Bater Makhabel, Pradeepta Mishra,
Nathan Danneman, Richard Heimann

R: Mining Spatial, Text, Web, and Social Media Data

Learning Path

Create and customize data mining algorithms

Packt

R: Mining Spatial, Text, Web, and Social Media Data

Create and customize data mining algorithms

A course in three modules

Packt

BIRMINGHAM - MUMBAI

R: Mining Spatial, Text, Web, and Social Media Data

Copyright © 2016 Packt Publishing

All rights reserved. No part of this course may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this course to ensure the accuracy of the information presented. However, the information contained in this course is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this course.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this course by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Published on: April 2017

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78829-374-7

www.packtpub.com

Credits

Authors

Bater Makhabel

Pradeepta Mishra

Nathan Danneman

Richard Heimann

Reviewers

Jason H.D. Cho

Gururaghav Gopal

Vibhav Kamath

Hasan Kurban

Alexey Grigorev

Carlos J. Gil Bellosta

Vibhav Vivek Kamath

Feng Mai

Ajay Ohri

Yanchang Zhao

Content Development Editor

Trusha Shriyan

Graphics

Jason Monterio

Production Coordinator

Melwyn Dsa

Preface

The necessity to handle many, complex statistical analysis projects is hitting statisticians and analysts across the globe. With increasing interest in data analysis, R offers a free and open source environment that is perfect for both learning and deploying predictive modeling solutions in the real world. With its constantly growing community and plethora of packages, R offers functionality for dealing with a truly vast array of problems.

It's been decades since the R programming language was born and has become an eminent and well known not only within the community of scientist, but also in the wider community of developers. It has grown into a powerful tool to help developers produce efficient and consistent source code for data related tasks. The R development team and independent contributors have created good documentation so getting started programming with R isn't that hard.

What this learning path covers

Module 1, Learning Data mining with R, will teach you how to manipulate data with R using code snippets and be introduced to mining frequent patterns, association, and correlations while working with R programs. You will discover how to write code for various predication models, stream data, and time-series data. You will also be introduced to solutions written in R based on RHadoop projects. You will finish this module by feeling confident in your ability to know which data mining algorithm to apply in any situation.

Module 2, R Data Mining Blueprints, explores data mining techniques and shows you how to apply different mining concepts to various statistical and data applications in a wide range of fields. You will learn about R and its application to data mining, and give you relevant and useful information you can use to develop and improve your applications. This module will help you complete complex data mining cases and guide you through handling issues you might encounter during projects.

Module 3, Social Media Mining with R, begins by introducing you to the topic of social media data, including its sources and properties. It then explains the basics of R programming in a straightforward, unassuming way. Thereafter, you will be made aware of the inferential dangers associated with social media data and how to avoid them, before describing and implementing a suite of social media mining techniques.

What you need for this learning path

Any modern PC with installed Windows, Linux or Mac OS should be sufficient to run the code samples in the book. All the software used in the book is open source and freely available on the Web:

`http://www.r-project.org/`

Who this learning path is for

This learning path is for R developers who are looking up to making a career in data analysis or data mining. Those who come across data mining problems of different complexities from web, text, numerical, political, and social media domains will find all information in this single learning path.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at `http://www.packtpub.com`. If you purchased this course elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at `https://github.com/PacktPublishing/R-Mining-spatial-text-web-and-social-media-data/`. We also have other code bundles from our rich catalog of books, videos, and courses available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

Module 1: Learning Data mining with R

Chapter 1: Warming Up 1

Big data 2
Data source 4
Data mining 4
Social network mining 8
Text mining 11
Web data mining 12
Why R? 14
Statistics 14
Machine learning 16
Data attributes and description 17
Data cleaning 21
Data integration 23
Data dimension reduction 23
Data transformation and discretization 25
Visualization of results 27
Time for action 28
Summary 29

Chapter 2: Mining Frequent Patterns, Associations, and Correlations 31

An overview of associations and patterns 32
Market basket analysis 38
Hybrid association rules mining 58
Mining sequence dataset 59
The R implementation 62
High-performance algorithms 65
Time for action 65
Summary 66

Chapter 3: Classification 67
Classification 68
Generic decision tree induction 70
High-value credit card customers classification using ID3 76
Web spam detection using C4.5 82
Web key resource page judgment using CART 90
Trojan traffic identification method and Bayes classification 93
Identify spam e-mail and Naïve Bayes classification 98
Rule-based classification of player types in computer games and rule-based classification 102
Time for action 108
Summary 109

Chapter 4: Advanced Classification 111
Ensemble (EM) methods 111
Biological traits and the Bayesian belief network 118
Protein classification and the k-Nearest Neighbors algorithm 120
Document retrieval and Support Vector Machine 121
Classification using frequent patterns 128
Classification using the backpropagation algorithm 131
Time for action 137
Summary 137

Chapter 5: Cluster Analysis 139
Search engines and the k-means algorithm 142
Automatic abstraction of document texts and the k-medoids algorithm 150
The CLARA algorithm 153
CLARANS 155
Unsupervised image categorization and affinity propagation clustering 156
News categorization and hierarchical clustering 160
Time for action 166
Summary 166

Chapter 6: Advanced Cluster Analysis 169
Customer categorization analysis of e-commerce and DBSCAN 169
Clustering web pages and OPTICS 172
Visitor analysis in the browser cache and DENCLUE 175
Recommendation system and STING 180
Web sentiment analysis and CLIQUE 181
Opinion mining and WAVE clustering 183
User search intent and the EM algorithm 186
Customer purchase data analysis and clustering high-dimensional data 188
SNS and clustering graph and network data 191

Time for action 193
Summary 193

Chapter 7: Outlier Detection 195
Credit card fraud detection and statistical methods 196
Activity monitoring – the detection of fraud involving mobile phones and proximity-based methods 199
Intrusion detection and density-based methods 205
Intrusion detection and clustering-based methods 210
Monitoring the performance of the web server and classification-based methods 212
Detecting novelty in text, topic detection, and mining contextual outliers 214
Collective outliers on spatial data 217
Outlier detection in high-dimensional data 219
Time for action 221
Summary 221

Chapter 8: Mining Stream, Time-series, and Sequence Data 223
The credit card transaction flow and STREAM algorithm 224
Predicting future prices and time-series analysis 227
Stock market data and time-series clustering and classification 230
Web click streams and mining symbolic sequences 233
Mining sequence patterns in transactional databases 237
Time for action 238
Summary 239

Chapter 9: Graph Mining and Network Analysis 241
Graph mining 241
Mining frequent subgraph patterns 242
Social network mining 246
Time for action 249
Summary 249

Chapter 10: Mining Text and Web Data 251
Text mining and TM packages 252
Text summarization 252
The question answering system 257
Genre categorization of web pages 258
Categorizing newspaper articles and newswires into topics 259
Web usage mining with web logs 262
Time for action 265
Summary 265

Appendix: Algorithms and Data Structures 267

Module 1

Learning Data mining with R

Develop key skills and techniques with R to create and customize data mining algorithms

1
Warming Up

In this chapter, you will learn basic data mining terms such as data definition, preprocessing, and so on.

The most important data mining algorithms will be illustrated with R to help you grasp the principles quickly, including but not limited to, classification, clustering, and outlier detection. Before diving right into data mining, let's have a look at the topics we'll cover:

- Data mining
- Social network mining
- Text mining
- Web data mining
- Why R
- Statistics
- Machine learning
- Data attributes and description
- Data measuring
- Data cleaning
- Data integration
- Data reduction
- Data transformation and discretization
- Visualization of results

In the history of humankind, the results of data from every aspect is extensive, for example websites, social networks by user's e-mail or name or account, search terms, locations on map, companies, IP addresses, books, films, music, and products.

Data mining techniques can be applied to any kind of old or emerging data; each data type can be best dealt with using certain, but not all, techniques. In other words, the data mining techniques are constrained by data type, size of the dataset, context of the tasks applied, and so on. Every dataset has its own appropriate data mining solutions.

New data mining techniques always need to be researched along with new data types once the old techniques cannot be applied to it or if the new data type cannot be transformed onto the traditional data types. The evolution of stream mining algorithms applied to Twitter's huge source set is one typical example. The graph mining algorithms developed for social networks is another example.

The most popular and basic forms of data are from databases, data warehouses, ordered/sequence data, graph data, text data, and so on. In other words, they are federated data, high dimensional data, longitudinal data, streaming data, web data, numeric, categorical, or text data.

Big data

Big data is large amount of data that does not fit in the memory of a single machine. In other words, the size of data itself becomes a part of the issue when studying it. Besides volume, two other major characteristics of big data are variety and velocity; these are the famous three Vs of big data. Velocity means data process rate or how fast the data is being processed. Variety denotes various data source types. Noises arise more frequently in big data source sets and affect the mining results, which require efficient data preprocessing algorithms.

As a result, distributed filesystems are used as tools for successful implementation of parallel algorithms on large amounts of data; it is a certainty that we will get even more data with each passing second. Data analytics and visualization techniques are the primary factors of the data mining tasks related to massive data. The characteristics of massive data appeal to many new data mining technique-related platforms, one of which is RHadoop. We'll be describing this in a later section.

Some data types that are important to big data are as follows:

- The data from the camera video, which includes more metadata for analysis to expedite crime investigations, enhanced retail analysis, military intelligence, and so on.
- The second data type is from embedded sensors, such as medical sensors, to monitor any potential outbreaks of virus.

- The third data type is from entertainment, information freely published through social media by anyone.
- The last data type is consumer images, aggregated from social medias, and tagging on these like images are important.

Here is a table illustrating the history of data size growth. It shows that information will be more than double every two years, changing the way researchers or companies manage and extract value through data mining techniques from data, revealing new data mining studies.

Year	Data Sizes	Comments
N/A		1 MB (Megabyte) = 2^{20}. The human brain holds about 200 MB of information.
N/A		1 PB (Petabyte) = 2^{50}. It is similar to the size of 3 years' observation data for Earth by NASA and is equivalent of 70.8 times the books in America's Library of Congress.
1999	1 EB	1 EB (Exabyte) = 2^{60}. The world produced 1.5 EB of unique information.
2007	281 EB	The world produced about 281 Exabyte of unique information.
2011	1.8 ZB	1 ZB (Zetabyte)= 2^{70}. This is all data gathered by human beings in 2011.
Very soon		1 YB(Yottabytes)= 2^{80}.

