      continue <- TRUE
      acc.train <- env$metric$get(env$train.metric)$value
      if (is.null(env$acc.Log)) {
        env$acc.Log <- acc.train
      } else {
        if ((abs(acc.train - mean(tail(env$acc.Log, mean.n))) < tol &
            abs(acc.train - max(env$acc.Log)) < tol &
            iteration > min.iter) |
            acc.train == 1) {
          cat("Training finished with final accuracy: ",
              round(acc.train * 100, 2), "%\n", sep = "")
          continue <- FALSE
        }
        env$acc.Log <- c(env$acc.Log, acc.train)
      }
    }
    if (iteration %% period == 0) {
      cat("[", iteration, "]", " training accuracy: ",
          round(acc.train * 100, 2), "%\n", sep = "")
    }
  return(continue)
}
```

```
##### training with custom stopping
mx.set.seed(2235)
model <- mx.model.FeedForward.create(
  symbol = mlp,
  X = dta.train[, c("x", "y")],
  y = dta.train[, c("label")],
  num.round = 2000,
  array.layout = "rowmajor",
  Learning.rate = 1,
  epoch.end.callback = mx.callback.train.stop(),
  eval.metric = mx.metric.accuracy,
  verbose = FALSE
)

## [100] training accuracy: 75.56 %
## [200] training accuracy: 76 %
## [300] training accuracy: 76 %
## [400] training accuracy: 76.45 %
## Training finished with final accuracy: 76.45 %

Labeled_spiral_data <- as.data.frame(cbind(dta.test[, c("x", "y")],
as.factor(pred.label)))
colnames(Labeled_spiral_data) <- c("x", "y", "label")
Labeled_spiral_data$label <- as.factor(Labeled_spiral_data$label)
p4 <- ggplot(Labeled_spiral_data, aes(x = x, y = y, color = label)) +
  geom_point() + ggtitle("Structure of Predicted-Labels on Test Data")
p4
```

23.5.3 IBS Study

Let's try another example using the IBS Neuroimaging study (Figs. 23.20 and 23.21).

```
# IBS NI Data
# UCLA Data
wiki_url <-
read_html("http://wiki.stat.ucla.edu/socr/index.php/SOCR_Data_April2011_NI_IBS_Pain")
IBSData <- html_table(html_nodes(wiki_url, "table"))[[2]] # table 2
set.seed(1234)
test.ind = sample(1:354, 50, replace = F) # select 50/354 of cases for
testing, train on remaining (354-50)/354 cases

# UMich Data (includes MISSING data): use `mice` to impute missing data
with mean: newData <- mice(data, m=5, maxit=50, meth='pmm', seed=500);
summary(newData)
# wiki_url <-
read_html("http://wiki.socr.umich.edu/index.php/SOCR_Data_April2011_NI_IBS_Pain")
# IBSData <- html_table(html_nodes(wiki_url, "table"))[[1]] # load Table 1
# set.seed(1234)
# test.ind = sample(1:337, 50, replace = F) # select 50/337 of cases for
testing, train on remaining (337-50)/337 cases
# summary(IBSData); IBSData[IBSData=="."] <- NA; newData <- mice(IBSData,
m=5, maxit=50, meth='pmm', seed=500); summary(newData)
```

```

html_nodes(wiki_url, "#content")
## {xml_nodeset (1)}
## [1] <div id="content">\n\t<a name="top" id="top"></a>\n\t\t\t\t<h1
id= ...

# View (IBSDData); dim(IBSDData): Select an outcome response "DX"(3),
"FS_IQ" (5)

# scale/normalize all input variables
IBSDData <- na.omit(IBSDData)
IBSDData[,4:66] <- scale(IBSDData[,4:66]) # scale the entire dataset
train.x = data.matrix(IBSDData[-test.ind, c(4:66)]) # exclude outcome
train.y = IBSDData[-test.ind, 3]-1
test.x = data.matrix(IBSDData[test.ind, c(4:66)])
test.y = IBSDData[test.ind, 3]-1

# View(data.frame(train.x, train.y))
# View(data.frame(test.x, test.y))
# table(test.y); table(train.y)
# num.round - number of iterations to train the model

act <- mx.symbol.Variable("data")
fc <- mx.symbol.FullyConnected(act, num.hidden = 10)
act <- mx.symbol.Activation(data = fc, act_type = "sigmoid")
fc <- mx.symbol.FullyConnected(act, num.hidden = 2)
mlp <- mx.symbol.SoftmaxOutput(fc)

mx.set.seed(2235)
model <- mx.model.FeedForward.create(
    symbol = mlp,
    array.batch.size=20,
    X = train.x, y=train.y,
    num.round = 200,
    array.Layout = "rowmajor",
    Learning.rate = exp(-1),
    eval.metric = mx.metric.accuracy, verbose=FALSE)
preds = predict(model, test.x)
pred.Label = max.col(t(preds))-1; table(pred.Label, test.y)

##          test.y
## pred.Label 0 1
##          0 23 10
##          1 10  7

library("crossval")
diagnosticErrors(crossval::confusionMatrix(pred.Label, test.y, negative = 0))

##      acc      sens      spec      ppv      npv      lor
## 0.6000000 0.4117647 0.6969697 0.4117647 0.6969697 0.4762342
## attr(,"negative")
## [1] 0

ggplot(data.frame(test.y, preds[2,]),
       aes(x=preds[2,], group=test.y, fill=as.factor(test.y)))+
  geom_histogram(position="dodge", binwidth=0.25)+theme_bw()

```

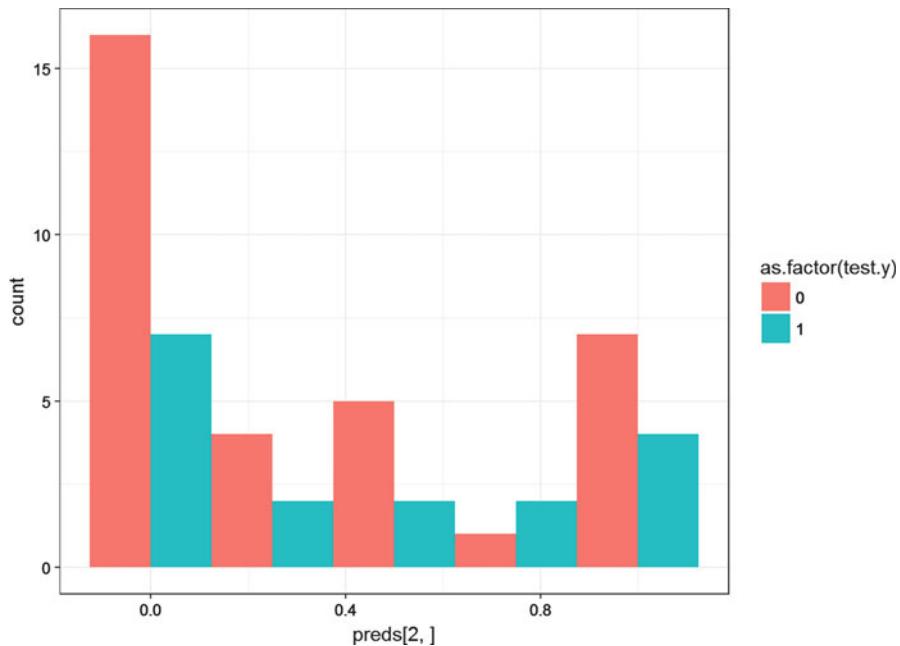


Fig. 23.20 Frequency of the feed-forward neural network prediction probabilities (x-axis) for the IBS data relative to testing set labels (colors)

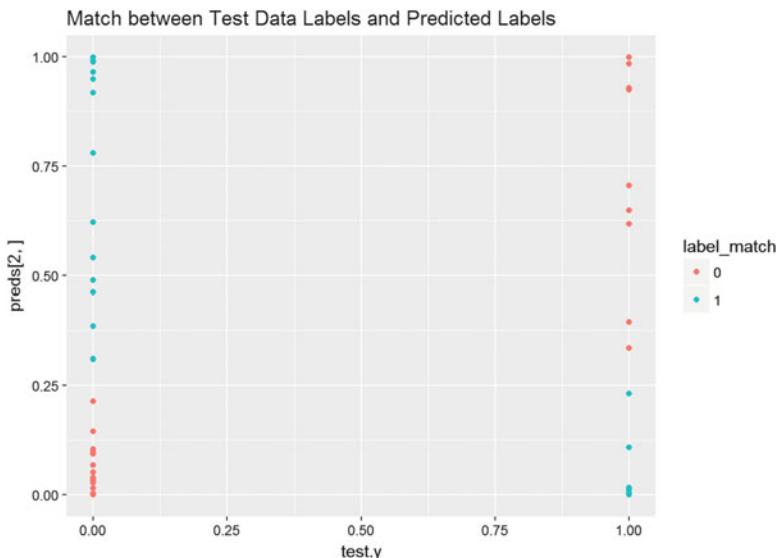


Fig. 23.21 Validation results of the binarized feed-forward neural network prediction probabilities (y-axis) for the IBS testing data (x-axis) with label-coding for match(0)/mismatch(1)

```
# convert pred-probability to binary classes threshold=0.3?
bin_preds <- ifelse (preds[2,]<0.3, 0, 1)
# get a factor variable comparing binary test-labels vs. predicted-labels
label_match <- as.factor(ifelse (test.y==bin_preds, 0, 1))
p5 <- ggplot(data.frame(test.y, preds[2,]), aes(x = test.y, y = preds[2,],
color=label_match))+geom_point() + ggtitle("Match between Test Data Labels and
Predicted Labels")
p5
```

This histogram plot suggests that the classification is not good (Fig. 23.20).

23.5.4 Country QoL Ranking Data

Another case study we have seen before is the country quality of life (QoL) dataset. Let's explore a new neural network model and use it to predict the overall country QoL.

```

X = train.x, y=train.y,
num.round = 15,
array.layout = "rowmajor",
learning.rate = exp(-1),
eval.metric = mx.metric.accuracy)

## Start training with 1 devices
## [1] Train-accuracy=0.4166666666666667
## [2] Train-accuracy=0.442857142857143
## [3] Train-accuracy=0.442857142857143
## [4] Train-accuracy=0.442857142857143
## [5] Train-accuracy=0.442857142857143
## [6] Train-accuracy=0.442857142857143
## [7] Train-accuracy=0.6
## [8] Train-accuracy=0.8
## [9] Train-accuracy=0.914285714285714
## [10] Train-accuracy=0.928571428571429
## [11] Train-accuracy=0.942857142857143
## [12] Train-accuracy=0.942857142857143
## [13] Train-accuracy=0.942857142857143
## [14] Train-accuracy=0.971428571428572
## [15] Train-accuracy=0.971428571428572

preds = predict(model, test.x); preds

##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.5204602 0.8808465 0.007948651 0.009155557 0.8622462 0.8432776
## [2,] 0.4795398 0.1191535 0.992051363 0.990844429 0.1377538 0.1567224
##           [,7]      [,8]      [,9]      [,10]     [,11]     [,12]
## [1,] 0.4493238 0.6563529 0.97970927 0.7055513 0.98414272 0.9647682
## [2,] 0.5506761 0.3436471 0.02029071 0.2944487 0.01585729 0.0352319
##           [,13]     [,14]     [,15]     [,16]     [,17]     [,18]
## [1,] 0.6106228 0.91565907 0.8317797 0.0252018 0.7618818 0.01770884
## [2,] 0.3893772 0.08434091 0.1682204 0.9747981 0.2381181 0.98229110
##           [,19]     [,20]     [,21]     [,22]     [,23]     [,24]
## [1,] 0.007323461 0.7766624 0.94527471 0.007209368 0.09066615 0.007661197
## [2,] 0.992676497 0.2233376 0.05472526 0.992790580 0.90933383 0.992338777
##           [,25]     [,26]     [,27]     [,28]     [,29]     [,30]
## [1,] 0.0489373 0.009559323 0.91361207 0.1901348 0.90563852 0.97519016
## [2,] 0.9510627 0.990440726 0.08638796 0.8098652 0.09436146 0.02480989

pred.Label = max.col(t(preds))-1; table(pred.Label, test.y)

##           test.y
## pred.Label 0 1
##           0 17 1
##           1 1 11

```

We only need 15 rounds to achieve 97% accuracy (Figs. 23.22 and 23.23).