Scalability and efficiency

Efficiency, scalability, performance, optimization, and the ability to perform in real time are important issues for almost any algorithms, and it is the same for data mining. There are always necessary metrics or benchmark factors of data mining algorithms.

As the amount of data continues to grow, keeping data mining algorithms effective and scalable is necessary to effectively extract information from massive datasets in many data repositories or data streams.

The storage of data from a single machine to wide distribution, the huge size of many datasets, and the computational complexity of the data mining methods are all factors that drive the development of parallel and distributed data-intensive mining algorithms.

Data source

Data serves as the input for the data mining system and data repositories are important. In an enterprise environment, database and logfiles are common sources. In web data mining, web pages are the source of data. The data that continuously fetched various sensors are also a typical data source.

Here are some free online data sources particularly helpful to learn about data mining:

- **Frequent Itemset Mining Dataset Repository**: A repository with datasets for methods to find frequent itemsets (`http://fimi.ua.ac.be/data/`).
- **UCI Machine Learning Repository**: This is a collection of dataset, suitable for classification tasks (`http://archive.ics.uci.edu/ml/`).
- **The Data and Story Library at statlib**: DASL (pronounced "dazzle") is an online library of data files and stories that illustrate the use of basic statistics methods. We hope to provide data from a wide variety of topics so that statistics teachers can find real-world examples that will be interesting to their students. Use DASL's powerful search engine to locate the story or data file of interest. (`http://lib.stat.cmu.edu/DASL/`)
- **WordNet**: This is a lexical database for English (`http://wordnet.princeton.edu`)

Data mining

Data mining is the discovery of a *model* in data; it's also called exploratory data analysis, and discovers useful, valid, unexpected, and understandable knowledge from the data. Some goals are shared with other sciences, such as statistics, artificial intelligence, machine learning, and pattern recognition. Data mining has been frequently treated as an algorithmic problem in most cases. Clustering, classification, association rule learning, anomaly detection, regression, and summarization are all part of the tasks belonging to data mining.

The data mining methods can be summarized into two main categories of data mining problems: feature extraction and summarization.

Feature extraction

This is to extract the most prominent features of the data and ignore the rest. Here are some examples:

- **Frequent itemsets**: This model makes sense for data that consists of *baskets* of small sets of items.
- **Similar items**: Sometimes your data looks like a collection of sets and the objective is to find pairs of sets that have a relatively large fraction of their elements in common. It's a fundamental problem of data mining.

Summarization

The target is to summarize the dataset succinctly and approximately, such as clustering, which is the process of examining a collection of *points* (data) and grouping the points into *clusters* according to some measure. The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another.

The data mining process

There are two popular processes to define the data mining process in different perspectives, and the more widely adopted one is CRISP-DM:

- **Cross-Industry Standard Process for Data Mining** (**CRISP-DM**)
- **Sample, Explore, Modify, Model, Assess** (**SEMMA**), which was developed by the SAS Institute, USA

CRISP-DM

There are six phases in this process that are shown in the following figure; it is not rigid, but often has a great deal of backtracking:

Let's look at the phases in detail:

- **Business understanding**: This task includes determining business objectives, assessing the current situation, establishing data mining goals, and developing a plan.
- **Data understanding**: This task evaluates data requirements and includes initial data collection, data description, data exploration, and the verification of data quality.
- **Data preparation**: Once available, data resources are identified in the last step. Then, the data needs to be selected, cleaned, and then built into the desired form and format.

- **Modeling**: Visualization and cluster analysis are useful for initial analysis. The initial association rules can be developed by applying tools such as generalized rule induction. This is a data mining technique to discover knowledge represented as rules to illustrate the data in the view of causal relationship between conditional factors and a given decision/outcome. The models appropriate to the data type can also be applied.
- **Evaluation** :The results should be evaluated in the context specified by the business objectives in the first step. This leads to the identification of new needs and in turn reverts to the prior phases in most cases.
- **Deployment**: Data mining can be used to both verify previously held hypotheses or for knowledge.

SEMMA

Here is an overview of the process for SEMMA:

Let's look at these processes in detail:

- **Sample**: In this step, a portion of a large dataset is extracted
- **Explore**: To gain a better understanding of the dataset, unanticipated trends and anomalies are searched in this step
- **Modify**: The variables are created, selected, and transformed to focus on the model construction process

- **Model**: A variable combination of models is searched to predict a desired outcome
- **Assess**: The findings from the data mining process are evaluated by its usefulness and reliability

Social network mining

As we mentioned before, data mining finds a *model* on data and the mining of social network finds the *model* on graph data in which the social network is represented.

Social network mining is one application of web data mining; the popular applications are social sciences and bibliometry, PageRank and HITS, shortcomings of the coarse-grained graph model, enhanced models and techniques, evaluation of topic distillation, and measuring and modeling the Web.

Social network

When it comes to the discussion of social networks, you will think of Facebook, Google+, LinkedIn, and so on. The essential characteristics of a social network are as follows:

- There is a collection of entities that participate in the network. Typically, these entities are people, but they could be something else entirely.
- There is at least one relationship between the entities of the network. On Facebook, this relationship is called friends. Sometimes, the relationship is all-or-nothing; two people are either friends or they are not. However, in other examples of social networks, the relationship has a degree. This degree could be discrete, for example, friends, family, acquaintances, or none as in Google+. It could be a real number; an example would be the fraction of the average day that two people spend talking to each other.
- There is an assumption of nonrandomness or locality. This condition is the hardest to formalize, but the intuition is that relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related.

Here are some varieties of social networks:

- **Telephone networks**: The nodes in this network are phone numbers and represent individuals
- **E-mail networks**: The nodes represent e-mail addresses, which represent individuals
- **Collaboration networks**: The nodes here represent individuals who published research papers; the edge connecting two nodes represent two individuals who published one or more papers jointly

Social networks are modeled as undirected graphs. The entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network. If there is a degree associated with the relationship, this degree is represented by labeling the edges.

Here is an example in which Coleman's High School Friendship Data from the sna R package is used for analysis. The data is from a research on friendship ties between 73 boys in a high school in one chosen academic year; reported ties for all informants are provided for two time points (fall and spring). The dataset's name is `coleman`, which is an array type in R language. The node denotes a specific student and the line represents the tie between two students.

Text mining

Text mining is based on the data of text, concerned with exacting relevant information from large natural language text, and searching for interesting relationships, syntactical correlation, or semantic association between the extracted entities or terms. It is also defined as automatic or semiautomatic processing of text. The related algorithms include text clustering, text classification, natural language processing, and web mining.

One of the characteristics of text mining is text mixed with numbers, or in other point of view, the hybrid data type contained in the source dataset. The text is usually a collection of unstructured documents, which will be preprocessed and transformed into a numerical and structured representation. After the transformation, most of the data mining algorithms can be applied with good effects.

The process of text mining is described as follows:

- Text mining starts from preparing the text corpus, which are reports, letters and so forth
- The second step is to build a semistructured text database that is based on the text corpus
- The third step is to build a term-document matrix in which the term frequency is included
- The final result is further analysis, such as text analysis, semantic analysis, information retrieval, and information summarization

Information retrieval and text mining

Information retrieval is to help users find information, most commonly associated with online documents. It focuses on the acquisition, organization, storage, retrieval, and distribution for information. The task of **Information Retrieval** (**IR**) is to retrieve relevant documents in response to a query. The fundamental technique of IR is measuring similarity. Key steps in IR are as follows:

- Specify a query. The following are some of the types of queries:
    - **Keyword query**: This is expressed by a list of keywords to find documents that contain at least one keyword
    - **Boolean query**: This is constructed with Boolean operators and keywords
    - **Phrase query**: This is a query that consists of a sequence of words that makes up a phrase

  - **Proximity query**: This is a downgrade version of the phrase queries and can be a combination of keywords and phrases
  - **Full document query**: This query is a full document to find other documents similar to the query document
  - **Natural language questions**: This query helps to express users' requirements as a natural language question

- Search the document collection.
- Return the subset of relevant documents.

Mining text for prediction

Prediction of results from text is just as ambitious as predicting numerical data mining and has similar problems associated with numerical classification. It is generally a classification issue.

Prediction from text needs prior experience, from the sample, to learn how to draw a prediction on new documents. Once text is transformed into numeric data, prediction methods can be applied.

Web data mining

Web mining aims to discover useful information or knowledge from the web hyperlink structure, page, and usage data. The Web is one of the biggest data sources to serve as the input for data mining applications.

Web data mining is based on IR, **machine learning** (**ML**), statistics, pattern recognition, and data mining. Web mining is not purely a data mining problem because of the heterogeneous and semistructured or unstructured web data, although many data mining approaches can be applied to it.

Web mining tasks can be defined into at least three types:

- **Web structure mining**: This helps to find useful information or valuable structural summary about sites and pages from hyperlinks
- **Web content mining**: This helps to mine useful information from web page contents
- **Web usage mining**: This helps to discover user access patterns from web logs to detect intrusion, fraud, and attempted break-in

The algorithms applied to web data mining are originated from classical data mining algorithms. They share many similarities, such as the mining process; however, differences exist too. The characteristics of web data mining makes it different from data mining for the following reasons:

- The data is unstructured
- The information of the Web keeps changing and the amount of data keeps growing
- Any data type is available on the Web, such as structured and unstructured data
- Heterogeneous information is on the web; redundant pages are present too
- Vast amounts of information on the web is linked
- The data is noisy

Web data mining differentiates from data mining by the huge dynamic volume of source dataset, a big variety of data format, and so on. The most popular data mining tasks related to the Web are as follows:

- **Information extraction** (**IE**): The task of IE consists of a couple of steps, tokenization, sentence segmentation, part-of-speech assignment, named entity identification, phrasal parsing, sentential parsing, semantic interpretation, discourse interpretation, template filling, and merging.
- **Natural language processing** (**NLP**): This researches the linguistic characteristics of human-human and human-machine interactive, models of linguistic competence and performance, frameworks to implement process with such models, processes'/models' iterative refinement, and evaluation techniques for the result systems. Classical NLP tasks related to web data mining are tagging, knowledge representation, ontologies, and so on.
- **Question answering**: The goal is to find the answer from a collection of text to questions in natural language format. It can be categorized into slot filling, limited domain, and open domain with bigger difficulties for the latter. One simple example is based on a predefined FAQ to answer queries from customers.
- **Resource discovery**: The popular applications are collecting important pages preferentially; similarity search using link topology, topical locality and focused crawling; and discovering communities.