```

ggplot(data.frame(test.y, preds[,2]),
aes(x=preds[,2], group=test.y, fill=as.factor(test.y)))+
  geom_histogram(position="dodge", binwidth=0.25)+theme_bw()

```

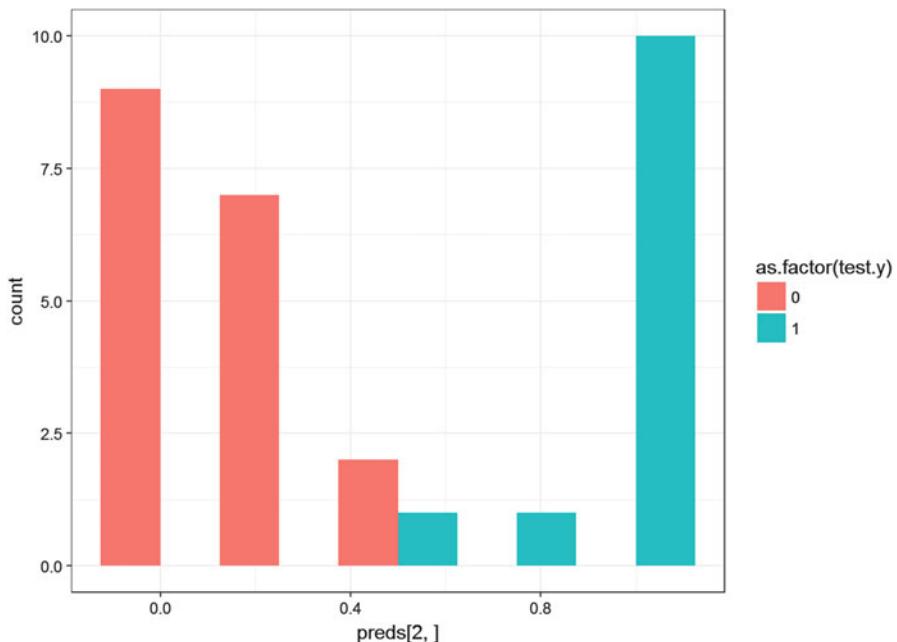


Fig. 23.22 Frequency of the feed-forward neural network prediction probabilities (x-axis) for the QoL data relative to testing set labels (colors)

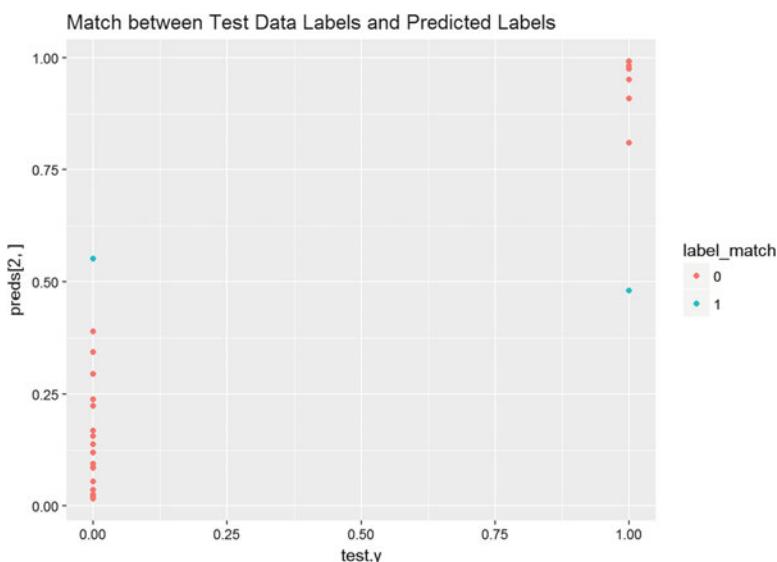


Fig. 23.23 Validation results of the binarized feed-forward neural network prediction probabilities (y-axis) for the QoL testing data with label-coding for match(0)/mismatch(1)

```

#calculate sensitivity & specificity and more
library("crossval")
diagnosticErrors(crossval::confusionMatrix(pred.label,test.y,negative = 0))

##      acc      sens      spec      ppv      npv      lor
## 0.9333333 0.9166667 0.9444444 0.9166667 0.9444444 5.2311086
## attr(,"negative")
## [1] 0

# convert pred-probability to binary classes threshold=0.5?
bin_preds <- ifelse (preds[2,]<0.5, 0, 1)
# get a factor variable comparing binary test-labels vs. predicted-labels
label_match <- as.factor(ifelse (test.y==bin_preds, 0, 1))
p6 <- ggplot(data.frame(test.y, preds[2,]), aes(x = test.y, y = preds[2,],
color=label_match))+geom_point()+ggtitle("Match between Test Data Labels
and Predicted Labels")
p6

```

23.5.5 Handwritten Digits Classification

In Chap. 11 (ML, NN, SVM Classification) we discussed Optical Character Recognition (OCR). Specifically, we analyzed handwritten notes (unstructured text) and converted it to printed text.

MNIST includes a large set of human annotated/labeled handwritten digits imaging data set. Every digit is represented by a 28×28 thumbnail image. You can download the training and testing data from Kaggle.

The `train.csv` and `test.csv` data files contain gray-scale images of hand-drawn digits, 0, 1, 2, ..., 9. Each 2D image is 28×28 in size and each of the 784 pixels has a single pixel-intensity representing the lightness or darkness of that pixel (stored as a 1 byte integer [0,255]). Higher intensities correspond to darker pixels.

The training data, `train.csv`, has 785 columns, where the first column, **label**, codes the actual the digit drawn by the user. The remaining 784 columns contain the $28 \times 28 = 784$ pixel-intensities of the associated 2D image. Columns in the training set have $pixel_K$ names, where $0 \leq K \leq 783$. To reconstruct a 2D image out of each row in the training data we use this relation between pixel-index (K) and X, Y image coordinates:

$$K = Y \times 28 + X,$$

where $0 \leq X, Y \leq 27$. Thus, $pixel_K$ is located on row Y and column X of the corresponding 2D Image of size 28×28 . For instance, $pixel_{60} = (2 \times 28 + 4) \leftrightarrow (X = 4, Y = 2)$ represents the pixel on the third row and fifth column in the image. Diagrammatically, omitting the “pixel” prefix, the pixels may be ordered to reconstruct the 2D image as follows (Table 23.3).

Note that the point-to-pixelID transformation ($K = Y \times 28 + X$) may easily be inverted as a pixelID-to-point mapping: $X = K \bmod 28$ (remainder of the integer division ($K/28$)) and $Y = K$ (integer part of the division $K/28$)). For example:

Table 23.3 Schematic for reconstructing a 28×28 square image using a list of 784 intensities corresponding to colors in the image reflecting the manual handwritten digits

Row	Col0	Col1	Col2	Col3	Col5	...	Col26	Col27
Row0	0	1	2	3	4	...	26	27
Row1	28	29	30	31	32	...	54	55
Row2	56	57	58	59	60	...	82	83
RowK
Row26	728	729	730	731	732	...	754	755
Row27	756	757	758	759	760	...	782	783

```

K <- 60
X <- K %% 28 # X= K mod 28, remainder of integer division 60/28
Y <- K%/%28 # integer part of the division
# This validates that the application of both, the back and forth
transformations, leads to an identity
K; X; Y; Y * 28 + X

## [1] 60
## [1] 4
## [1] 2
## [1] 60

```

The `test` data (`test.csv`) has the same organization as the training data, except that it does not contain the first **label** column. It includes 28,000 images and we can predict image labels that can be stored as *ImageId*, *Label* pairs, which can be visually compared to the 2D images for validation/inspection.

```

require(mxnet)

# train.csv
pathToZip <- tempfile()
download.file("http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650
/data/DigitRecognizer_TrainingData.zip", pathToZip)
train <- read.csv(unzip(pathToZip))
dim(train)

## [1] 42000 785
unlink(pathToZip)

# test.csv
pathToZip <- tempfile()
download.file("http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650
/data/DigitRecognizer_TestingData.zip", pathToZip)
test <- read.csv(unzip(pathToZip))
dim(test)

## [1] 28000 784
unlink(pathToZip)

train <- data.matrix(train)
test <- data.matrix(test)

```

```
train.x <- train[,-1]
train.y <- train[,1]

# Scaling will be discussed below
train.x <- t(train.x/255)
test <- t(test/255)
```

Let's look at some of these example images (Figs. 23.24, 23.25, 23.26 and 23.27):

```
library("imager")
# first convert the CSV data (one row per image, 28,000 rows)
array_3D <- array(test, c(28, 28, 28000))
mat_2D <- matrix(array_3D[,,1], nrow = 28, ncol = 28)
plot(as.cimg(mat_2D))
```

Fig. 23.24 Image rendering of the first handwritten digit, stored as a 28×28 array of intensities

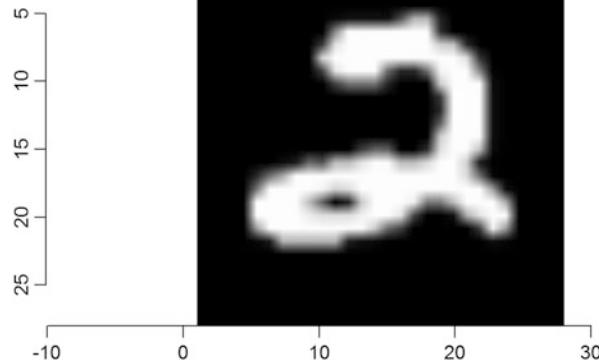


Fig. 23.25 Rendering of the fifth handwritten digit in the list of 28,000

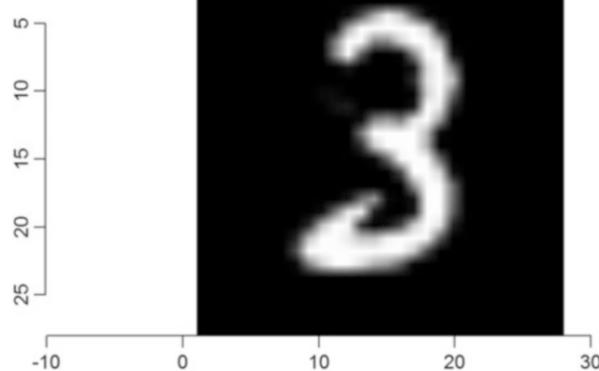


Fig. 23.26 Another strategy for indexing and plotting handwritten digits as 2D images

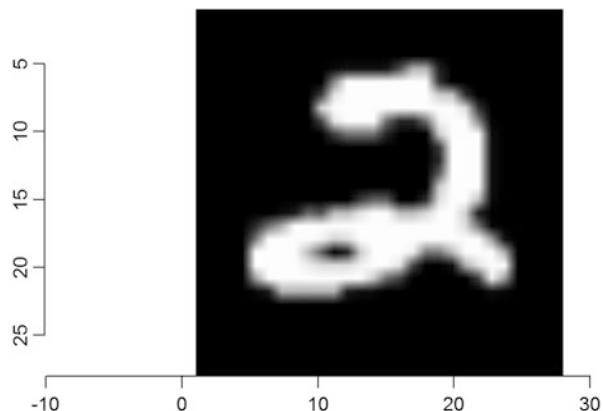


Fig. 23.27 Sequential image plot of the first four handwritten digits



```
# extract all N=28,000 images
N <- 28000
img_3D <- as.cimg(array_3D[, ,], 28, 28, N)

# plot the k-th image (1<=k<=N)
k <- 5
plot(img_3D, k)

image_2D <- function(img, index){
  img[, , index, , drop=FALSE]
}

plot(image_2D(img_3D, 1))
```

```
# Plot a collage of the first 4 images
imappend(List(image_2D(img_3D, 1), image_2D(img_3D, 2), image_2D(img_3D, 3),
image_2D(img_3D, 4)), "y") %>% plot
```

```
# img <- image_2D(img_3D, 1)
# for (i in 10:20) { imappend(list(img, image_2D(img_3D, i)), "x") }
```

In these CSV data files, each 28×28 image is represented as a single row. The intensities of these greyscale images are stored as 1 byte integers, in the range $[0, 255]$, which we linearly transformed into $[0, 1]$. Note that we only scale the X input, not the output (labels). Also, we don't have manual gold-standard validation labels for the testing data, i.e., `t$test.y` is not available for the handwritten digits data.

```
# We already scaled earlier
# train.x <- t(train.x/255)
# test <- t(test/255)
```

Next, we can transpose the input matrix to n (*pixels*) \times m (*examples*), as column major format required by `mxnet`. The image labels are evenly distributed:

```
table(train.y); prop.table(table(train.y))

## train.y
##    0     1     2     3     4     5     6     7     8     9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188

## train.y
##    0     1     2     3     4     5
## 0.09838095 0.11152381 0.09945238 0.10359524 0.09695238 0.09035714
##    6     7     8     9
## 0.09850000 0.10478571 0.09673810 0.09971429
```

The majority class (1) in the training set includes 11.2% of the observations.

Configuring the Neural Network

```
data <- mx.symbol.Variable("data")
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
```

`data <- mx.symbol.Variable("data")` represents the input layer. The first hidden layer, set by `fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)`, takes the data as an input, its name, and the number of hidden neurons to generate an output layer.

`act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")` sets the activation function, which takes the output from the first hidden layer "fc1" and

generates an output that is fed into the second hidden layer "fc2", which uses fewer hidden neurons (64).

The process repeats with the second activation "act2", resembling "act1" but using different input source and name. As there are only ten digits (0, 1, . . . , 9), in the last layer "fc3", we set the number of neurons to 10. At the end, we set the activation to softmax to obtain a probabilistic prediction.

Training

We are almost ready for the training process. Before we start the computation, let's decide what device we should use.