Why R?

R is a high-quality, cross-platform, flexible, widely used open source, free language for statistics, graphics, mathematics, and data science—created by statisticians for statisticians.

R contains more than 5,000 algorithms and millions of users with domain knowledge worldwide, and it is supported by a vibrant and talented community of contributors. It allows access to both well-established and experimental statistical techniques.

R is a free, open source software environment maintained by R-projects for statistical computing and graphics, and the R source code is available under the terms of the Free Software Foundation's GNU General Public License. R compiles and runs on a wide variety for a variety of platforms, such as UNIX, LINUX, Windows, and Mac OS.

What are the disadvantages of R?

There are three shortages of R:

- One is that it is memory bound, so it requires the entire dataset store in memory (RAM) to achieve high performance, which is also called in-memory analytics.
- Similar to other open source systems, anyone can create and contribute package with strict or less testing. In other words, packages contributing to R communities are bug-prone and need more testing to ensure the quality of codes.
- R seems slow than some other commercial languages.

Fortunately, there are packages available to overcome these problems. There are some solutions that can be categorized as parallelism solutions; the essence here is to spread work across multiple CPUs that overcome the R shortages that were just listed. Good examples include, but are not limited to, RHadoop. You will read more on this topic soon in the following sections. You can download the SNOW add-on package and the Parallel add-on package from **Comprehensive R Archive Network** (**CRAN**).

Statistics

Statistics studies the collection, analysis, interpretation or explanation, and presentation of data. It serves as the foundation of data mining and the relations will be illustrated in the following sections.

Statistics and data mining

Statisticians were the first to use the term data mining. Originally, data mining was a derogatory term referring to attempts to extract information that was not supported by the data. To some extent, data mining constructs statistical models, which is an underlying distribution, used to visualize data.

Data mining has an inherent relationship with statistics; one of the mathematical foundations of data mining is statistics, and many statistics models are used in data mining.

Statistical methods can be used to summarize a collection of data and can also be used to verify data mining results.

Statistics and machine learning

Along with the development of statistics and machine learning, there is a continuum between these two subjects. Statistical tests are used to validate the machine learning models and to evaluate machine learning algorithms. Machine learning techniques are incorporated with standard statistical techniques.

Statistics and R

R is a statistical programming language. It provides a huge amount of statistical functions, which are based on the knowledge of statistics. Many R add-on package contributors come from the field of statistics and use R in their research.

The limitations of statistics on data mining

During the evolution of data mining technologies, due to statistical limits on data mining, one can make errors by trying to extract what really isn't in the data.

Bonferroni's Principle is a statistical theorem otherwise known as **Bonferroni correction**. You can assume that big portions of the items you find are bogus, that is, the items returned by the algorithms dramatically exceed what is assumed.

Machine learning

The data to which a ML algorithm is applied is called a training set, which consists of a set of pairs (`x`, `y`), called training examples. The pairs are explained as follows:

- `x`: This is a vector of values, often called the feature vector. Each value, or feature, can be categorical (values are taken from a set of discrete values, such as `{S, M, L}`) or numerical.
- `y`: This is the label, the classification or regression values for `x`.

The objective of the ML process is to discover a function $y = f(x)$ that best predicts the value of `y` associated with each value of `x`. The type of `y` is in principle arbitrary, but there are several common and important cases.

- `y`: This is a real number. The ML problem is called regression.
- `y`: This is a Boolean value true or false, more commonly written as +1 and -1, respectively. In this class, the problem is binary classification.
- `y`: Here this is a member of some finite set. The member of this set can be thought of as *classes*, and each member represents one class. The problem is multiclass classification.
- `y`: This is a member of some potentially infinite set, for example, a parse tree for `x`, which is interpreted as a sentence.

Until now, machine learning has not proved successful in situations where we can describe the goals of the mining more directly. Machine learning and data mining are two different topics, although some algorithms are shared between them—algorithms are shared especially when the goal is to extract information. There are situations where machine learning makes sense. The typical one is when we have idea of what we looking for in the dataset.

Approaches to machine learning

The major classes of algorithms are listed here. Each is distinguished by the function f.

- **Decision tree**: This form of f is a tree and each node of the tree has a function of `x` that determines which child or children the search must proceed for.
- **Perceptron**: These are threshold functions applied to the components of the vector $x = \{x_1, x_2, ...x_n\}$. A weight w_i is associated with the ith components, for each *i = 1, 2, … n*, and there is a threshold $\sum_{i=1}^{n} w_i x_i \geq \theta$. The output is +1 if and the output is -1 otherwise.

- **Neural nets**: These are acyclic networks of perceptions, with the outputs of some perceptions used as inputs to others.
- **Instance-based learning**: This uses the entire training set to represent the function f.
- **Support-vector machines**: The result of this class is a classifier that tends to be more accurate on unseen data. The target for class separation denotes as looking for the optimal hyper-plane separating two classes by maximizing the margin between the classes' closest points.

Machine learning architecture

The data aspects of machine learning here means the way data is handled and the way it is used to build the model.

- **Training and testing**: Assuming all the data is suitable for training, separate out a small fraction of the available data as the test set; use the remaining data to build a suitable model or classifier.
- **Batch versus online learning**: The entire training set is available at the beginning of the process for batch mode; the other one is online learning, where the training set arrives in a stream and cannot be revisited after it is processed.
- **Feature selection**: This helps to figure out what features to use as input to the learning algorithm.
- **Creating a training set**: This helps to create the label information that turns data into a training set by hand.

Data attributes and description

An **attribute** is a field representing a certain feature, characteristic, or dimensions of a data object.

In most situations, data can be modeled or represented with a matrix, columns for data attributes, and rows for certain data records in the dataset. For other cases, that data cannot be represented with matrices, such as text, time series, images, audio, video, and so forth. The data can be transformed into a matrix by appropriate methods, such as feature extraction.

The type of data attributes arises from its contexts or domains or semantics, and there are numerical, non-numerical, categorical data types or text data. Two views applied to data attributes and descriptions are widely used in data mining and R. They are as follows:

- **Data in algebraic or geometric view**: The entire dataset can be modeled into a matrix; linear algebraic and abstract algebra plays an important role here.
- **Data in probability view**: The observed data is treated as multidimensional random variables; each numeric attribute is a random variable. The dimension is the data dimension. Irrespective of whether the value is discrete or continuous, the probability theory can be applied here.

To help you learn R more naturally, we shall adopt a geometric, algebraic, and probabilistic view of the data.

Here is a matrix example. The number of columns is determined by *m*, which is the dimensionality of data. The number of rows is determined by *n*, which is the size of dataset.

$$A = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \left(X_1 \cdots X_m \right)$$

Where x_i denotes the *i* row, which is an m-tuple as follows:

$$x_i = (x_{i1} = \left(x_{i1}, \cdots, x_1 \right)$$

And x_j denotes the *j* column, which is an n-tuple as follows:

$$X_j = (X_{1j} = \left(X_{1j}, \cdots, X_{nj} \right)$$

Numeric attributes

Numerical data is convenient to deal with because it is quantitative and allows arbitrary calculations. The properties of numerical data are the same as integer or float data.

Numeric attributes taken from a finite or countable infinite set of values are called **discrete**, for example a human being's age, which is the integer value starting from 1,150. Other attributes taken from any real values are called **continuous**. There are two main kinds of numeric types:

- **Interval-scaled**: This is the quantitative value, measured on a scale of equal unit, such as the weight of some certain fish in the scale of international metric, such as gram or kilogram.
- **Ratio-scaled**: This value can be computed by ratios between values in addition to differences between values. It is a numeric attribute with an inherent zero-point; hence, we can say a value is a multiple of another value.

Categorical attributes

The values of categorical attributes come from a set-valued domain composed of a set of symbols, such as the size of human costumes that are categorized as `{S, M, L}`. The categorical attributes can be divided into two groups or types:

- **Nominal**: The values in this set are unordered and are not quantitative; only the equality operation makes sense here.
- **Ordinal**: In contrast to the nominal type, the data has an ordered meaning here. The inequality operation is available here in addition to the equality operation.

Data description

The basic description can be used to identify features of data, distinguish noise, or outliers. A couple of basic statistical descriptions are as follows:

- **Measures of central tendency**: This measures the location of middle or center of a data distribution: the mean, median, mode, midrange, and so on.
- **Measure of the dispersion of the data**: This is the range, quartiles, interquartile range, and so on.

Data measuring

Data measuring is used in clustering, outlier detection, and classification. It refers to measures of proximity, similarity, and dissimilarity. The similarity value, a real value, between two tuples or data records ranges from 0 to 1, the higher the value the greater the similarity between tuples. Dissimilarity works in the opposite way; the higher the dissimilarity value, the more dissimilar are the two tuples.

For a dataset, data matrix stores the n data tuples in n x m matrix (n tuples and m attributes):

$$\begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

The dissimilarity matrix stores a collection of proximities available for all n tuples in the dataset, often in a n x n matrix. In the following matrix, $d(i, j)$ means the dissimilarity between two tuples; value 0 for highly similar or near between each other, 1 for completely same, the higher the value, the more dissimilar it is.

$$\begin{pmatrix} 0 \\ d(2,1)0 \\ d(3,1)d(3,2)0 \\ \vdots\vdots\vdots \\ d(n,1)d(n,2)\ldots 0 \end{pmatrix}$$

Most of the time, the dissimilarity and similarity are related concepts. The similarity measure can often be defined using a function; the expression constructed with measures of dissimilarity, and vice versa.

Here is a table with a list of some of the most used measures for different attribute value types:

Attribute value types	**Dissimilarity**
Nominal attributes	The dissimilarity between two tuples can be computed by the following equation: $d(i, j) = (p-m)/p$ Where, p is the dimension of data and m is the number of matches that is in same state.
Ordinal attributes	The treatment of ordinal attributes is similar to that of numeric attributes, but it needs a transformation first before applying the methods.
Interval-scaled	**Euclidean**, **Manhattan**, and **Minkowski** distances are used to calculate the dissimilarity of data tuples.

Data cleaning

Data cleaning is one part of data quality. The aim of **Data Quality** (**DQ**) is to have the following:

- Accuracy (data is recorded correctly)
- Completeness (all relevant data is recorded)
- Uniqueness (no duplicated data record)
- Timeliness (the data is not old)
- Consistency (the data is coherent)

Data cleaning attempts to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data. Data cleaning is usually an iterative two-step process consisting of discrepancy detection and data transformation.