```
devices <- mx.cpu()
```

Here we assign CPU to mxnet. After all these preparation, you can run the following command to train the neural network! Note that in mxnet, the correct function to control the random process is mx.set.seed.

```
mx.set.seed(1234)
model <- mx.model.FeedForward.create(softmax, X=train.x, y=train.y,
                                      ctx=devices, num.round=10, array.batch.size=100,
                                      learning.rate=0.07, momentum=0.9, eval.metric=mx.metric.accuracy,
                                      initializer=mx.init.uniform(0.07),
                                      epoch.end.callback=mx.callback.log.train.metric(100)
)
## Start training with 1 devices
## [1] Train-accuracy=0.863031026252982
## [2] Train-accuracy=0.958285714285716
## [3] Train-accuracy=0.970785714285717
## [4] Train-accuracy=0.977857142857146
## [5] Train-accuracy=0.983238095238099
## [6] Train-accuracy=0.98521428571429
## [7] Train-accuracy=0.987095238095242
## [8] Train-accuracy=0.989309523809528
## [9] Train-accuracy=0.99214285714286
## [10] Train-accuracy=0.991452380952384
```

For 10 rounds, the training accuracy exceeds 99%. It may not be worthwhile trying 100 rounds, as this would increase substantially the computational complexity.

Forecasting

Now, we will demonstrate how to generate a forecasting model based on testing data, and how to evaluate its prediction performance. The pred matrix has 28,000 rows and 10 columns, containing the desired classification probabilities from the output layer of the neural net. To extract the maximum label for each row, we can use the max.col:

```

# evaluate: "preds" is the matrix of the possibility of each of the 10
# numbers
preds <- predict(model, test)

pred.Label <- max.col(t(preds)) - 1
table(pred.Label)

## pred.Label
##   0   1   2   3   4   5   6   7   8   9
## 2774 3228 2862 2728 2781 2401 2777 2868 2826 2755

# preds1 <- ifelse(preds[2,] <= 0.5, 0, 1) # dichotomize to labels
# pred.label = t(preds1)
# table(pred.label, test.y)
# calculate sensitivity & specificity
# sensitivity(factor(preds1), factor(as.numeric(test.y)),positive = 1)
# specificity(factor(preds1), factor(as.numeric(test.y)),negative = 0)
# preds <- predict(model, test.x)
# dim(preds)
# preds1 <- ifelse(preds[2,] <= 0.5, 0, 1) # dichotomize to labels
# pred.label = t(preds1)
# table(pred.label, test.y)

```

For binary classification, mxnet outputs two prediction classes, whereas for multi-class predictions, it outputs a matrix of size n (*classes*) \times m (*examples*), where the *rows* correspond to the probability of the class in the specific *column*, so all column sums add up to 1.0.

The predictions are stored in a $28,000$ (*rows*) \times 10 (*columns*) matrix, including the desired classification probabilities from the output layer. The R `max.col` function extracts the maximum label for each row.

```

pred.Label <- max.col(t(preds)) - 1
table(pred.Label)

## pred.Label
##   0   1   2   3   4   5   6   7   8   9
## 2774 3228 2862 2728 2781 2401 2777 2868 2826 2755

```

We can save the predicted labels of the testing handwritten digits to CSV:

```

predicted_Labels <- data.frame(ImageId=1:ncol(test), Label=pred.Label)
write.csv(predicted_Labels, file='predicted_Labels.csv', row.names=FALSE,
quote=FALSE)

```

We can open the `predicted_Labels.csv` file and inspect the ML-labels (saved in the 2-column `ImageID` and `Label` format CSV) assigned to the 28,000 manually drawn digits. As the **testing handwritten digits data** do not have human-provided labels, we can't quantitatively assess the validity of the algorithm on the testing data (Fig. 23.28). However, we can visually inspect random handwritten digit instances (7 in the example below, image indices 4 : 10) against their predictions and gain intuition of the accuracy rate of the ML classifier (Table 23.4, Fig. 23.29).

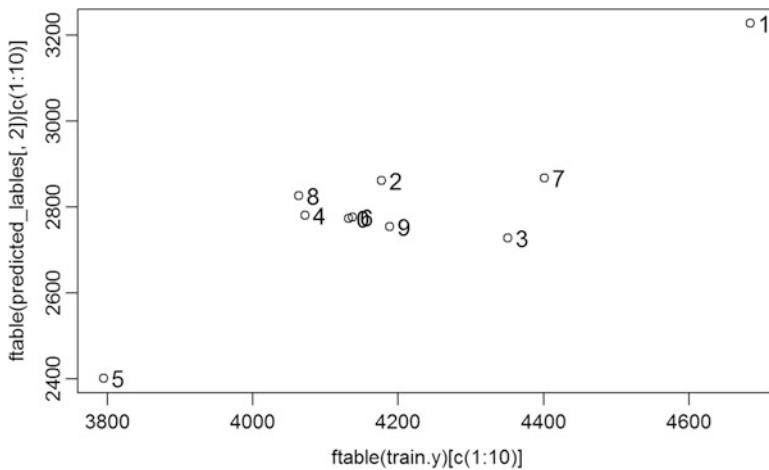
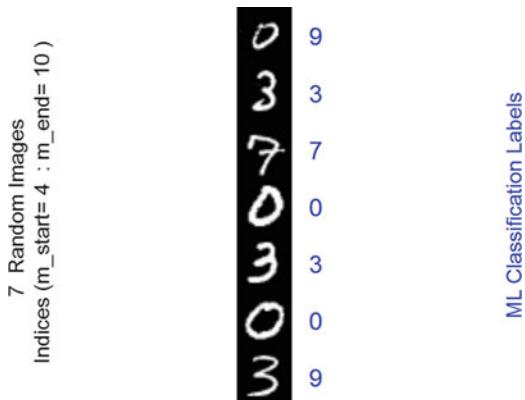


Fig. 23.28 Plot of the agreement between relative frequencies in the number of train.y labels (in range 0–9) against the testing data predicted labels. These quantities are not directly related (frequencies of digits in training.y and predicted.testing.data); we can't explicitly validate the testing-data predictions, as we don't have gold-standard test.y labels! However, numbers closer to the diagonal of the plot would indicate expected good classifications, whereas, off diagonal points may suggest less effective labeling

Table 23.4 Predicted labels for the set of the first 7 handwritten digits

ImageId	Label
1	2
2	0
3	9
4	9
5	3
6	7
7	0

Fig. 23.29 Visual validation of the handwritten digits (left) and their neural network prediction (right) for the set of seven images. The number and indices of these testing data images can be manually specified



```



```

```
table(ftable(train.y)[c(1:10)], ftable(predicted_labels[,2])[c(1:10)])
```

```
##
```

	2401	2728	2755	2774	2777	2781	2826	2862	2868	3228
## 3795	1	0	0	0	0	0	0	0	0	0
## 4063	0	0	0	0	0	0	1	0	0	0
## 4072	0	0	0	0	0	1	0	0	0	0
## 4132	0	0	0	1	0	0	0	0	0	0
## 4137	0	0	0	0	1	0	0	0	0	0
## 4177	0	0	0	0	0	0	0	1	0	0
## 4188	0	0	1	0	0	0	0	0	0	0
## 4351	0	1	0	0	0	0	0	0	0	0
## 4401	0	0	0	0	0	0	0	0	1	0
## 4684	0	0	0	0	0	0	0	0	0	1

Examining the Network Structure Using LeNet

We can use the mxnet package LeNet convolutional neural network (CNN) protocol for learning the network.

Let's first construct the network.

```
# input
data <- mx.symbol.Variable('data')
# first conv
conv1 <- mx.symbol.Convolution(data=data, kernel=c(5,5), num_filter=20)
tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max",
                           kernel=c(2,2), stride=c(2,2))
# second conv
conv2 <- mx.symbol.Convolution(data=pool1, kernel=c(5,5), num_filter=50)
tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max",
                           kernel=c(2,2), stride=c(2,2))
# first fullc
flatten <- mx.symbol.Flatten(data=pool2)
fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
# second fullc
fc2 <- mx.symbol.FullyConnected(data=tanh3, num_hidden=10)
# loss
Lenet <- mx.symbol.SoftmaxOutput(data=fc2)
```

Next, we will reshape the matrices into arrays.

```
train.array <- train.x
dim(train.array) <- c(28, 28, 1, ncol(train.x))
test.array <- test
dim(test.array) <- c(28, 28, 1, ncol(test))
```

Compare the training speed on different devices – CPU vs. GPU. Start by defining the devices.

```

n.gpu <- 1
device.cpu <- mx.cpu()
device.gpu <- lapply(0:(n.gpu-1), function(i) {
  mx.gpu(i)
})

```

Passing a list of devices is useful for high-end computational platforms (e.g., multi-GPU systems); mxnet can train on multiple GPUs or CPUs.

To train using the CPU, try fewer iterations as protocol is computationally very intense.

```

mx.set.seed(1234)
tic <- proc.time()
model <- mx.model.FeedForward.create(lenet, X=train.array, y=train.y,
  ctx=device.cpu, num.round=1, array.batch.size=100,
  learning.rate=0.05, momentum=0.9, wd=0.0001,
  eval.metric=mx.metric.accuracy,
  epoch.end.callback=mx.callback.log.train.metric(100))

## Start training with 1 devices
## [1] Train-accuracy=0.522267303102625

print(proc.time() - tic)

##    user    system   elapsed
## 313.22    66.45   50.94

```

The corresponding training on GPU is similar, but it requires a separate GPU-compilation of mxnet (/mxnet/src/storage/storage.cc:78) with USE_CUDA=1 to enable GPU usage.

```

mx.set.seed(1234)
tic <- proc.time()
model <- mx.model.FeedForward.create(lenet, X=train.array, y=train.y,
  ctx=device.gpu, num.round=5, array.batch.size=100,
  learning.rate=0.05, momentum=0.9, wd=0.0001,
  eval.metric=mx.metric.accuracy,
  epoch.end.callback=mx.callback.log.train.metric(100))

print(proc.time() - tic)

```

GPU training is faster than CPU. Everyone can submit a new classification result to Kaggle and see a ranking result for their classifier. Make sure you follow the specific result-file submission format.

```

preds <- predict(model, test.array)
pred.label <- max.col(t(preds)) - 1
submission <- data.frame(ImgId=1:ncol(test), Label=pred.label)
write.csv(submission, file='submission.csv', row.names=FALSE, quote=FALSE)

```

23.6 Classifying Real-World Images

A real-world example of deep learning is classification of 2D images (pictures) or 3D volumes (e.g., neuroimages).

The image classification examples below shows the use a pre-trained **Inception-BatchNorm Network** to predict a class of real world image. The network architecture is described the 2015 Ioffe and Szegedy paper. The pre-trained Inception-BatchNorm network is available online. This advanced model gives a state-of-the-art prediction accuracy on imaging data. We also need the R `imager` package to load and preprocess the 2D images.

```
# install.packages("imager")
require(mxnet)
require(imager)
```

23.6.1 Load the Pre-trained Model

Download and unzip the pre-trained model to a working folder, and load the model and the mean image (used for preprocessing) using `mx.nd.load` into R. This download can either be done manually, or automated, as shown below.