The process of data mining contains two steps in most situations. They are as follows:

- The first step is to perform audition on the source dataset to find the discrepancy.
- The second step is to choose the transformation to fix (based on the accuracy of the attribute to be modified and the closeness of the new value to the original value). This is followed by applying the transformation to correct the discrepancy.

Missing values

During the process to seize data from all sorts of data sources, there are many cases when some fields are left blank or contain a null value. Good data entry procedures should avoid or minimize the number of missing values or errors. The missing values and defaults are indistinguishable.

If some fields are missing a value, there are a couple of solutions—each with different considerations and shortages and each is applicable within a certain context.

- **Ignore the tuple**: By ignoring the tuple, you cannot make use of the remaining values except the missing one. This method is applicable when the tuple contains several attributes with missing values or the percentage of missing value per attribute doesn't vary considerably.
- **Filling the missing value manually**: This is not applicable for large datasets.
- **Use a global constant to fill the value**: Applying the value to fill the missing value will misguide the mining process, and is not foolproof.

- **Use a measure for a central tendency for the attribute to fill the missing value**: The measures of central tendency can be used for symmetric data distribution.
- **Use the attribute mean or median**: Use the attribute mean or median for all samples belonging to the same class as the given tuple.
- **Use the most probable value to fill the missing value**: The missing data can be filled with data determined with regression, inference-based tool, such as Bayesian formalism or decision tree induction.

The most popular method is the last one; it is based on the present values and values from other attributes.

Junk, noisy data, or outlier

As in a physics or statistics test, noise is a random error that occurs during the test process to seize the measured data. No matter what means you apply to the data gathering process, noise inevitably exists.

Approaches for data smoothing are listed here. Along with the progress of data mining study, new methods keep occurring. Let's have a look at them:

- **Binning**: This is a local scope smoothing method in which the neighborhood values are used to compute the final value for the certain bin. The sorted data is distributed into a number of bins and each value in that bin will be replaced by a value depending on some certain computation of the neighboring values. The computation can be bin median, bin boundary, which is the boundary data of that bin.
- **Regression**: The target of regression is to find the best curve or something similar to one in a multidimensional space; as a result, the other values will be used to predict the value of the target attribute or variable. In other aspects, it is a popular means for smoothing.
- **Classification or outlier**: The classifier is another inherent way to find the noise or outlier. During the process of classifying, most of the source data is grouped into couples of groups, except the outliers.

Data integration

Data integration combines data from multiple sources to form a coherent data store. The common issues here are as follows:

- **Heterogeneous data**: This has no common key
- **Different definition**: This is intrinsic, that is, same data with different definition, such as a different database schema
- **Time synchronization**: This checks if the data is gathered under same time periods
- **Legacy data**: This refers to data left from the old system
- **Sociological factors**: This is the limit of data gathering

There are several approaches that deal with the above issues:

- **Entity identification problem**: Schema integration and object matching are tricky. This referred to as the entity identification problem.
- **Redundancy and correlation analysis**: Some redundancies can be detected by correlation analysis. Given two attributes, such an analysis can measure how strongly one attribute implies the other, based on the available data.
- **Tuple Duplication**: Duplication should be detected at the tuple level to detect redundancies between attributes
- **Data value conflict detection and resolution**: Attributes may differ on the abstraction level, where an attribute in one system is recorded at a different abstraction level

Data dimension reduction

Reduction of dimensionality is often necessary in the analysis of complex multivariate datasets, which is always in high-dimensional format. So, for example, problems modeled by the number of variables present, the data mining tasks on the multidimensional analysis of qualitative data. There are also many methods for data dimension reduction for qualitative data.

The goal of dimensionality reduction is to replace large matrix by two or more other matrices whose sizes are much smaller than the original, but from which the original can be approximately reconstructed, usually by taking their product with loss of minor information.

Eigenvalues and Eigenvectors

An eigenvector for a matrix is defined as when the matrix (A in the following equation) is multiplied by the eigenvector (v in the following equation). The result is a constant multiple of the eigenvector. That constant is the eigenvalue associated with this eigenvector. A matrix may have several eigenvectors.

$$Av = \lambda v$$

An eigenpair is the eigenvector and its eigenvalue, that is, (v, λ) in the preceding equation.

Principal-Component Analysis

The **Principal-Component Analysis** (**PCA**) technique for dimensionality reduction views data that consists of a collection of points in a multidimensional space as a matrix, in which rows correspond to the points and columns to the dimensions.

The product of this matrix and its transpose has eigenpairs, and the principal eigenvector can be viewed as the direction in the space along which the points best line up. The second eigenvector represents the direction in which deviations from the principal eigenvector are the greatest.

Dimensionality reduction by PCA is to approximate the data by minimizing the root-mean-square error for the given number of columns in the representing matrix, by representing the matrix of points by a small number of its eigenvectors.

Singular-value decomposition

The **singular-value decomposition** (**SVD**) of a matrix consists of following three matrices:

- U
- $\sum$
- V

U and V are column-orthonormal; as vectors, the columns are orthogonal and their length is 1. $\sum$ is a diagonal matrix and the values along its diagonal are called singular values. The original matrix equals to the product of U, $\sum$, and the transpose of V.

SVD is useful when there are a small number of concepts that connect the rows and columns of the original matrix.

Dimensionality reduction by SVD for matrix U and V are typically as large as the original. To use fewer columns for U and V, delete the columns corresponding to the smallest singular values from U, V, and ∑. This minimizes the error in reconstruction of the original matrix from the modified U, ∑, and V.

CUR decomposition

The CUR decomposition seeks to decompose a sparse matrix into sparse, smaller matrices whose product approximates the original matrix.

The CUR chooses from a given sparse matrix a set of columns C and a set of rows R, which play the role of U and V^T in SVD. The choice of rows and columns is made randomly with a distribution that depends on the square root of the sum of the squares of the elements. Between C and R is a square matrix called U, which is constructed by a pseudo-inverse of the intersection of the chosen rows and columns.

By CUR solution, the three component matrices C, U, and R will be retrieved. The product of those three will approximate the original matrix M. For R community, rCUR is an R package for the CUR matrix decomposition.

Data transformation and discretization

As we know from the previous section, there are always some data formats that are best suited for specific data mining algorithms. Data transformation is an approach to transform the original data to preferable data format for the input of certain data mining algorithms before the processing.

Data transformation

Data transformation routines convert the data into appropriate forms for mining. They're shown as follows:

- **Smoothing**: This uses binning, regression, and clustering to remove noise from the data
- **Attribute construction**: In this routine, new attributes are constructed and added from the given set of attributes
- **Aggregation**: In this summary or aggregation, operations are performed on the data

- **Normalization**: Here, the attribute data is scaled so as to fall within a smaller range
- **Discretization**: In this routine, the raw values of a numeric attribute are replaced by interval label or conceptual label
- **Concept hierarchy generation for nominal data**: Here, attributes can be generalized to higher level concepts

Normalization data transformation methods

To avoid dependency on the choice of measurement units on data attributes, the data should be normalized. This means transforming or mapping the data to a smaller or common range. All attributes gain an equal weight after this process. There are many normalization methods. Let's have a look at some of them:

- **Min-max normalization**: This preserves the relationships among the original data values and performs a linear transformation on the original data. The applicable ones of the actual maximum and minimum values of an attribute will be normalized.
- **z-score normalization**: Here the values for an attribute are normalized based on the mean and standard deviation of that attribute. It is useful when the actual minimum and maximum of an attribute to be normalized are unknown.
- **Normalization by decimal scaling**: This normalizes by moving the decimal point of values of attribute.

Data discretization

Data discretization transforms numeric data by mapping values to interval or concept labels. Discretization techniques include the following:

- **Data discretization by binning**: This is a top-down unsupervised splitting technique based on a specified number of bins.
- **Data discretization by histogram analysis**: In this technique, a histogram partitions the values of an attribute into disjoint ranges called buckets or bins. It is also an unsupervised method.
- **Data discretization by cluster analysis**: In this technique, a clustering algorithm can be applied to discretize a numerical attribute by partitioning the values of that attribute into clusters or groups.

- **Data discretization by decision tree analysis**: Here, a decision tree employs a top-down splitting approach; it is a supervised method. To discretize a numeric attribute, the method selects the value of the attribute that has minimum entropy as a split-point, and recursively partitions the resulting intervals to arrive at a hierarchical discretization.
- **Data discretization by correlation analysis**: This employs a bottom-up approach by finding the best neighboring intervals and then merging them to form larger intervals, recursively. It is supervised method.

Visualization of results

Visualization is the graphic presentation of data-portrayals meant to reveal complex information at a glance, referring to all types of structured representation of information. This includes graphs, charts, diagrams, maps, storyboards, and other structured illustrations.

Good visualization of results gives you the chance to look at data through the eyes of experts. It is beautiful not only for their aesthetic design, but also for the elegant layers of detail that efficiently generate insight and new understanding.

The result of every data mining algorithm can be visualized and clarified by the use of the algorithms. Visualization plays an important role in the data mining process.

There are four major features that create the best visualizations:

- **Novel**: It must not only merely being a conduit for information, but offer some novelty in the form of new style of information.
- **Informative**: The attention to these factors and the data itself will make a data visualization effective, successful, and beautiful.
- **Efficient**: A nice visualization has an explicit goal, a clearly defined message, or a special perspective on the information that it is made to convey. It must be as simple as possible and straightforward, but shouldn't lose out on necessary, relevant complexity. The irrelevant data serves as noises here. It should reflect the qualities of the data that they represent, reveal properties and relationships inherent and implicit in the data source to bring new knowledge, insight, and enjoyment to final user.
- **Aesthetic**: The graphic must serve the primary goal of presenting information, not only axes and layout, shapes, lines, and typography, but also the appropriate usage of these ingredients.

Visualization with R

R provides the production of publication-quality diagrams and plots. There are graphic facilities distributed with R, and also some facilities that are not part of the standard R installation. You can use R graphics from command line.

The most important feature of the R graphics setup is the existence of two distinct graphics systems within R:

- The traditional graphics system
- Grid graphics system

The most appropriate facilities will be evaluated and applied to the visualization of every result of all algorithms listed in the book.

Functions in the graphics systems and add-on packages can be divided into several types:

- High-level functions that produce complete plots
- Low-level functions to add further output to an existing plot
- The ones to work interactively with graphical output

> R graphics output can be produced in a wide range of graphical formats, such as PNG, JPEG, BMP, TIFF, SVG, PDF, and PS.

To enhance your knowledge about this chapter, here are some practice questions for you to have check about the concepts.

Time for action

Let's now test what we've learned so far:

- What is the difference between data mining and machine learning?
- What is data preprocessing and data quality?
- Download R and install R on your machine.
- Compare and contrast data mining and machine learning.

Summary

In this chapter, we looked at the following topics:

- An introduction to data mining and available data sources
- A quick overview of R and the necessity to use R
- A description of statistics and machine learning, and their relations to data mining
- The two standard industrial data mining process
- Data attributes types and the data measurement approaches
- The three important steps in data preprocessing
- An introduction to the scalability and efficiency of data mining algorithms, and data visualization methods and necessities
- A discussion on social network mining, text mining, and web data mining
- A short introduction about RHadoop and Map Reduce

In the following chapters, the reader will learn how to implement various data mining algorithms and manipulate data with R.

2

Mining Frequent Patterns, Associations, and Correlations

In this chapter, we will learn how to mine frequent patterns, association rules, and correlation rules when working with R programs. Then, we will evaluate all these methods with benchmark data to determine the interestingness of the frequent patterns and rules. We will cover the following topics in this chapter:

- Introduction to associations and patterns
- Market basket analysis
- Hybrid association rules mining
- Mining sequence datasets
- High-performance algorithms

The algorithms to find frequent items from various data types can be applied to numeric or categorical data. Most of these algorithms have one common basic algorithmic form, which is **A-Priori**, depending on certain circumstances. Another basic algorithm is **FP-Growth**, which is similar to A-Priori. Most pattern-related mining algorithms derive from these basic algorithms.

With frequent patterns found as one input, many algorithms are designed to find association and correlation rules. Each algorithm is only a variation from the basic algorithm.

Along with the growth, size, and types of datasets from various domains, new algorithms are designed, such as the multistage algorithm, the multihash algorithm, and the limited-pass algorithm.

An overview of associations and patterns

One popular task for data mining is to find relations among the source dataset; this is based on searching frequent patterns from various data sources, such as market baskets, graphs, and streams.

All the algorithms illustrated in this chapter are written from scratch in the R language for the purpose of explaining association analysis, and the code will be demonstrated using the standard R packages for the algorithms such as arules.

Patterns and pattern discovery

With many applications across a broad field, frequent pattern mining is often used in solving various problems, such as the market investigation for a shopping mall from the transaction data.

Frequent patterns are the ones that often occur in the source dataset. The dataset types for frequent pattern mining can be itemset, subsequence, or substructure. As a result, the frequent patterns found are known as:

- Frequent itemset
- Frequent subsequence
- Frequent substructures

These three frequent patterns will be discussed in detail in the upcoming sections.

These newly founded frequent patterns will serve as an important platform when searching for recurring interesting rules or relationships among the given dataset.

Various patterns are proposed to improve the efficiency of mining on a dataset. Some of them are as follows; they will be defined in detail later:

- Closed patterns
- Maximal patterns
- Approximate patterns
- Condensed patterns
- Discriminative frequent patterns

The frequent itemset

The **frequent itemset** originated from true market basket analysis. In a store such as Amazon, there are many orders or transactions; a certain customer performs a transaction where their Amazon shopping cart includes some items. The mass result of all customers' transactions can be used by the storeowner to find out what items are purchased together by customers. As a simple definition, itemset denotes a collection of zero or more items.

We call a transaction a **basket**, and a set of items can belong to any basket. We will set the variable `s` as the support threshold, which is compared with the count of a certain set of items that appear in all the baskets. If the count of a certain set of items that appear in all the baskets is not less than `s`, we would call the itemset a **frequent itemset**.

An itemset is called a k-itemset if it contains *k* pieces of items, where k is a non-zero integer. The support count of an itemset is $support_count(X)$, the count of itemset contained X, given the dataset.

For a predefined minimum support threshold *s*, the itemset `X` is a frequent itemset if $support_count(X) \geq s$. The minimum support threshold `s` is a customizable parameter, which can be adjusted by domain experts or experiences.

The frequent itemset is also used in many domains. Some of them are shown in the following table:

	Items	**Baskets**	**Comments**
Related concepts	Words	Documents	
Plagiarism	Documents	Sentences	
Biomarkers	Biomarkers and diseases	The set of data about a patient	

If an itemset is frequent, then any of its subset must be frequent. This is known as the A-Priori principle, the foundation of the A-Priori algorithm. The direct application of the A-Priori principle is to prune the huge number of frequent itemsets.

One important factor that affects the number of frequent itemsets is the minimum support count: the lower the minimum support count, the larger the number of frequent itemsets.

For the purpose of optimizing the frequent itemset-generation algorithm, some more concepts are proposed:

- An itemset X is closed in dataset S, if $\forall Y \in S, X \subset Y, then\, support_count(X) \neq support_count(Y)$; X is also called a closed itemset. In other words, if X is frequent, then X is a closed frequent itemset.
- An itemset X is a maximal frequent itemset if $\forall Y \in S, X \subset Y, then\, Y\, is\, not\, frequent$; in other words, Y does not have frequent supersets.
- An itemset X is considered a constrained frequent itemset once the frequent itemset satisfies the user-specified constraints.
- An itemset X is an approximate frequent itemset if X derives only approximate support counts for the mined frequent itemsets.
- An itemset X is a top-k frequent itemset in the dataset S if X is the k-most frequent itemset, given a user-defined value k.

The following example is of a transaction dataset. All itemsets only contain items from the set, $D = \{\mathrm{I}_k | \{I_k | k \in [1,7]\}$.Let's assume that the minimum support count is 3.

tid (transaction id)	List of items in the itemset or transaction
T001	I_1, I_2, I_4, I_7
T002	I_2, I_3, I_6
T003	I_1, I_4, I_6
T004	I_1, I_2, I_5
T005	I_2, I_3, I_4
T006	I_2, I_5, I_6
T007	I_2, I_4, I_7
T008	I_1, I_7
T009	I_1, I_2, I_3
T010	I_1, I_2, I_4

Then, we will get the frequent itemsets $L_1 = \{\mathrm{I}_k | k \in \{1,2,4,6,7\}\}$ and $L_2 = \{\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}\}$.

The frequent subsequence

The frequent sequence is an ordered list of elements where each element contains at least one event. An example of this is the page-visit sequence on a site by the specific web page the user is on more concretely speaking, the order in which a certain user visits web pages. Here are two examples of the frequent subsequence:

- **Customer**: Successive shopping records of certain customers in a shopping mart serves as the sequence, each item bought serves as the event item, and all the items bought by a customer in one shopping are treated as elements or transactions
- **Web usage data**: Users who visit the history of the WWW are treated as a sequence, each UI/page serves as the event or item, and the element or transaction can be defined as the pages visited by users with one click of the mouse

The length of a sequence is defined by the number of items contained in the sequence. A sequence of length k is called a k-sequence. The size of a sequence is defined by the number of itemsets in the sequence. We call a sequence $s_1 = < a_1 a_2 ... a_r >$ as a subsequence of the sequence $s_2 =< b_1 b ... b_r >$ or s_2 as the super sequence of s_1 when $\exists 1 \le j_1 \le j_2 \le ... \le j_{r-1} \le j_r \le v, and\ a_1 \sqsubseteq b_{j_1}, a_2 \sqsubseteq b_{j_2}, ..., a_r \sqsubseteq b_{j_r}$ is satisfied.

The frequent substructures

In some domains, the tasks under research can be modeled with a graph theory. As a result, there are requirements for mining common subgraphs (subtrees or sublattices); some examples are as follows:

- **Web mining**: Web pages are treated as the vertices of graph, links between pages serve as edges, and a user's page-visiting records construct the graph.
- **Network computing**: Any device with computation ability on the network serves as the vertex, and the interconnection between these devices serves as the edge. The whole network that is made up of these devices and interconnections is treated as a graph.
- **Semantic web**: XML elements serve as the vertices, and the parent/child relations between them are edges; all these XML files are treated as graphs.

A graph G is represented by `G = (V, E)`, where `V` represents a group of vertices, and `E` represents a group of edges. A graph $G' = (V', E')$ is called as subgraph of graph `G = (V, E)` once $V' \subseteq V$ and $V' \subseteq V$. Here is an example of a subgraph. There is the original graph with vertices and edges on the left-hand side of the following figure and the subgraph on the right-hand side with some edges omitted (or omission of vertices in other circumstances):

Relationship or rules discovery

Mining of association rules is based on the frequent patterns found. The different emphases on the interestingness of relations derives two types of relations for further research: association rules and correlation rules.

Association rules

In a later section, a method to show association analysis is illustrated; this is a useful method to discover interesting relationships within a huge dataset. The relations can be represented in the form of association rules or frequent itemsets.

Association rule mining is to find the result rule set on a given dataset (the transaction data set or other sequence-pattern-type dataset), a predefined minimum support count *s*, and a predefined confidence c, given any found rule $X \rightarrow Y$ $\text{support_count}(\text{X} \rightarrow \text{Y}) \geq \text{S}$, and $confidence(X \rightarrow Y) \geq c$.

$X \rightarrow Y$ is an association rule where $X \cap Y = \phi$; X and Y are disjoint. The interesting thing about this rule is that it is measured by its **support** and **confidence**. Support means the frequency in which this rule appears in the dataset, and confidence means the probability of the appearance of `Y` when `X` is present.

For association rules, the key measures of rule interestingness are rule support and confidence. Their relationship is given as follows:

$$confidence(X \rightarrow Y) = P(Y|X) = \frac{P(X \cup Y)}{P(X)} = \frac{support_count(X \cup Y)}{support_count(X)}$$

`support_count(X)` is the count of itemset in the dataset, contained `X`.

As a convention, in `support_count(X)`, in the confidence value and support count value are represented as a percentage between 0 and 100.

The association rule $X \rightarrow Y$ is strong once $confidence(X \rightarrow Y) \geq c$ and $support_count(X \cup Y) > s$. The predefined minimum support threshold is s, and `c` is the predefined minimum confidence threshold.