```
pathToZip <- tempfile()
download.file("http://www.socr.umich.edu/people/dinov/2017/Spring/DSPA_HS650
/data/Inception.zip", pathToZip)
model_file <- unzip(pathToZip)

# setwd(paste(getwd(),"results", sep='/'))
model = mx.model.Load(paste(getwd(),"Inception_BN", sep='/'), iteration=39)

mean.img = as.array(mx.nd.Load(
  paste(getwd(),"mean_224.nd", sep='/')
  )
[[ "mean_img" ]]

)
dim(mean.img)
## [1] 224 224 3

# plot(mean.img)
```

23.6.2 Load, Preprocess and Classify New Images – US Weather Pattern

To classify a new image, select the image and load it in. Below, we show the classification of several alternative images (Fig. 23.30).

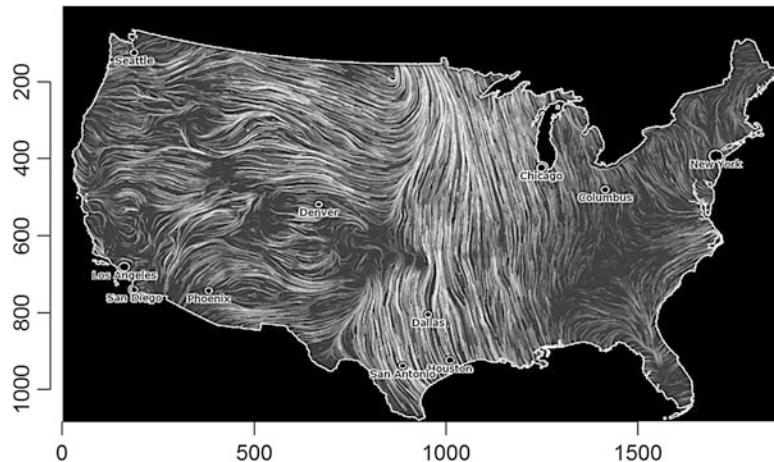


Fig. 23.30 A U.S. weather pattern map as an example image for neural network image recognition

```
library("imager")
# One should be able to load the image directly from the web (but sometimes
# there may be problems, in which case, we need to first download the image
# and then load it in R:
# im <-
imager::load.image("http://wiki.socr.umich.edu/images/6/69/DataManagement
Fig1.png")

# download file to local working directory, use "wb" mode to avoid problems
download.file("http://wiki.socr.umich.edu/images/6/69/DataManagementFig1.png",
", paste(getwd(),"results/image.png", sep="/"), mode = 'wb')

# report download image path
paste(getwd(),"results/image.png", sep="/")

img <- Load.image(paste(getwd(),"results/image.png", sep="/"))
dim(img)

## [1] 1875 1084      1      4
plot(img)
```

Before feeding the image to the deep learning network for classification, we need to do some preprocessing to make it fit the `deeplnet` input requirements. This image preprocessing (cropping and subtraction of the mean) can be done directly in R.

```
preproc.image <- function(im, mean.image) {
  # crop the image
  shape <- dim(im)
  short.edge <- min(shape[1:2])
  xx <- floor((shape[1] - short.edge) / 2)
  yy <- floor((shape[2] - short.edge) / 2)
  cropped <- crop.borders(im, xx, yy)
  # resize to 224 x 224, needed by input of the model.
```

```
resized <- resize(cropped, 224, 224)
plot(resized)
# convert to array (x, y, channel)
arr <- as.array(resized[,,,c(1:3)]) * 255
plot(as.cimg(arr))
dim(arr) <- c(224, 224, 3)
# subtract the mean
normed <- arr - mean.img
# Reshape to format needed by mxnet (width, height, channel, num)
dim(normed) <- c(224, 224, 3, 1)
return(normed)
}
```

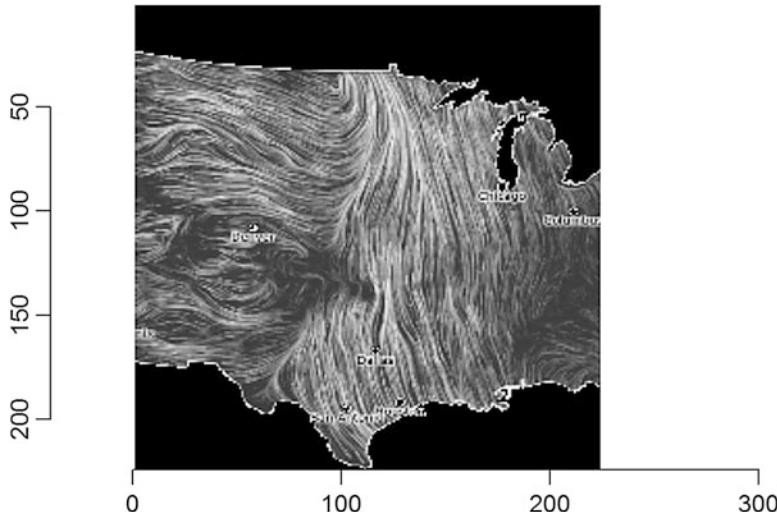


Fig. 23.31 Normalized US weather pattern map image

Call the preprocessing function with the normalized image (Fig. 23.31).

```
normed <- preproc.image(img, mean.img)
# plot(normed)
```

The image classification uses a *predict* function to get the probability over all (learned) classes.

```
prob <- predict(model, X=normed)
dim(prob)
## [1] 1000      1
```

The `prob` prediction generates a $1,000 \times 1$ array representing the probability of the input image to resemble (be classified as) the top 1,000 known image categories. We can report the indices of the top-10 closest image classes to the input image:

```
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
max.idx
## [1] 855 563 229 581 620 948 951 186 204 311
```

Alternatively, we can map these top-10 indices into named `image-classes`.

```
synsets <- readLines("synset.txt")
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx], " "
; Probability: ", prob[max.idx]))
## [1] "Top Predicted Image-Label Classes: Name=n04418357 theater curtain,
theatre curtain; Probability: 0.0493971668183804"
## [2] "Top Predicted Image-Label Classes: Name=n03388043 fountain;
Probability: 0.0431815795600414"
## [3] "Top Predicted Image-Label Classes: Name=n02105505 komondor;
Probability: 0.0371582210063934"
## [4] "Top Predicted Image-Label Classes: Name=n03457902 greenhouse,
nursery, glasshouse; Probability: 0.0368415862321854"
## [5] "Top Predicted Image-Label Classes: Name=n03637318 Lampshade,
Lamp shade; Probability: 0.0317880213260651"
## [6] "Top Predicted Image-Label Classes: Name=n07734744 mushroom;
Probability: 0.0292572267353535"
## [7] "Top Predicted Image-Label Classes: Name=n07747607 orange;
Probability: 0.0284675862640142"
## [8] "Top Predicted Image-Label Classes: Name=n02094114 Norfolk terrier;
Probability: 0.026896309107542"
## [9] "Top Predicted Image-Label Classes: Name=n02098286 West Highland
white terrier; Probability: 0.0257413759827614"
## [10] "Top Predicted Image-Label Classes: Name=n02219486 ant, emmet,
pismire; Probability: 0.0205500852316618"
```

Clearly, this U.S. weather pattern image is not well classified. The optimal prediction suggests this may be a *theater curtain*; however, the confidence is very low, $Prob \sim 0.049$. None of the other top-10 classes capture the type of the actual image either.

The machine learning image classifications results won't always be this poor. Let's try classifying several alternative images.

23.6.3 Lake Mapourika, New Zealand

Let's try the automated image classification of this lakeside panorama (Figs. 23.32 and 23.33).

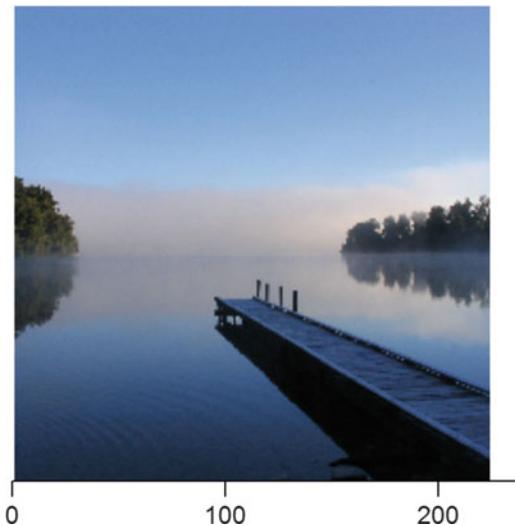
```
download.file("https://upload.wikimedia.org/wikipedia/commons/2/23/Lake_mapourika_NZ.jpeg", paste(getwd(), "results/image.png", sep="/"), mode = 'wb')
im <- Load.image(paste(getwd(), "results/image.png", sep="/"))
```

`plot(im)`

Fig. 23.32 A lakeside panorama image for neural network image recognition



Fig. 23.33 Normalized lakeside panorama image



```

normed <- preproc.image(im, mean.img)

prob <- predict(model, X=normed)
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx], " "
; Probability: ", prob[max.idx])))

## [1] "Top Predicted Image-Label Classes: Name=n02894605 breakwater,
groin, groyne, mole, bulwark, seawall, jetty; Probability:0.648901104927063"
## [2] "Top Predicted Image-Label Classes: Name=n03216828 dock, dockage,
docking facility; Probability: 0.183006703853607"
## [3] "Top Predicted Image-Label Classes: Name=n09332890 Lakeside,
Lakeshore; Probability: 0.127718329429626"
## [4] "Top Predicted Image-Label Classes: Name=n03160309 dam, dike, dyke;
Probability: 0.0115784741938114"
## [5] "Top Predicted Image-Label Classes: Name=n03095699 container ship,
containership, container vessel; Probability: 0.00913785584270954"
## [6] "Top Predicted Image-Label Classes: Name=n09428293 seashore, coast,
seacoast, sea-coast; Probability: 0.0043862983584404"
## [7] "Top Predicted Image-Label Classes: Name=n03933933 pier;
Probability: 0.00410780590027571"
## [8] "Top Predicted Image-Label Classes: Name=n02859443 boathouse;
Probability: 0.00246214028447866"
## [9] "Top Predicted Image-Label Classes: Name=n09399592 promontory,
headland, head, foreland; Probability: 0.00168424111325294"
## [10] "Top Predicted Image-Label Classes: Name=n09421951 sandbar,
sand bar; Probability: 0.00106814480386674"

```

This photo does represent a lakeside, which is reflected by the top three class labels:

- Breakwater, groin, groyne, mole, bulwark, seawall, jetty.
- Dock, dockage, docking facility.
- Lakeside, lakeshore.

23.6.4 Beach Image

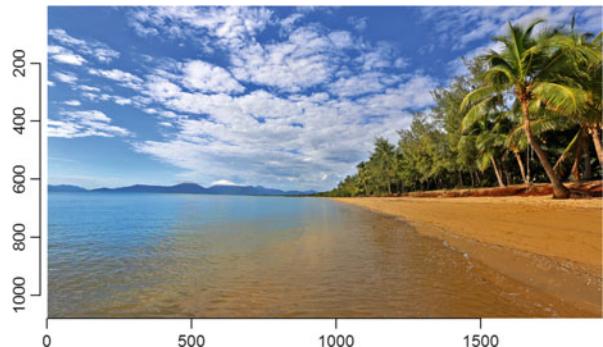
Another costal boundary between water and land is represented in this beach image (Fig. 23.34).

```

download.file("https://upload.wikimedia.org/wikipedia/commons/9/90/Holloways
_beach_1920x1080.jpg", paste(getwd(),"results/image.png", sep="/"), mode =
'wb')
im <- Load.image(paste(getwd(),"results/image.png", sep="/"))
plot(im)

```

Fig. 23.34 A beach image for neural network image recognition



```
# normed <- preproc.image(im, mean.img)
prob <- predict(model, X=normed)
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx],
"; Probability: ", prob[max.idx]))
## [1] "Top Predicted Image-Label Classes: Name=n09421951 sandbar,
## sand bar; Probability: 0.69039398431778"
## [2] "Top Predicted Image-Label Classes: Name=n09332890 lakeside,
## Lakeshore; Probability: 0.20282569527626"
## [3] "Top Predicted Image-Label Classes: Name=n09428293 seashore,
## coast, seacoast, sea-coast; Probability: 0.0899285301566124"
## [4] "Top Predicted Image-Label Classes: Name=n02894605 breakwater,
## groin, groyne, mole, bulwark, seawall, jetty; Probability: 0.006692836"
## [5] "Top Predicted Image-Label Classes: Name=n09399592 promontory,
## headland, head, foreland; Probability: 0.00204332848079503"
## [6] "Top Predicted Image-Label Classes: Name=n02859443 boathouse;
## Probability: 0.00106108584441245"
## [7] "Top Predicted Image-Label Classes: Name=n02951358 canoe;
## Probability: 0.000664844119455665"
## [8] "Top Predicted Image-Label Classes: Name=n09246464 cliff, drop,
## drop-off; Probability: 0.000416322873206809"
## [9] "Top Predicted Image-Label Classes: Name=n04357314 sunscreen,
## sunblock, sun blocker; Probability: 0.000338666519382969"
## [10] "Top Predicted Image-Label Classes: Name=n04606251 wreck;
## Probability: 0.000292503653327003"
```

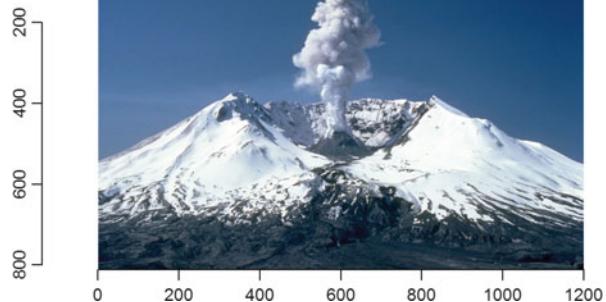
This photo was classified appropriately and with high-confidence as:

- Sandbar, sand bar.
- Lakeside, lakeshore.
- Seashore, coast, seacoast, sea-coast.

23.6.5 Volcano

Here is another natural image representing the Mount St. Helens Volcano (Fig. 23.35).

Fig. 23.35 A volcano image for neural network image recognition