The meaning of the found association rules should be explained with caution, especially when there is not enough to judge whether the rule implies causality. It only shows the co-occurrence of the prefix and postfix of the rule. The following are the different kinds of rules you can come across:

- A rule is a Boolean association rule if it contains association of the presence of the item
- A rule is a single-dimensional association if there is, at the most, only one dimension referred to in the rules
- A rule is a multidimensional association rule if there are at least two dimensions referred to in the rules
- A rule is a correlation-association rule if the relations or rules are measured by statistical correlation, which, once passed, leads to a correlation rule
- A rule is a quantitative-association rule if at least one item or attribute contained in it is quantitative

Correlation rules

In some situations, the support and confidence pairs are not sufficient to filter uninteresting association rules. In such a case, we will use support count, confidence, and correlations to filter association rules.

There are a lot of methods to calculate the correlation of an association rule, such as χ^2 analyses, all-confidence analysis, and cosine. For a k-itemset $X = \{i_1, i_2, \ldots, i_k\}$, define the all-confidence value of X as:

$$all_confidence(X) = support_count(X) / max\{support_count(i_j) \vee \forall i_j \in X\}$$

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{P(Y)} = P(X \cup Y) / (P(X)P(Y))$$

Market basket analysis

Market basket analysis is the methodology used to mine a shopping cart of items bought or just those kept in the cart by customers. The concept is applicable to a variety of applications, especially for store operations. The source dataset is a massive data record. The aim of market basket analysis is to find the association rules between the items within the source dataset.

The market basket model

The market basket model is a model that illustrates the relation between a basket and its associated items. Many tasks from different areas of research have this relation in common. To summarize them all, the market basket model is suggested as the most typical example to be researched.

The basket is also known as the transaction set; this contains the itemsets that are sets of items belonging to same itemset.

The A-Priori algorithm is a level wise, itemset mining algorithm. The Eclat algorithm is a tidset intersection itemset mining algorithm based on tidset intersection in contrast to A-Priori. FP-growth is a frequent pattern tree algorithm. The tidset denotes a collection of zeros or IDs of transaction records.

A-Priori algorithms

As a common strategy to design algorithms, the problem is divided into two subproblems:

- The frequent itemset generation
- Rule generation

The strategy dramatically decreases the search space for association mining algorithms.

Input data characteristics and data structure

As the input of the A-Priori algorithm, the original input itemset is binarized, that is, 1 represents the presence of a certain item in the itemset; otherwise, it is 0. As a default assumption, the average size of the itemset is small. The popular preprocessing method is to map each unique available item in the input dataset to a unique integer ID.

The itemsets are usually stored within databases or files and will go through several passes. To control the efficiency of the algorithm, we need to control the count of passes. During the process when itemsets pass through other itemsets, the representation format for each itemset you are interested in is required to count and store for further usage of the algorithm.

There is a monotonicity feature in the itemsets under research; this implies that every subset of a frequent itemset is frequent. This characteristic is used to prune the search space for the frequent itemset in the process of the A-Priori algorithm. It also helps compact the information related to the frequent itemset. This feature gives us an intrinsic view that focuses on smaller-sized frequent itemsets. For example, there are three frequent 2-itemsets contained by one certain frequent 3-itemset.

> When we talk about k-itemsets means an itemset containing k items.

The basket is in a format called the **horizontal format** and contains a basket or transaction ID and a number of items; it is used as the basic input format for the A-Priori algorithm. In contrast, there is another format known as the **vertical format**; this uses an item ID and a series of the transaction IDs. The algorithm that works on vertical data format is left as an exercise for you.

The A-Priori algorithm

Two actions are performed in the generation process of the A-Priori frequent itemset: one is **join**, and the other is **prune**.

> One important assumption is that the items within any itemset are in a lexicographic order.

- **Join action**: Given that L_k is the set of frequent k-itemsets, a set of candidates to find L_k is generated. Let's call it C_k.

$$C_k = L_{k-1} \bowtie L_{k-1} = \{l' | \exists\, l_1, l_2 \in L_{k-1}, \forall\, m \in [1, k-2], l_1[m] = l_2[m] \text{ and } l_1[k-1] \le l_2[k-1], \text{ then } l' < l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1] > \}$$

- **Prune action**: $L_k \subseteq C_k$, the size of C_k, the candidate itemset, is usually much bigger than L_k, to save computation cost; monotonicity characteristic of frequent itemset is used here to prune the size of C_k.

$$\forall c \in C_k, \exists c_{k-1} \text{ is } a(k-1)-\text{subset of } c, \text{and } c_{k-1} \notin L_{k-1} \Rightarrow c_k \notin L_k$$

Here is the pseudocode to find all the frequent itemsets:

```
APRIORI (D, I, minsup)
F ← ∅
C^(1) ← {∅}
foreach i ∈ I do  Add i as child of ∅ in C^(1) with sup(i) ← 0
k ← 1
while C^(k) ≠ ∅ do
    COMPUTESUPPORT (C^(k), D)
    foreach  leaf X ∈ C^(k) do
        if sup(X) ≥ minsup then  F ← F ∪ {(X, sup(X))}
        else  remove X from C^(k)
    C^(k+1) ← EXTENDPREFIXTREE (C^(k))
    k ← k + 1
return F^(k)

COMPUTESUPPORT (C^(k), D):
foreach ⟨t, i(t)⟩ ∈ D do
    foreach k-subset X ⊆ i(t) do
        if X ∈ C^(k) then  sup(X) ← sup(X) + 1
```

$\textsc{ExtendPrefixTree}\ (\mathcal{C}^{(k)})$:
foreach *leaf* $X_a \in \mathcal{C}^{(k)}$ **do**
  foreach *leaf* $X_b \in \textsc{sibling}(X_a)$, *such that* $b > a$ **do**
    $X_{ab} \leftarrow X_a \cup X_b$

    if $X_j \in \mathcal{C}^{(k)}$, ***for all*** $X_j \subset X_{ab}$, *such that* $|X_j| = |X_{ab}| - 1$ **then**
      Add X_{ab} as child of X_a with $sup(X_{ab}) \leftarrow 0$

  if *no extensions from* X_a **then** remove X_a from $\mathcal{C}^{(k)}$
return $\mathcal{C}^{(k)}$

The R implementation

R code of the A-Priori frequent itemset generation algorithm goes here. D is a transaction dataset. Suppose `MIN_SUP` is the minimum support count threshold. The output of the algorithm is L, which is a frequent itemsets in D.

The output of the A-Priori function can be verified with the R add-on package, arules, which is a pattern-mining and association-rules-mining package that includes A-Priori and éclat algorithms. Here is the R code:

```
Apriori <- function (data, I, MIN_SUP, parameter = NULL){
  f <- CreateItemsets()
  c <- FindFrequentItemset(data,I,1, MIN_SUP)
  k <- 2
  len4data <- GetDatasetSize(data)
  while( !IsEmpty(c[[k-1]]) ){
        f[[k]] <- AprioriGen(c[k-1])
         for( idx in 1: len4data ){
             ft <- GetSubSet(f[[k]],data[[idx]])
             len4ft <- GetDatasetSize(ft)
             for( jdx in 1:len4ft ){
                IncreaseSupportCount(f[[k]],ft[jdx])
             }
         }
         c[[k]] <- FindFrequentItemset(f[[k]],I,k,MIN_SUP)
         k <- k+1
  }
  c
}
```

To verify the R code, the arules package is applied while verifying the output.

> Arules (*Hahsler et al., 2011*) provides the support to mine frequent itemsets, maximal frequent itemsets, closed frequent itemsets, and association rules too. A-Priori and Eclat algorithms are both available. Also cSPADE can be found in arulesSequence, the add-on for arules.

Given:

$$D = \{tinnedfruit, tuna, milk, coke, water, biscuits, oil, soap\}$$

At first, we will sort D into an ordered list in a predefined order algorithm or simply the natural order of characters, which is used here. Then:

$$D = \begin{Bmatrix} I1 = biscuits, I2 = coke, I3 = milk, I4 = oil, \\ I5 = soap, I6 = tinnedfruit, I7 = tuna \end{Bmatrix} = \{I_k | k \in [1,7]\}$$

Let's assume that the minimum support count is 5; the following table is an input dataset:

tid (transaction id)	List of items in the itemset or transaction
T001	I_1, I_2, I_4, I_7
T002	I_2, I_3, I_6
T003	I_1, I_4, I_6
T004	I_1, I_2, I_5
T005	I_2, I_3, I_4
T006	I_2, I_5, I_6
T007	I_2, I_4, I_7
T008	I_1, I_7
T009	I_1, I_2, I_3
T010	I_1, I_2, I_4

In the first scan or pass of the dataset D, get the count of each candidate itemset C_1. The candidate itemset and its related count:

Itemset	Support count
$\{I_1\}$	6
$\{I_2\}$	8
$\{I_3\}$	2
$\{I_4\}$	5
$\{I_5\}$	2
$\{I_6\}$	3
$\{I_7\}$	3

We will get the L_1 after comparing the support count with minimum support count.

Itemset	Support count
$\{I_1\}$	6
$\{I_2\}$	8
$\{I_4\}$	5

We will generate C_2 by L_1, $C_2 = \{\{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}\}$.

Itemset	Support count
$\{I_1, I_2\}$	4
$\{I_1, I_4\}$	3
$\{I_2, I_4\}$	4

After comparing the support count with the minimum support count, we will get $L_2 = \phi$. The algorithm then terminates.

A-Priori algorithm variants

The various variants of A-Priori algorithms are designed mainly for the purpose of efficiency and scalability. Some of the improvements of the A-Priori algorithms are discussed in the upcoming sections.

The Eclat algorithm

The A-Priori algorithm loops as many times as the maximum length of the pattern somewhere. This is the motivation for the **Equivalence CLASS Transformation** (**Eclat**) algorithm. The Eclat algorithm explores the vertical data format, for example, using `<item id, tid set>` instead of `<tid, item id set>` that is, with the input data in the vertical format in the sample market basket file, or to discover frequent itemsets from a transaction dataset. The A-Priori property is also used in this algorithm to get frequent (*k+1*) itemsets from k itemsets.

The candidate itemset is generated by set intersection. The vertical format structure is called a tidset as defined earlier. If all the transaction IDs related to the item *I* are stored in a vertical format transaction itemset, then the itemset is the tidset of the specific item.

The support count is computed by the intersection between tidsets. Given two tidsets, `X` and `Y`, $support_count(X \cap Y)$ is the cardinality of $X \cap Y$. The pseudocode is $\mathrm{F} \leftarrow \phi$, $\mathrm{P} \leftarrow \{<i, t(i)> | i \in I, |t(i)| \geq MIN_SUP\}$.

$$
\begin{array}{l}
\textbf{ECLAT } (P, \textit{minsup}, \mathcal{F}): \\
\textbf{foreach } \langle X_a, \mathbf{t}(X_a) \rangle \in P \textbf{ do} \\
\quad \mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\} \\
\quad P_a \leftarrow \emptyset \\
\quad \textbf{foreach } \langle X_b, \mathbf{t}(X_b) \rangle \in P, \textit{ with } X_b > X_a \textbf{ do} \\
\quad\quad X_{ab} = X_a \cup X_b \\
\quad\quad \mathbf{t}(X_{ab}) = \mathbf{t}(X_a) \cap \mathbf{t}(X_b) \\
\quad\quad \textbf{if } sup(X_{ab}) \geq \textit{minsup} \textbf{ then} \\
\quad\quad\quad P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{t}(X_{ab}) \rangle\} \\
\quad \textbf{if } P_a \neq \emptyset \textbf{ then } \ \textsc{Eclat} \ (P_a, \textit{minsup}, \mathcal{F})
\end{array}
$$

The R implementation

Here is the R code for the Eclat algorithm to find the frequent patterns. Before calling the function, `f` is set to empty, and `p` is the set of frequent 1-itemsets:

```
Eclat  <- function (p,f,MIN_SUP){
  len4tidsets <- length(p)
  for(idx in 1:len4tidsets){
     AddFrequentItemset(f,p[[idx]],GetSupport(p[[idx]]))
     Pa <- GetFrequentTidSets(NULL,MIN_SUP)
       for(jdx in idx:len4tidsets){
         if(ItemCompare(p[[jdx]],p[[idx]]) > 0){
             xab <- MergeTidSets(p[[idx]],p[[jdx]])
             if(GetSupport(xab)>=MIN_SUP){
                AddFrequentItemset(pa,xab,
                GetSupport(xab))
               }
           }
     }
     if(!IsEmptyTidSets(pa)){
       Eclat(pa,f,MIN_SUP)
    }
  }
}
```

Here is the running result of one example, `I = {beer, chips, pizza, wine}`. The transaction dataset with horizontal and vertical formats, respectively, are shown in the following table:

tid	X
1	{beer, chips, wine}
2	{beer, chips}
3	{pizza, wine}
4	{chips, pizza}

x	tidset
beer	{1,2}
chips	{1,2,4}
pizza	{3,4}
wine	{1,3}

The binary format of this information is in the following table.

tid	beer	chips	pizza	wine
1	1	1	0	1
2	1	1	0	0
3	0	0	1	1
4	0	1	1	0

Before calling the Eclat algorithm, we will set *MIN_SUP*=2, $F \leftarrow \{\}$,

$$P \leftarrow \{< beer,12 >,< chips,124 >,< pizza,34 >,< wine,13 >\}$$

The running process is illustrated in the following figure. After two iterations, we will get frequent tidsets, `{beer, 12 >, < chips, 124>, <pizza, 34>, <wine, 13>, < {beer, chips}, 12>}`:

The output of the Eclat function can be verified with the R add-on package, arules.

The FP-growth algorithm

The FP-growth algorithm is an efficient method targeted at mining frequent itemsets in a large dataset. The main difference between the FP-growth algorithm and the A-Priori algorithm is that the generation of a candidate itemset is not needed here. The pattern-growth strategy is used instead. The FP-tree is the data structure.

Input data characteristics and data structure

The data structure used is a hybrid of vertical and horizontal datasets; all the transaction itemsets are stored within a tree structure. The tree structure used in this algorithm is called a frequent pattern tree. Here is example of the generation of the structure, I = {A, B, C, D, E, F}; the transaction dataset D is in the following table, and the FP-tree building process is shown in the next upcoming image. Each node in the FP-tree represents an item and the path from the root to that item, that is, the node list represents an itemset. The support information of this itemset is included in the node as well as the item too.

tid	**X**
1	{A, B, C, D, E}
2	{A, B, C, E}
3	{A, D, E}
4	{B, E, D}
5	{B, E, C}
6	{E, C, D}
7	{E, D}

The sorted item order is listed in the following table:

item	E	D	C	B	A
support_count	7	5	4	4	3

Reorder the transaction dataset with this new decreasing order; get the new sorted transaction dataset, as shown in this table:

tid	**X**
1	{E, D, C, B, A}
2	{E, C, B, A}
3	{E, D, A}
4	{E, D, B}
5	{E, C, B}
6	{E, D, C}
7	{E, D}

The FP-tree building process is illustrated in the following images, along with the addition of each itemset to the FP-tree. The support information is calculated at the same time, that is, the support counts of the items on the path to that specific node are incremented.

The most frequent items are put at the top of the tree; this keeps the tree as compact as possible. To start building the FP-tree, the items should be decreasingly ordered by the support count. Next, get the list of sorted items and remove the infrequent ones. Then, reorder each itemset in the original transaction dataset by this order.

Given *MIN_SUP=3*, the following itemsets can be processed according to this logic:

FP-tree building process, step1~step3

The result after performing steps 4 and 7 are listed here, and the process of the algorithm is very simple and straight forward:

FP-tree building process, step4~step7

A header table is usually bound together with the frequent pattern tree. A link to the specific node, or the item, is stored in each record of the header table.

The FP-tree serves as the input of the FP-growth algorithm and is used to find the frequent pattern or itemset. Here is an example of removing the items from the frequent pattern tree in a reverse order or from the leaf; therefore, the order is *A*, *B*, *C*, *D*, and *E*. Using this order, we will then build the projected FP-tree for each item.

The FP-growth algorithm

Here is the pseudocode with recursion definition; the input values are $R \leftarrow GenerateFPTree(D), P \leftarrow \phi, F \leftarrow \phi$

$\mathcal{F}[I] := \{\}$
for all $i \in \mathcal{I}$ occurring in $\mathcal{D}$ **do**
  $\mathcal{F}[I] := \mathcal{F}[I] \cup \{I \cup \{i\}\}$

  $\mathcal{D}^i := \{\}$
  $H := \{\}$
  for all $j \in \mathcal{I}$ occurring in $\mathcal{D}$ such that $j > i$ **do**
    if $support(I \cup \{i, j\}) \geq \sigma$ **then**
      $H := H \cup \{j\}$
    end if
  end for
  for all $(tid, X) \in \mathcal{D}$ with $i \in X$ **do**
    $\mathcal{D}^i := \mathcal{D}^i \cup \{(tid, X \cap H)\}$
  end for
  // Depth-first recursion
  Compute $\mathcal{F}[I \cup \{i\}](\mathcal{D}^i, \sigma)$
  $\mathcal{F}[I] := \mathcal{F}[I] \cup \mathcal{F}[I \cup \{i\}]$
end for

The R implementation

Here is the R source code of the main FP-growth algorithm:

```
FPGrowth  <- function (r,p,f,MIN_SUP){
    RemoveInfrequentItems(r)
    if(IsPath(r)){
       y <- GetSubset(r)
       len4y <- GetLength(y)
       for(idx in 1:len4y){
          x <- MergeSet(p,y[idx])
          SetSupportCount(x, GetMinCnt(x))
          Add2Set(f,x,support_count(x))
       }
  }else{
      len4r <- GetLength(r)
      for(idx in 1:len4r){
         x <- MergeSet(p,r[idx])
         SetSupportCount(x, GetSupportCount(r[idx]))
         rx <- CreateProjectedFPTree()
```

```
            path4idx <- GetAllSubPath(PathFromRoot(r,idx))
            len4path <- GetLength(path4idx)
            for( jdx in 1:len4path ){
              CountCntOnPath(r, idx, path4idx, jdx)
              InsertPath2ProjectedFPTree(rx, idx, path4idx, jdx,
                GetCnt(idx))
            }
            if( !IsEmpty(rx) ){
              FPGrowth(rx,x,f,MIN_SUP)
            }
        }
    }
}
```

The GenMax algorithm with maximal frequent itemsets

The GenMax algorithm is used to mine **maximal frequent itemset** (**MFI**) to which the maximality properties are applied, that is, more steps are added to check the maximal frequent itemsets instead of only frequent itemsets. This is based partially on the tidset intersection from the Eclat algorithm. The diffset, or the differential set as it is also known, is used for fast frequency testing. It is the difference between two tidsets of the corresponding items.

The candidate MFI is determined by its definition: assuming M as the set of MFI, if there is one X that belongs to M and it is the superset of the newly found frequent itemset Y, then Y is discarded; however, if X is the subset of Y, then X should be removed from M.

Here is the pseudocode before calling the GenMax algorithm,
$M \leftarrow \phi, and\ P \leftarrow \{< \mathrm{X}_i,\ \mathrm{t}(\mathrm{X}_i) > | \mathrm{X}_i \in D, support_count(X_i) \geq MIN_SUP\}$,
where D is the input transaction dataset.

$\textsc{GenMax}$ $(P,\ minsup,\ \mathcal{M})$:
$Y \leftarrow \bigcup X_i$
if $\exists Z \in \mathcal{M},\ such\ that\ Y \subseteq Z$ **then**
  return
foreach $\langle X_i, \mathbf{t}(X_i)\rangle \in P$ **do**
  $P_i \leftarrow \emptyset$
  foreach $\langle X_j, \mathbf{t}(X_j)\rangle \in P,\ with\ j > i$ **do**
    $X_{ij} \leftarrow X_i \cup X_j$
    $\mathbf{t}(X_{ij}) = \mathbf{t}(X_i)\ \cap\ \mathbf{t}(X_j)$
    if $sup(X_{ij}) \geq minsup$ **then** $P_i \leftarrow P_i \cup \{\langle X_{ij}, \mathbf{t}(X_{ij})\rangle\}$
  if $P_i \neq \emptyset$ **then** $\textsc{GenMax}$ $(P_i,\ minsup,\ \mathcal{M})$
  else if $\nexists Z \in \mathcal{M}, X_i \subseteq Z$ **then**
    $\mathcal{M} = \mathcal{M} \cup X_i$

The R implementation

Here is the R source code of the main GenMax algorithm:

```
GenMax  <- function (p,m,MIN_SUP){
  y <- GetItemsetUnion(p)
  if( SuperSetExists(m,y) ){
    return
  }
  len4p <- GetLenght(p)
  for(idx in 1:len4p){
     q <- GenerateFrequentTidSet()
     for(jdx in (idx+1):len4p){
       xij <- MergeTidSets(p[[idx]],p[[jdx]])
         if(GetSupport(xij)>=MIN_SUP){
            AddFrequentItemset(q,xij,GetSupport(xij))
          }
     }
     if( !IsEmpty(q) ){
       GenMax(q,m,MIN_SUP)
     }else if( !SuperSetExists(m,p[[idx]]) ){
     Add2MFI(m,p[[idx]])
     }
   }
}
```

The Charm algorithm with closed frequent itemsets

Closure checks are performed during the mining of closed frequent itemsets. Closed frequent itemsets allow you to get the longest frequent patterns that have the same support. This allows us to prune frequent patterns that are redundant. The Charm algorithm also uses the vertical tidset intersection for efficient closure checks.