```
download.file("https://upload.wikimedia.org/wikipedia/commons/thumb/d/dc/MSH82_st_helens_plume_from_harrys_ridge_05-19-82.jpg/1200px-MSH82_st_helens_plume_from_harrys_ridge_05-19-82.jpg", paste(getwd(), "results/image.png", sep="/"), mode = 'wb')
im <- load.image(paste(getwd(), "results/image.png", sep="/"))

plot(im)
```

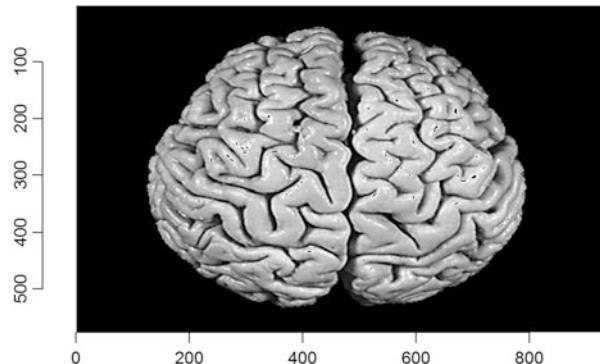
```
prob <- predict(model, X=normed)
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx],
"; Probability: ", prob[max.idx]))

## [1] "Top Predicted Image-Label Classes: Name=n09472597 volcano;
Probability: 0.993182718753815"
## [2] "Top Predicted Image-Label Classes: Name=n09288635 geyser;
Probability: 0.00681292032822967"
## [3] "Top Predicted Image-Label Classes: Name=n09193705 alp;
Probability: 4.15803697251249e-06"
## [4] "Top Predicted Image-Label Classes: Name=n03344393 fireboat;
Probability: 1.48333114680099e-07"
## [5] "Top Predicted Image-Label Classes: Name=n04310018 steam locomotive;
Probability: 1.17537313215621e-08"
## [6] "Top Predicted Image-Label Classes: Name=n03388043 fountain;
Probability: 7.4444117537098e-09"
## [7] "Top Predicted Image-Label Classes: Name=n04228054 ski;
Probability: 2.90055357510255e-09"
## [8] "Top Predicted Image-Label Classes: Name=n02950826 cannon;
Probability: 2.27150032117152e-09"
## [9] "Top Predicted Image-Label Classes: Name=n03773504 missile;
Probability: 1.69992575571598e-09"
## [10] "Top Predicted Image-Label Classes: Name=n04613696 yurt;
Probability: 1.25635490899612e-09"
```

The predicted top class labels for this image are perfect:

- Volcano.
- Geyser.
- Alp.

Fig. 23.36 A cortical brain surface image for neural network image recognition



23.6.6 Brain Surface

The next image represents a 2D snapshot of 3D shape reconstruction of a brain cortical surface. This image is particularly difficult to automatically classify because (1) few people have ever seen a real brain, (2) the mathematical and computational models used to obtain the 2D manifold representing the brain surface do vary, and (3) the patterns of sulcal folds and gyral crests are quite inconsistent between people (Fig. 23.36).

```
download.file("http://wiki.socr.umich.edu/images/e/ea/BrainCortex2.png", paste(getwd(),"results/image.png", sep="/"), mode = 'wb')
im <- Load.image(paste(getwd(),"results/image.png", sep="/"))

plot(im)

# normed <- preproc.image(im, mean.img)

prob <- predict(model, X=normed)
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx],
"; Probability: ", prob[max.idx]))

## [1] "Top Predicted Image-Label Classes: Name=n01917289 brain coral;
Probability: 0.4974305331707"
## [2] "Top Predicted Image-Label Classes: Name=n07734744 mushroom;
Probability: 0.229991897940636"
## [3] "Top Predicted Image-Label Classes: Name=n13052670 hen-of-the-woods,
hen of the woods, Polyporus frondosus, Grifola frondosa;
Probability: 0.0925175696611404"
## [4] "Top Predicted Image-Label Classes: Name=n03598930 jigsaw puzzle;
Probability: 0.0433991812169552"
## [5] "Top Predicted Image-Label Classes: Name=n07718747 artichoke,
globe artichoke; Probability: 0.0150045640766621"
## [6] "Top Predicted Image-Label Classes: Name=n07860988 dough;
Probability: 0.0124379806220531"
## [7] "Top Predicted Image-Label Classes: Name=n07715103 cauliflower;
Probability: 0.0115451859310269"
## [8] "Top Predicted Image-Label Classes: Name=n12985857 coral fungus;
Probability: 0.0109992604702711"
## [9] "Top Predicted Image-Label Classes: Name=n07714990 broccoli;
Probability: 0.00990161567687988"
## [10] "Top Predicted Image-Label Classes: Name=n03637318 Lampshade,
Lamp shade; Probability: 0.00754355266690254"
```

The top class labels for the brain image are:

- Brain coral.
- Mushroom.
- Hen-of-the-woods, hen of the woods, *Polyporus frondosus*, *Grifola frondosa*.
- Jigsaw puzzle.

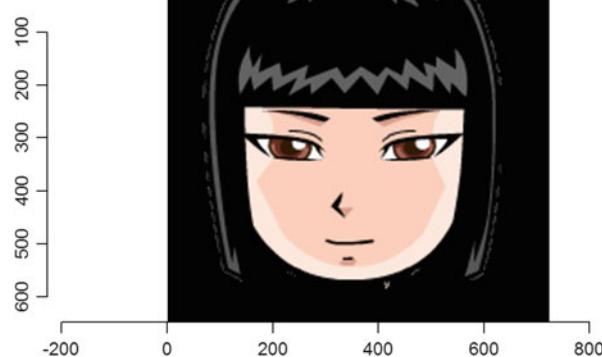
Imagine if we can train a brain image classifier that labels individuals (volunteers or patients) solely based on their brain scans into different classes reflecting their development state, clinical phenotypes, disease traits, or aging profiles. This will require a substantial amount of expert-labeled brain scans, intense model training and extensive validation. However, any progress in this direction will lead to effective computational clinical decision support systems that can assist physicians with diagnosis, tracking, and prognostication of brain growth and aging in health and disease.

23.6.7 Face Mask

The last example is a synthetic computer-generated image representing a cartoon face or a mask (Fig. 23.37).

```
download.file("http://wiki.socr.umich.edu/images/f/fb/FaceMask1.png",
  paste(getwd(), "results/image.png", sep="/"), mode = 'wb')
im <- Load.image(paste(getwd(), "results/image.png", sep="/"))
plot(im)
```

Fig. 23.37 A facial mask image for neural network image recognition



```

prob <- predict(model, X=normed)
max.idx <- order(prob[,1], decreasing = TRUE)[1:10]
print(paste0("Top Predicted Image-Label Classes: Name=", synsets[max.idx],
"; Probability: ", prob[max.idx]))
```

```

## [1] "Top Predicted Image-Label Classes: Name=n03724870 mask;
Probability: 0.376201003789902"
## [2] "Top Predicted Image-Label Classes: Name=n04229816 ski mask;
Probability: 0.253164798021317"
## [3] "Top Predicted Image-Label Classes: Name=n02708093 analog clock;
Probability: 0.0562068484723568"
## [4] "Top Predicted Image-Label Classes: Name=n02865351 bolo tie, bolo,
bola tie, bola; Probability: 0.029578423127532"
## [5] "Top Predicted Image-Label Classes: Name=n04192698 shield, buckler;
Probability: 0.0278499200940132"
## [6] "Top Predicted Image-Label Classes: Name=n03590841 jack-o'-Lantern;
Probability: 0.0175030305981636"
## [7] "Top Predicted Image-Label Classes: Name=n02974003 car wheel;
Probability: 0.0172393135726452"
## [8] "Top Predicted Image-Label Classes: Name=n07892512 red wine;
Probability: 0.0168519839644432"
## [9] "Top Predicted Image-Label Classes: Name=n03249569 drum,
membranophone, tympan; Probability: 0.0141900414600968"
## [10] "Top Predicted Image-Label Classes: Name=n04447861 toilet seat;
Probability: 0.013601747341454"
```

The top class labels for the face mask are:

- Mask.
- Ski mask.
- Analog clock.

You can easily test the same image classifier on your own images and identify classes of pictures that are either well or poorly classified by the deep learning based machine learning model.

23.7 Assignment: 23. Deep Learning, Neural Networks

23.7.1 Deep Learning Classification

- Download the Alzheimer's data from the SOCR Archive.
- Properly preprocess the data and remove outliers.
- Build a multi-layer perceptron as a classifier and select proper parameters.
- Classify AD and NC and report the detailed classification accuracy metrics using cross table, accuracy, sensitivity, specificity, LOR, AUC.
- Generate some data/results visualizations, at least include histograms and model graph structures. See Chap. 23.
- Try to construct a deeper and more elaborate network model and report the prediction results.
- Compare your results with alternative data-driven methods (e.g., KNN).

23.7.2 Deep Learning Regression

- Download the Allometric relationship data from SOCR data.
- Preprocess the data and set density as the response variable.
- Create an MXNet feedforward neural net model and properly specify the parameters.
- Train a model, predict, and report RMSE on the test data, evaluate the result, and justify your evaluation.
- Output the model's structure.

23.7.3 Image Classification

Apply the deep learning neural network techniques to classify some images using the pre-trained model as demonstrated in this chapter:

- Google images.
- SOCR Neuroimaging data.
- Your own images.

References

- Carneiro, G, Mateus, D, Loïc, P, Bradley, A, Manuel, J, Tavares, RS, Belagiannis, V, Papa, JP, Jacinto, C, Loog, M, Lu, Z, Cardoso, JS, Cornebise, J (eds). (2016) *Deep Learning and Data Labeling for Medical Applications: First International Workshop, LABELS 2016*, Springer, ISBN 3319469762, 9783319469768.
- Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:150203167. 2015.
- Wiley, JF. (2016) *R Deep Learning Essentials*, Packt Publishing, ISBN 1785284711, 9781785284717.
- Zhou, K, Greenspan, H, Shen, D. (2017) *Deep Learning for Medical Image Analysis*, Academic Press, ISBN 0128104090, 9780128104095.
- MXNET R Tutorial.
- Deep Learning with MXNetR.
- Deep Neural Networks.
- Google's TensorFlow API.
- <https://github.com/dmlc/mxnet/blob/master/R-package/vignettes/classifyRealImageWithPretrainedModel.Rmd>

Summary

The amount, complexity, and speed of aggregation of biomedical and healthcare data will rapidly increase over the next decade. It's likely to double every 1–2 years. This is fueled by enormous strides in digital and communication technologies, IoT devices, and Cloud services, as well as rapid algorithmic, computational and hardware advances. The proliferating public demand for (near) real-time detection, precise interpretation, and reliable prognostication of human conditions in health and disease also accelerates that trend.

The future does look promising despite the law of diminishing returns, which dictates that sustaining the trajectory clinical gains and the speed of breakthrough developments derived from this increased volume of information, paired with our ability to interpret it, will demand increasingly more resources. Even incremental advances, partial solutions, or lower rates of progress will likely lead to substantive improvements in many human experiences and enhanced medical treatments. Figure 1 below illustrates a common predictive analytics protocol for interrogating big and complex biomedical and health datasets. The process starts by identifying a challenge, followed by determining the sources of data and meta-data, cleaning, harmonizing and wrangling the data components, preprocessing the aggregated archive, model-based and model-free scientific inference, and ends with prediction, validation, and dissemination of data, software, protocols, and research findings.

Our long-term success will require major headways on multiple fronts of data science and predictive analytics. There are urgent demands to develop new algorithms and optimize existing ones, introduce novel computational infrastructure, as well as enhance the abilities of the workforce by overhauling education and training activities. Data science and predictive analytics represents a new and transdisciplinary field, where engagement of heterogeneous experts, multi-talented team-work, and open-science collaborations will be of paramount importance.

The DSPA textbook attempts to lay the foundation for some of the techniques, strategies, and approaches driving contemporary analytics involving Big Data (large size, complex formats, incomplete observations, incongruent features, multiple sources, and multiple scales). It includes some of the mathematical formalisms,



Fig. 1 Major steps in a general predictive data analytics protocol

computational algorithms, machine learning procedures, and demonstrations for Big Data visualization, simulation, mining, pattern identification, forecasting and interpretation.

This textbook (1) contains a transdisciplinary treatise of predictive health analytics; (2) provides a complete and self-contained treatment of the theory, experimental modeling, system development, and validation of predictive health analytics; (3) includes unique case-studies, advanced scientific concepts, lightweight tools, web demos, and end-to-end workflow protocols that can be used to learn, practice, and apply to new challenges; and (4) includes unique interactive content supported by the active community of over 100,000 *R*-developers. These techniques can be translated to many other disciplines (e.g., social network and sentiment analysis, environmental applications, operations research, and manufacturing engineering).

The following two examples may contextually explain the need for inventive data-driven science, computational abilities, interdisciplinary expertise, and modern technologies necessary to achieve desired outcomes, like improving human health, or optimizing future returns on investment. These aims can only be accomplished by experienced teams of researchers who can develop robust decision support systems using modern techniques and protocols, like the ones described in this textbook.

- A **geriatric neurologist** is examining a patient complaining of gait imbalance and postural instability. To determine if the patient may have Parkinson's disease, the physician acquires clinical, cognitive, phenotypic, imaging, and genetics data (Big Healthcare Data). Currently, most clinics and healthcare centers are not equipped with skilled data analysts that can wrangle, harmonize and interpret such complex datasets, nor do they have access to normative population-wide summaries. *A reader that completes the DSPA course of study will have the basic competency and ability to manage the data, generate a protocol for deriving candidate biomarkers, and provide an actionable decision support system.* This protocol will help the physician understand holistically the patient's health and make a comprehensive evidence-based clinical diagnosis as well as provide a data-driven prognosis.
- To improve the return on investment for their shareholders, a **healthcare manufacturer** needs to forecast the demand for their new product based on observed environmental, demographic, market conditions, and bio-social sentiment data. This clearly represents another example of Big Biosocial Data. The organization's data-analytics team is tasked with building a workflow that identifies, aggregates, harmonizes, models and analyzes all available data elements to generate a trend forecast. This system needs to provide an automated, adaptive, scalable, and

reliable prediction of the optimal investment and R&D allocation that maximizes the company's bottom line. *Readers that complete the materials in the DSPA textbook will be able to ingest the observed structured and unstructured data, mathematically represent the data as a unified computable object, apply appropriate model-based and model-free prediction techniques to forecast the expected relation between the company's investment, product manufacturing costs, and the general healthcare demand for this product by patients and healthcare service providers.* Applying this protocol to pilot data collected by the company will result in valuable predictions quantifying the interrelations between costs and benefits, supply and demand, as well as consumer sentiment and health outcomes.

The DSPA materials (book chapters, code and scripts, data, case studies, electronic materials, and web demos) may be used as a reference or as a retraining or refresher guide. These resources may be useful for formal education and informal training, as well as, for health informatics, biomedical data analytics, biosocial computation courses, or MOOCs. Although the textbook is intended to be utilized for one, or two, semester-long graduate-level courses, readers, trainees and instructors should review the early sections of the textbook for utilization strategies and explore the suggested completion pathways.

As acknowledged in the front matter, this textbook relies on the enormous contributions and efforts by a broad community, including researchers, developers, students, clinicians, bioinformaticians, data scientists, *open-science* investigators, and funding organizations. The author strongly encourages all DSPA readers, educators, and practitioners to actively *contribute* to data science and predictive analytics, share data, algorithms, code, protocols, services, successes, failures, pipeline workflows, research findings, and learning modules. Corrections, suggestions for improvements, enhancements, and expansions of the DSPA materials are always welcome and may be incorporated in electronic updates, errata, and revised editions with appropriate credits.

Glossary

Table 1 Glossary of terms and abbreviations use in the textbook

Notation	Description
ADNI	Alzheimer's Disease Neuroimaging Initiative
AD	Alzheimer's Disease patients
Allometric relationship	Relationship of body size to shape, anatomy, physiology and behavior
ALS	Amyotrophic lateral sclerosis
API	Application program interface
Apriori	Apriori Association Rules Learning (Machine Learning) Algorithm
ARIMA	Time-series autoregressive integrated moving average model
array	Arrays are R data objects used to represent data in more than two dimensions
BD	Big Data
cor	correlation
CV	Cross Validation (an internal staistical validation of a prediction, classification or forecasting method)
DL	Deep Learning
DSPA	Data Science and Predictive Analytics
Eigen	Referring to the general Eigen-spectra, eigen-value, eigen-vector, eigen-function
FA	Factor analysis
GPU or CPU	Graphics or Central Processing Unit (computer chipset)
GUI	graphical user interface
HHMI	Howard Hughes Medical Institute
I/O	Input/Output
IDF	inverse document frequency
IoT	Internet of Things
JSON	JavaScript Object Notation
k-MC	k-Means Clustering

(continued)

Table 1 (continued)

Notation	Description
lm()	linear model
lowess	locally weighted scatterplot smoothing
LP or QP	linear or quadratic programming
MCI	mildly cognitively impaired patients
MIDAS	Michigan Institute for Data Science
ML	Machine-Learning
MOOC	massive open online course
MXNet	Deep Learning technique using R package MXNet
NAND	Negative- <i>AND</i> logical operator
NC or HC	Normal (or Healthy) control subjects
NGS	Next Generation Sequence (Analysis)
NLP	Natural Language Processing
OCR	optical character recognition
PCA	Principal Component Analysis
PD	Parkinson's Disease patients
PPMI	Parkinson's Progression Markers Initiative
(R)AWS	(Risk for) Alcohol Withdrawal Syndrome
RMSE	root-mean-square error
SEM	structural equation modeling
SOCR	Statistics Online Computational Resource
SQL	Structured Query Language (for database queries)
SVD	Singular value decomposition
SVM	Support Vector Machines
TM	Text Mining
TS	Time-series
w.r.t.	With Respect To, e.g., “ <i>Take the derivative of this expression w.r.t. a_1 and set the derivative to 0, which yields $(S - \lambda I_N)a_1 = 0$.</i> ”
XLSX	Microsoft Excel Open XML Format Spreadsheet file
XML	eXtensible Markup Language
XOR	Exclusive <i>OR</i> logical operator

Index

A

Accuracy, 10, 211, 275, 276, 283, 301–303, 307, 323–325, 334, 335, 337, 339, 340, 342, 343, 377, 409, 424, 432, 463, 475, 479–482, 484, 485, 497, 500, 502, 504, 507, 508, 511, 561, 562, 573, 576, 583, 599, 605, 692, 698, 704, 726, 767, 781, 782, 784, 793, 800, 801, 806
Activation, 383–385, 403, 767–769, 774, 775, 781, 785, 799, 800
Activation functions, 384, 385, 767, 781
add, 16, 22, 24, 33, 41, 146, 155, 158, 159, 162, 225, 227, 230, 292, 332, 373, 386, 391, 402, 403, 418, 424, 454, 479, 530, 538, 595, 605, 633, 645, 712, 801
Alcohol withdrawal syndrome (RAWS), 3, 824
Allometric, 266, 817, 823
Allometric relationship, 817
ALSFRS, 4, 559, 733, 783
Alzheimer’s disease (AD), 4, 149–151, 569, 823
Alzheimer’s disease neuroimaging initiative (ADNI), 4, 823
Amyotrophic lateral sclerosis (ALS), 4, 140, 141, 559–569, 733, 783–784, 823
Analog clock, 816
Appendix, 56–60, 138–139, 149, 183–197, 420
Application program interface (API), 525, 784, 823
Apriori, 267, 268, 423–427, 431, 441, 472, 823
ARIMA, 623, 626, 628, 630–638, 823

array, 20, 25, 31–33, 145

array (), 18

Assessment, 282–286, 510–511

Assessment: 22. deep learning, neural networks, 816–817

assocplot, 40

assocplot(x) Cohen’s Friendly graph shows the deviations from independence of rows and columns in a two dimensional contingency table, 40

attr, 27

Attributes, 26, 27, 144, 289, 311, 313, 315, 342, 530, 560, 561, 670

axes, 41, 46, 47, 131, 152, 154, 159, 171, 191, 219, 249, 258, 261, 368, 595, 648

axes=TRUE, 41

B

Bar, 15, 140, 143, 147, 159, 161, 162, 164

barplot, 39, 161, 162, 164, 463

barplot(x) histogram of the values of x. Use horiz=FALSE for horizontal bars, 39

Beach, 811–812

Big Data, 1, 4, 8–10, 12, 642, 661, 765, 819, 823

Biomedical, 8–9

Bivariate, 39, 40, 46, 77, 140, 153–156, 173, 238, 240, 252, 738–739, 766, 770

Black box, 383, 766

boxplot, 39, 70, 161

boxplot(x) ‘box-and-whiskers’ plot, 39

Brain, 4, 178, 286, 511, 769, 814–815

C

c(), 18–20, 552
 c(), seq(), rep(), and data.frame(). Sometimes we use list() and array() to create data too, 18
 C/C++, 13
 Cancer, 293, 294, 296, 298, 302, 303, 424, 427, 432
 Caret, 322, 477, 486, 487, 491, 492, 497–510, 554, 555, 564, 776
 Chapter, 13, 63, 69, 139, 143, 149, 164, 183, 201, 222, 245, 268, 271, 274, 289, 295, 298, 300, 301, 308, 317, 322, 329, 334, 336, 337, 342, 345–348, 353, 358, 361, 370, 373, 380, 383, 390, 392, 394, 398, 401, 409, 414–416, 420, 427, 442, 447–449, 465, 475–480, 488, 491, 492, 494, 527, 546, 553, 554, 557, 563, 564, 570, 573, 574, 585, 592, 599, 601, 623, 657, 659, 672, 674, 684, 689, 695, 697, 712, 713, 715, 717, 719, 720, 723, 727, 733, 735, 736, 738, 749, 753, 756, 763, 766, 795, 817
 Chapter 22, 415, 817
 Chapter 23, 164
 Chronic disease, 316, 330, 335, 383, 416, 476, 503
 Classification, 144, 267, 268, 281, 286–287, 289, 304–305, 307, 323, 331–332, 396–403, 477, 478, 498, 510, 533, 773–782, 795–805, 816
 Clinical, 258, 612, 614, 695
 Coast, 812
 Cognitive, 2, 4, 7, 149, 700, 820
 Color, 45, 46, 87, 132, 151, 154, 165, 167, 172, 269, 444, 649, 660
 confusionMatrix, 283, 322, 477, 480, 482, 485, 776, 787
 Constrained, 244, 587, 735, 740–747, 750
 Contingency table, 35, 40, 78, 500
 contour, 40
 contour(x, y, z) contour plot (data are interpolated to draw the curves), x and y must be vectors and z must be a matrix so that $\text{dim}(z) = c(\text{length}(x), \text{length}(y))$ (x and y may be omitted), 40
 coplot, 40
 coplot(x~y | z) bivariate plot of x and y for each value or interval of values of z, 40
 Coral, 815
 Cosine, 659, 685, 695
 Cosine similarity, 695

Cost function, 217, 503, 573, 586, 703, 735, 743, 747, 757, 758
 CPU, 553, 765, 775, 782, 800, 804, 805, 823
 Create, 19, 22, 76–78, 83, 132, 174, 202, 214, 222, 224, 273, 274, 299, 315, 318, 319, 370, 380, 383, 390, 450, 461, 489, 491, 504, 538, 607, 630, 638, 644, 645, 647, 661, 674, 688, 717, 775, 781
 Crossval, 776, 787
 Cross validation, 477, 599–601, 733–734, 823

D

Data frame, 19, 21, 22, 24, 28, 29, 31, 33–36, 39, 40, 47, 48, 66, 131, 132, 153, 164, 172, 174, 273, 274, 299, 300, 319, 438, 451, 490, 514, 526, 529, 537, 540, 547–549, 555, 561, 562, 565, 608
 data.frame, 19, 25, 83, 103, 164, 273
 Data science, 1, 9, 11, 661, 823, 824
 Data Science and Predictive Analytics (DSPA), 1, 11–13, 198, 492, 623, 661, 819–821, 823
 Decision tree, 307, 310–316, 498, 510, 533
 Deep learning, 765–768, 816–817, 823, 824
 classification, 816
 regression, 817
 Denoising, 735, 756, 757, 760, 763
 Density, 46, 48, 49, 72, 98, 132, 133, 140, 141, 143–147, 173, 174, 198, 287, 289
 Device, 775, 800
 diagnosticErrors, 718, 776
 Dichotomous, 40, 271, 318, 459, 460, 478, 655, 698, 733, 746, 747, 770
 Dimensionality reduction, 233, 265–266
 Divide-and-conquer, 307, 311, 373
 Divide and conquer classification, 307
 Divorce, 443, 448–455, 467, 470
 dotchart, 39
 dotchart(x) if x is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column), 39
 Download, 15, 555, 806, 817

E

Earthquake, 132–135, 157, 159, 172
 Ebola, 5
 Eigen, 219, 823
 Entropy, 311–313, 342

- Error, 28, 47, 57, 60, 162, 163, 217, 254, 258, 270, 280, 281, 287, 302, 305, 311, 313, 316, 321, 324–325, 328, 329, 331, 332, 350, 361, 378, 388, 391, 393, 412, 478–480, 487, 491, 500, 501, 504, 507, 509, 562, 565, 573, 576, 579, 582–584, 586, 587, 599, 618, 640, 645, 648, 697, 701–703, 712, 714, 725, 733, 734, 784, 824
- Evaluation, 268, 282, 322, 335, 361, 443, 451, 475, 477, 491, 492, 501, 504, 507, 510, 543, 546, 554, 697, 703, 817
- Exome, 6
- Expectations, 11–12
- Explanations, 41, 510
- F**
- Face, 815–816
- Factor, 21, 24, 46, 79, 210, 219, 233, 255, 256, 259, 265, 287, 292, 294, 299, 319, 333, 352, 359, 412, 417, 438, 561, 570, 575, 588, 600, 608, 630, 638–640, 644, 676, 677, 703, 725
- Factor analysis (FA), 233, 242, 243, 254–256, 262, 265, 638, 639, 644, 823
- False-negative, 700
- False-positive, 325, 573, 574, 619
- Feature selection, 557–559, 571–572
- Feedforward neural net, 817
- filled.contour(x, y, z) areas between the contours are colored, and a legend of the colors is drawn as well, 40
- Flowers, 39, 63, 309, 383, 410, 411, 414, 510
- Format, 13, 17–18, 22, 36, 38, 427, 513–515, 522, 524, 525, 529, 537, 553, 665, 799, 801, 805
- Foundations, 13, 638–641
- fourfoldplot(x) visualizes, with quarters of circles, the association between two dichotomous variables for different populations (x must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=(2, 2)` if `k = 1`), 40
- Frequencies, 29, 39, 46, 145, 193, 298, 429, 430, 439, 463, 484, 485, 667, 672, 685
- Function, 2, 4, 16, 20, 22, 28, 30, 32–35, 37, 47–50, 57–60, 66, 68–70, 76–78, 83, 131–133, 143, 145, 148, 149, 151, 153, 155, 157, 161, 162, 167, 172–175, 187, 202, 207, 208, 213, 216–219, 222, 224, 225, 234, 243, 246, 247, 251, 254, 255, 257, 260, 267, 269, 272–274, 289, 295, 299, 300, 308, 313, 314, 317, 319, 322, 323, 332, 334, 337, 351, 352, 356, 358, 361, 370, 375, 376, 378, 383–385, 390–392, 394–397, 401–403, 411–413, 427, 428, 432, 434, 438, 449–451, 455, 470, 475, 479, 480, 483, 490, 494, 499–501, 504–506, 508, 509, 514, 524, 526, 530, 532, 542, 547–554, 560, 561, 563, 569, 575, 579, 582, 586, 595, 600, 602, 607, 616, 625, 631, 632, 634, 637, 640, 644, 645, 649, 655, 660, 664–667, 673–676, 688, 702, 709, 713, 714, 716, 717, 735–741, 748, 749, 753, 767–770, 772, 774–776, 781, 782, 785, 799–801, 808, 823
- Functional magnetic resonance imaging (fMRI), 178–181, 623, 657
- Function optimization, 243, 735, 761–763
- G**
- Gaussian mixture modeling, 443
- Generalized estimating equations (GEE), 653–657
- Geyser, 174, 175, 813
- ggplot2, 14, 16, 131, 132, 157, 164, 172, 455, 648
- Gini, 311, 313, 335, 336, 342
- Glossary, 823
- Google, 383, 388–394, 396–398, 416, 491, 492, 494, 658, 697–700, 773, 784, 817
- GPU, 513, 553, 765, 775, 782, 804, 805, 823
- Graph, 14, 40, 47, 70, 75, 77, 164, 166, 198, 244, 287, 297, 305, 356, 376, 386, 391, 393, 399, 430, 431, 443, 448, 489, 528–533, 555, 562, 563, 570, 613, 626, 628, 649, 650, 658, 676, 775, 784
- Graphical user interfaces (GUIs), 15–16, 823
- H**
- Handwritten digits, 795, 799, 801
- HC, 135, 705, 824
- Heatmap, 134, 150–152
- Help, 16
- Heterogeneity, 11, 311
- Hidden, 135, 386, 391, 393, 394, 398, 416, 660, 765–767, 772, 774, 775, 781, 785, 799
- Hierarchical clustering, 443, 467–469, 727
- High-throughput big data analytics, 10
- hist, 39, 83, 144
- hist(x) histogram of the frequencies of x, 39

Histogram, 39, 46, 51, 68, 71–74, 87, 140, 143, 144, 146, 174, 180, 198, 222, 249, 250, 353, 356, 634, 792

Horizontal, 39, 45, 46, 70, 151, 152, 159, 230, 356, 368

Hospital, 346, 347, 513, 655, 656

Howard Hughes Medical Institute (HHMI), 5, 823

I

IBS, 789–792

if TRUE superposes the plot on the previous one (if it exists), 41

Image, 20, 24, 40, 83, 84, 176–178, 403, 404, 660, 781, 795, 796, 799, 801, 806–816

Image classification, 817

image(x, y, z) plotting actual data with colors, 40

Independent component analysis (ICA), 233, 242, 243, 250–254, 265

Index, 187, 313, 316, 388, 389, 392, 416, 513, 625, 641

Inference, 1, 13, 201, 282, 289, 513, 573, 638, 655, 659, 735, 819

Input/output (I/O), 22–24, 64, 765, 823

interaction.plot (f1, f2, y) if f1 and f2 are factors, plots the means of y (on the y-axis) with respect to the values of f1 (on the x-axis) and of f2 (different curves). The option fun allows to choose the summary statistic of y (by default fun=mean), 40

Interpolate, 48

Intersect, 30

Inverse document frequency (IDF), 659, 676–686, 695, 823

Iris, 63, 64, 308–310, 409–411, 414, 727

J

Java, 10, 13, 20, 72, 332, 334, 349, 534

Jitter, 143, 157

JSON, 198, 513, 514, 522, 525–526, 531, 533, 823

K

k-Means Clustering (k-MC), 443

k-nearest neighbor (kNN), 268, 269, 447

Knockoff, 574, 621

L

Lagrange, 401, 402, 735, 740–741, 749, 753–756, 762

Lake Mapourika, 810–811

Lattice, 46, 47

Layer, 386, 388, 394, 765–768, 770, 771, 773–775, 781, 782, 785, 799–801

Lazy learning, 267, 286–287

Length, 5, 19, 21, 26, 28, 35, 37, 40, 46, 47, 63, 64, 132, 174, 230, 231, 235, 270, 273, 346, 374, 377, 409, 480

Letters, 148, 193, 195, 215, 404, 530, 664

Linear algebra, 201, 229–231, 345

Linear mixed models, 623

Linear model, 574–582, 621, 650

Linear programming, 735, 748

list (), 18–20

lm (), 16, 225, 358, 553, 824

log, 30, 31, 40, 313, 517, 587, 610, 611, 615, 616, 640, 716

Log-linear, 40

Long, 5, 13, 18, 36, 514, 547, 565, 676, 784, 819

Longitudinal data, 40, 657–658

Lowess, 824

M

Machine learning, 2, 10, 267, 268, 289, 322, 383, 423, 443–444, 476, 477, 481, 497, 536, 549, 562, 659, 660, 667, 689, 765, 809, 816, 820

Managing data, 63, 140–141

Mask, 815–816

matplot(x, y) bivariate plot of the first column of x vs. the first one of y, the second one of x vs. the second one of y, etc, 40

Matrices, 20, 21, 24, 31, 149, 167, 201–203, 206–209, 213–216, 219, 220, 222, 229, 230, 233, 258, 478–480, 490, 549, 574, 640, 641, 645, 650, 667, 672, 698, 714, 716, 735, 782, 804

Matrix, 13, 21, 26, 28, 31, 32, 40, 46, 47, 81, 132, 149–151, 153, 161–163, 166, 167, 174, 201–209, 211, 212, 214–217, 219–222, 224, 225, 227–231, 235, 236, 238–240, 242, 244, 245, 247, 251, 254–258, 260, 265, 295, 299, 300, 304, 305, 319, 322, 324–325, 350, 351, 356, 391, 427, 429–432, 450, 463, 478, 480, 483, 484, 501, 506, 507, 528–530, 537, 540, 552, 555, 574, 582, 607, 608, 620, 639–641, 648, 650, 654, 655, 660,

- 667–668, 670–674, 676, 685, 688, 689, 695, 702, 716, 717, 727, 735, 739, 747, 748, 753, 766, 767, 782, 799–801
- Matrix computing, 201, 229–231, 345
- Michigan Institute for Data Science (MIDAS), 824
- Mild cognitive impairment (MCI), 4, 149, 151, 824
- Misclassification, 311, 324–325, 411, 418
- mlbench, 536, 774
- mlp, 774, 775, 781, 782, 785
- Model, 2, 10, 13, 47–48, 81, 93, 110, 120, 166, 201, 216, 217, 227, 230, 246, 252, 253, 260, 262, 267, 268, 274–276, 283, 286, 299–301, 345, 350, 356, 358–375, 377–381, 383, 385–387, 391–394, 397, 398, 405–409, 411–416, 418, 479, 488, 489, 510, 511, 571, 572, 658, 733, 734, 817
- Model performance, 268, 274–276, 300–301, 322–323, 333, 359–373, 377–380, 386, 392–394, 406–409, 412–414, 433–438, 451–454, 462–465, 475, 479, 480, 487, 488, 491, 492, 494, 495, 497, 501–503, 507, 564–569, 572, 605, 697, 698, 701
- Model-based, 2, 10, 345, 481, 566, 573, 660, 710, 819, 821
- Model-free, 2, 10, 481, 660, 689, 705, 819, 821
- Modeling, 1, 4, 9, 13, 48, 83, 201, 216–217, 233, 259, 307, 347, 349, 505, 513, 528, 582, 638, 640, 659, 668, 701, 703, 756, 775, 820, 824
- MOOCs, 821
- mosaicplot, 40
- mosaicplot(x) “mosaic” graph of the residuals from a log-linear regression of a contingency table, 40
- Multi-scale, 623
- Multi-source, 9, 514, 559
- MXNet, 774, 775, 782, 785, 799–801, 804, 805, 817
- N**
- NA, 22, 24, 28, 30, 38, 67, 69, 155, 287, 380, 427, 429, 538, 625
- na.omit, 28, 48
- na.omit(x), 28
- Naive Bayes, 289, 290, 299, 302–305, 476
- Natural language processing (NLP), 442, 659–668, 689–691, 694–695, 824
- Nearest neighbors, 267, 286–287, 719–720
- Negative AND (NAND), 771–772, 824
- Network, 383, 384, 386, 398, 533, 555, 730, 731, 773, 799–800, 804–806
- Neural networks, 383–388, 498, 510, 717–718, 765, 766, 816–817
- Neurodegeneration, 4–5
- Neuroimaging, 4, 7, 588–590, 608–621, 789, 817, 823
- New Zealand, 810–811
- Next Generation Sequence (NGS), 6–7, 824
- Next Generation Sequence (NGS) Analysis, 6–7
- Nodes, 164, 293, 307, 311, 316, 321, 336, 374, 376, 379, 383, 386, 391, 393, 394, 416, 524, 528–530, 532, 765, 766, 768, 775, 785
- Non-linear optimization, 752–753, 762
- Normal controls (NC), 4, 149, 151, 152, 167, 169–171, 824
- Numeric, 2, 19, 25, 46, 47, 66, 68, 71, 76, 77, 145, 149, 150, 212, 259, 273, 274, 299, 319, 370–371, 377, 396, 409, 503, 559, 570
- O**
- Objective function, 242, 250, 251, 401, 558, 573, 574, 579, 587, 592, 640, 641, 735–738, 740, 741, 747–749, 753, 754, 756–758
- Open-science, 1, 819, 821
- Optical character recognition (OCR), 383, 403–408, 795, 824
- Optimization, 13, 47, 243, 254, 401, 402, 513, 546, 573, 574, 579, 587, 592, 641, 735–753, 755, 756, 761, 762
- Optimize, 47, 337, 401, 739, 757, 758, 819
- P**
- Package, 30, 38, 46, 63, 78, 81, 131, 132, 138, 149, 157, 164, 167, 172, 174, 208, 247, 274, 294, 297, 299, 319, 322, 332, 356, 358, 375, 376, 378, 391, 410, 412, 413, 428, 434, 455, 467, 470, 483, 486–489, 491–493, 497, 501, 503, 505–507, 509, 514, 515, 517, 522–524, 526, 528, 531, 532, 534, 536, 547–555, 559–561, 588, 607, 608, 626, 627, 632, 641, 645, 648, 650, 661, 663, 666, 668, 672, 675, 686, 705, 723, 727, 741, 748, 754, 760, 765, 776, 804, 806, 824

- pairs, 5, 40, 45, 153–156, 164, 191, 234, 237, 239, 311, 356, 357, 371, 424–427, 529, 532, 770, 796
- pairs(x) if x is a matrix or a data frame, draws all possible bivariate plots between the columns of x, 40
- Parallel computing, 548–553, 555–556
- Parkinson's disease (PD), 51, 135, 261, 262, 265, 511, 571, 600, 608–621, 642–647, 650, 705, 711, 824
- Parkinson's Progression Markers Initiative (PPMI), 245, 286, 511, 571–572, 588, 642, 656, 705, 711, 719, 824
- Perceptron, 766, 769, 773, 775, 785
- Perl, 13
- persp(x, y, z) plotting actual data in perspective view, 40
- Petal, 39, 64, 727
- Pie, 39, 143, 147, 149, 167, 170, 198
- pie(x) circular pie-chart., 39
- Pipeline environment, 10
- Plot, 39–47, 66, 70, 71, 73, 74, 77, 84, 98, 131–133, 136, 140, 141, 143, 145–148, 150, 153–160, 162–164, 166–177, 180, 188, 191–194, 226, 230, 231, 233, 235, 239–241, 243, 247, 249, 250, 256, 262, 285, 286, 298, 323, 325, 326, 331, 346, 347, 353, 356, 357, 359, 361, 366, 368, 375–377, 414, 430, 431, 436, 439, 448, 452, 454, 456, 459, 467, 469, 471, 472, 532, 534, 535, 537, 539, 562, 563, 565, 570, 571, 590, 592, 597, 602, 618, 626, 629–631, 727, 736, 739, 742, 757, 768, 776, 778, 779, 792
- plot(x) plot of the values of x (on the y-axis) ordered on the x-axis, 39
- plot(x, y) bivariate plot of x (on the x-axis) and y (on the y-axis), 39
- plot.ts(x) if x is an object of class "ts", plot of x with respect to time, x may be multivariate but the series must have the same frequency and dates. Detailed examples are in Chap. 19
- big longitudinal data analysis, 40
- Predict, 3, 4, 9, 10, 48, 81, 267, 283, 300, 322, 334, 346, 377–380, 389, 391, 392, 411, 412, 475, 476, 478, 500, 504, 505, 555, 582, 584–586, 600, 602, 623, 674, 679, 700, 703, 712, 714, 717, 783, 792, 796, 806, 808, 817
- Predictive analytics (PA), 1, 9–10, 661, 823
- Principal component analysis (PCA), 233, 241–249, 254, 256–258, 260, 263, 265, 266, 533, 824
- Probabilistic learning, 304–305
- Probabilities, 31, 173, 300, 322, 476, 482, 500, 600, 684, 715, 716, 767, 800, 801
- Pruning, 307, 315, 316, 328, 330
- Python, 13
- Python, Java, C/C++, Perl, and many others, 13
- Q**
- QOL, 317
- qqnorm, 40
- qqnorm(x) quantiles of x with respect to the values expected under a normal law, 40
- qqplot, 40
- qqplot(x, y) quantiles of y with respect to the quantiles of x, 40
- Quadratic programming, 824
- Quality of life, 490, 792
- R**
- R, 1, 12–17, 20, 24, 25, 30–32, 37–39, 41, 46, 48–50, 56–60, 63–65, 67–69, 75–78, 84, 130, 131, 138–141, 143, 144, 149, 153, 161, 173, 175, 176, 178, 206, 208, 209, 211, 212, 214, 216, 219, 220, 224–227, 229, 230, 236, 240, 245, 246, 251, 254, 257, 258, 274, 294, 297, 307, 319, 332, 334, 349, 355, 356, 361, 374, 375, 378, 389, 409, 410, 420, 427, 428, 436, 438, 450, 460, 467, 470, 477, 479, 486, 488, 491, 501, 514, 515, 517, 524, 526–529, 532–534, 538, 540, 546–548, 550, 553, 559, 588, 617, 619, 624, 629, 641, 642, 650, 652, 659, 661, 694, 704, 714, 736–738, 748, 753, 754, 760, 765, 774, 782, 801, 806, 807, 820, 821, 823, 824
- Regularized, 574–582, 621
- Regularized linear model, 621
- Relationship, 77, 78, 155, 226, 245, 314, 345, 349, 350, 369, 383, 394, 448, 454, 532, 581, 584, 640, 817, 823
- rep(), 18
- Require, 12, 400, 409, 432, 527, 550, 555, 563, 772, 815, 819
- reshape2, 14, 16
- Risk for Alcohol Withdrawal Syndrome (RAWS), 3

- Root mean square error (RMSE), 329, 477, 565, 698, 701, 703, 784, 817, 824
RStudio, 13, 15–16
RStudio GUI, 15–16
- S**
Scatter plot
 scatter, 46, 153, 226, 230, 231
Sensitivity, 305, 485, 486, 714, 734, 776
seq (), 18, 38, 69
Sequencing, 6
set.seed, 37, 49, 287, 333, 492, 499, 782, 800
setdiff, 30
setequal, 30
Silhouette, 443, 446, 451, 452, 456–459, 463, 464, 469, 477, 723, 725
sin, cos, tan, asin, acos, atan, atan2, log, log10, exp and “set” functions union(x, y), intersect(x, y), setdiff(x, y), setequal (x, y), is.element(el, set) are available in R, 30
Singular value decomposition (SVD), 233, 241, 242, 256–258, 265, 824
Size, 16, 30, 46, 47, 49, 132, 135, 145, 154, 174, 192, 209, 210, 269, 315, 316, 323, 328, 336, 348, 390, 425, 426, 429, 450, 451, 495, 498, 500, 503, 510, 515, 534–536, 565, 566, 572, 592, 624, 676, 747, 767, 773, 774, 781, 784, 795, 819, 823
sMRI, 4, 178
softmax, 498, 767, 774, 781, 800
Sonar, 774–781
Sort, 28, 700
Specificity, 305, 485–486, 734, 776
Spectra, 231
Splitting, 268, 307, 311, 315, 373, 374, 536, 584, 686
SQL, 138–139, 513, 515–521, 537, 553, 824
Stacked, 39, 46, 196
stars(x) if x is a matrix or a data frame, draws a graph with segments or a star where each row of x is represented by a star and the columns are the lengths of the segments, 40
Statistics Online Computational Resource (SOCR), 4, 10, 11, 50, 51, 56, 72, 79, 130, 140, 147, 171, 173, 178, 187, 193, 198, 230, 258, 305, 342, 349, 522, 524, 525, 531–533, 540–543, 555, 569, 584, 669, 817, 824
stripplot, 39, 46
stripplot(x) plot of the values of x on a line (an alternative to boxplot() for small sample sizes), 39
Structural equation modeling (SEM), 623, 638–648, 824
Summary statistic, 35, 40, 67, 76, 140, 187, 352, 549
sunflowerplot(x, y) id. than plot() but the points with similar coordinates are drawn as flowers which petal number represents the number of points, 39
Support vector machines (SVM), 398–403
Surface, 132, 141, 174–176, 814–815
Symbol, 47, 404, 490, 781, 799
symbols(x, y, ...) draws, at the coordinates given by x and y, symbols (circles, squares, rectangles, stars, thermometers or “boxplots”) which sizes, colors... are specified by supplementary arguments, 40
- T**
Table, 13, 22–24, 29, 30, 32, 35, 38, 40, 76, 78, 79, 140, 144, 148, 166, 208, 268, 274, 275, 282, 292, 300, 301, 311, 317, 322, 412, 426, 450, 463, 477–480, 482, 483, 486, 501, 504, 511, 529, 530, 548, 555, 614, 641, 686, 771
TensorFlow, 765, 773, 784
Term frequency (TF), 659, 676–686, 695
termplot(mod.obj) plot of the (partial) effects of a regression model (mod.obj), 40
Testing, 7, 268, 274, 282, 287, 299, 303, 318, 324, 342, 396, 414, 491, 505, 579, 581, 584, 599, 600, 639, 648, 679, 684, 686, 690, 691, 697, 701, 703, 704, 719, 765, 775, 782, 784, 795, 799–801
Text mining (TM), 442, 659–668, 689–691, 694–695, 824
The following parameters are common to many plotting functions, 40
Then, try to perform a multiple classes (i.e AD, NC and MCI) classification and report the results, 816
Training, 8, 141, 260, 267–270, 274, 281, 287, 289, 292, 295–297, 299–300, 303, 304, 311, 318–321, 332–333, 337, 358–359, 374, 375, 380, 390, 391, 395, 396, 398, 410–412, 416, 418, 432–433, 450–451, 461, 491, 493, 495, 501, 503–505, 507, 553, 554, 558–564, 579, 584, 599, 600, 679, 684, 686, 688, 697, 701–704, 715,

- 719, 733, 765, 768, 769, 775, 776, 778, 782, 784, 795, 796, 799, 800, 804, 805, 815, 819, 821
- Transdisciplinary, 9, 819, 820
- Trauma, 163, 443, 459–467
- ts, 1, 31, 40, 47, 77, 80, 533, 629, 631
- ts.plot(x) id. but if x is multivariate the series may have different dates and must have the same frequency, 40
- U**
- Unconstrained, 735–741, 761
- Union, 30, 145, 424
- Unique, 1, 7, 29, 144, 150, 210, 334, 429, 463, 495, 527, 639, 667, 688, 698, 736, 774, 820
- Unstructured text, 289, 659, 660, 795
- V**
- Validation, 9, 267, 280, 281, 283, 287, 329, 340, 396, 414, 446, 448, 475, 477, 491–495, 501, 562, 574, 581, 598–600, 675, 679, 689–691, 697, 698, 700–704, 708, 709, 711, 712, 718, 733, 765, 784, 796, 799, 815, 819, 820, 823
- Visualization, 4, 143–145, 164, 198–199, 657
- Visualize, 4, 70, 77, 132, 149, 173, 178, 201, 222, 226, 247, 249, 252, 256, 280, 297, 305, 323, 356, 429, 434, 453, 528, 565, 571, 590, 592, 626, 648, 726
- Volcano, 175, 812–813
- W**
- which.max, 27
- Whole-genome, 6
- Whole-genome and exome sequencing, 6
- Wide, 13, 18, 36, 48, 381, 427, 514, 583, 656, 820
- With, 1, 2, 4, 5, 8, 9, 12, 16–20, 22–26, 28–31, 33, 36, 37, 39–41, 45–50, 57, 67, 69–71, 77, 80, 81, 83, 84, 130, 132, 133, 135, 139, 140, 143, 147, 149, 150, 153, 155–159, 161–163, 166, 171, 173–175, 202, 210, 212–216, 219, 220, 222, 230, 231, 233, 237, 242–244, 254–256, 258, 267, 269–271, 273, 289, 295, 299, 302–305, 307, 310–319, 322, 324, 325, 327, 331–334, 336, 337, 339, 340, 342, 347, 371, 374, 377–380, 385–390, 394, 398–403, 408, 411, 412, 416, 420, 423–425, 427, 429, 430, 432, 435, 438, 439, 441, 442, 444–446, 448–451, 453–460, 463–466, 469, 472, 475, 476, 484, 495, 497, 500, 502–506, 508, 511, 513–536, 540, 543, 546–559, 563–570, 572–574, 576, 584, 586, 599, 605–608, 612–614, 616, 617, 620, 625, 626, 628, 630, 634, 637–639, 642, 643, 648, 653, 655, 656, 663, 665, 668, 670, 672, 674, 676–678, 684, 686, 688, 689, 698, 700, 704, 710, 711, 715, 717, 718, 720, 734, 736, 746, 756, 769–774, 781, 784, 785, 800, 805, 808, 812, 815, 817, 819, 820
- X**
- XLSX, 526–527, 824
- XML, 7, 24, 513, 522–524, 555, 824
- Exclusive OR (XOR), 770, 771, 824
- Y**
- Youth, 271, 443, 465–467, 498