Here is the pseudocode before calling the Charm algorithm,
$C \leftarrow \phi, and\ P \leftarrow \{< X_i,\ t(X_i) > | X_i \in D, support_count(X_i) \geq MIN_SUP\}$,
where D is the input transaction dataset.

```
CHARM (D, min_sup):
    [P] = {X_i × t(X_i) : X_i ∈ I ∧ σ(X_i) ≥ min_sup}
    CHARM-EXTEND ([P], C = ∅)
    return C

CHARM-EXTEND ([P], C):
    for each X_i × t(X_i) in [P]
        [P_i] = ∅ and X = X_i
        for each X_j × t(X_j) in [P], with X_j ≥_f X_i
            X = X ∪ X_j and Y = t(X_i) ∩ t(X_j)
            CHARM-PROPERTY([P], [P_i])
        if ([P_i] ≠ ∅) then CHARM-EXTEND ([P_i], C)
        delete [P_i]
        C = C ∪ X

CHARM-PROPERTY ([P], [P_i]):
    if (σ(X) ≥ minsup) then
        if t(X_i) = t(X_j) then
            Remove X_j from [P]
            Replace all X_i with X
        else if t(X_i) ⊂ t(X_j) then
            Replace all X_i with X
        else if t(X_i) ⊃ t(X_j) then
            Remove X_j from [P]
            Add X × Y to [P_i]
        else if t(X_i) ≠ t(X_j) then
            Add X × Y to [P_i]
```

The R implementation

Here is the R source code of the main algorithm:

```
Charm  <- function (p,c,MIN_SUP){
  SortBySupportCount(p)
  len4p <- GetLength(p)
  for(idx in 1:len4p){
     q <- GenerateFrequentTidSet()
     for(jdx in (idx+1):len4p){
        xij <- MergeTidSets(p[[idx]],p[[jdx]])
        if(GetSupport(xij)>=MIN_SUP){
          if( IsSameTidSets(p,idx,jdx) ){
            ReplaceTidSetBy(p,idx,xij)
            ReplaceTidSetBy(q,idx,xij)
            RemoveTidSet(p,jdx)
          }else{
             if( IsSuperSet(p[[idx]],p[[jdx]]) ){
               ReplaceTidSetBy(p,idx,xij)
               ReplaceTidSetBy(q,idx,xij)
             }else{
               Add2CFI(q,xij)
             }
          }
        }
     }
     if( !IsEmpty(q) ){
       Charm(q,c,MIN_SUP)
     }
     if( !IsSuperSetExists(c,p[[idx]]) ){
        Add2CFI(m,p[[idx]])
     }
  }
}
```

The algorithm to generate association rules

During the process of generating an algorithm for A-Priori frequent itemsets, the support count of each frequent itemset is calculated and recorded for further association rules mining processing, that is, for association rules mining.

To generate an association rule $X \to Y, l = X \cup Y$, l is a frequent itemset. Two steps are needed:

- First to get all the nonempty subsets of l.
- Then, for subset X of l, $Y = l - X$, the rule $X \to Y$ is a strong association rule only if $confidence(X \to Y) \geq minimum_{confidence}$ The support count of any rule of a frequent itemset is not less than the minimum support count.

Here is the pseudocode:

$$
\begin{array}{ll}
1: & AprioriGenerateRules(D, F, MIN_{SUP}, MIN_CONF)\ \{ \\
2: & \quad R \leftarrow \phi; \\
3: & \quad for(each\ I \in F)\ \{ \\
4: & \quad\quad R \leftarrow R \cup I \Rightarrow \phi; \\
5: & \quad\quad C_1 \leftarrow \{\{1\} \mid i \in I\}; \\
6: & \quad\quad k \leftarrow 1 \\
7: & \quad\quad while(C_k \neq \phi)\{ \\
8: & \quad\quad\quad H_k \leftarrow \{X \in C_k \mid confidence(I - X \Rightarrow X) \geq MIN_CONF\}; \\
9: & \quad\quad\quad for(any\ X, Y \in H_k,\ X[i] == Y[i]\ for\ 1 \leq i \leq k-1, and\ X[k] < Y[k])\ )\{ \\
10: & \quad\quad\quad\quad I \leftarrow X \cup \{Y[k]\}; \\
11: & \quad\quad\quad\quad if(\forall J \subseteq I, |J| == k, J \in H_k)\{ \\
12: & \quad\quad\quad\quad\quad C_{k+1} \leftarrow C_{k+1} \cup I \\
13: & \quad\quad\quad\quad \} \\
14: & \quad\quad\quad \} \\
15: & \quad\quad\quad k \leftarrow k+1; \\
16: & \quad\quad \} \\
17: & \quad\quad R \leftarrow R \cup \{I - X \Rightarrow X | X \in H_1 \cup \ldots \in H_k\}; \\
18: & \quad \}
\end{array}
$$

The R implementation

R code of the algorithm to generate A-Priori association is as follows:

```
Here is the R source code of the main
  algorithm:
AprioriGenerateRules  <- function
  (D,F,MIN_SUP,MIN_CONF){
```

```
    #create empty rule set
    r <- CreateRuleSets()
    len4f <- length(F)
    for(idx in 1:len4f){
       #add rule F[[idx]] => {}
       AddRule2RuleSets(r,F[[idx]],NULL)
       c <- list()
       c[[1]] <- CreateItemSets(F[[idx]])
       h <- list()
       k <-1
       while( !IsEmptyItemSets(c[[k]]) ){
         #get heads of confident association rule in c[[k]]
         h[[k]] <- getPrefixOfConfidentRules(c[[k]],
         F[[idx]],D,MIN_CONF)
         c[[k+1]] <- CreateItemSets()

         #get candidate heads
         len4hk <- length(h[[k]])
         for(jdx in 1:(len4hk-1)){
            if( Match4Itemsets(h[[k]][jdx],
               h[[k]][jdx+1]) ){
               tempItemset <- CreateItemset
               (h[[k]][jdx],h[[k]][jdx+1][k])
               if( IsSubset2Itemsets(h[[k]],
                 tempItemset) ){
                 Append2ItemSets(c[[k+1]],
                 tempItemset)
           }
         }
       }
     }
     #append all new association rules to rule set
     AddRule2RuleSets(r,F[[idx]],h)
     }
    r
  }
```

To verify the R code, Arules and Rattle packages are applied while verifying the output.

> Arules (*Hahsler et al., 2011*) and Rattle packages provide support for association rule analysis. AruleViz is used to visualize the output's association rules.

Hybrid association rules mining

There are two interesting applications of association rules mining: one is **multilevel and multidimensional association rules mining**, while the other is **constraint-based mining**.

Mining multilevel and multidimensional association rules

For a given transactional dataset, if there is a conceptual hierarchy that exists from some dimensions of the dataset, then we can apply multilevel association rules mining to this dataset. Any association rules mining algorithm applicable to the transaction dataset can be used for this task. The following table shows an example from the Amazon store:

TID	Item purchased
1	Dell Venue 7 16 GB Tablet, HP Pavilion 17-e140us 17.3-Inch Laptop...
2	Samsung Galaxy Tab 3 Lite, Razer Edge Pro 256GB Tablet…
2	Acer C720P-2666 Chromebook, Logitech Wireless Combo MK270 with Keyboard and Mouse…
2	Toshiba CB35-A3120 13.3-Inch Chromebook, Samsung Galaxy Tab 3 (7-Inch, White)…

Have a look at the following flowchart that explains multilevel pattern mining:

Based on the conceptual hierarchy, lower-level concepts can be projected to higher-level concepts, and the new dataset with higher-level concepts can replace the original lower-level concepts.

The support counts are calculated at each conceptual level. Many A-Priori-like algorithms are designed with slightly different treatment to support count; here is a possible list of treatments available:

- A uniform minimum support threshold is used across all the levels
- Reduced minimum support threshold is used for lower levels
- Group-based minimum support threshold

Sometimes the A-Priori property is not always held here. There are some exceptions.

Multilevel association rules are mined from multiple levels of the conceptual hierarchy.

Constraint-based frequent pattern mining

Constraint-based frequent pattern mining is a heuristic method with some user-specified constraints to prune the search space.

The ordinary constraints are, but not limited to, the following:

- Knowledge-type constraint (specifies what we are going to mine)
- Data constraint (limits to the original dataset)
- Dimension-level constraints
- Interestingness constraints
- Rule constraints

Mining sequence dataset

Sequential pattern mining is the major task for sequence dataset mining. The A-Priori-life algorithm is used to mine sequence patterns that use the A-Priori-life algorithm, which applies a breath-first strategy. However, for the pattern-growth method, a depth-first strategy is used instead. The algorithm sometimes integrates with constraints for various reasons.

The common purchase patterns of the customers of the store can be mined from sequential patterns. In other aspects, especially advertisement or market campaign, sequential patterns play an important role. The individual customer's behavior can be predicted from sequential patterns in the domain of web log mining, web page recommendation system, bioinformatics analysis, medical treatment sequence track and analysis, and disaster prevention and safety management.

The rules in this chapter, which are mined from sequence patterns, are of many types. Some of them are listed as follows:

- A **sequential rule** is $X \to Y$, where $X \subset Y$
- A **label sequential rule** (**LSR**) is of the form $X \to Y$, where *Y* is a sequence, and *X* a sequence generated from *Y* by replacing some of its items with wildcards
- A **class sequential rule** (**CSR**) is defined as *X* if:

$$X \to y, given\ S\ as\ a\ sequence\ dataset, I\ as\ the\ set\ of\ all\ items \in S, Y as\ the\ set\ of\ all\ class\ lables, I \bigcap Y = \phi, X\ as\ a\ sequence, y \in Y,$$

Sequence dataset

A sequence dataset S is defined as a set of tuples, *(sid, s)*, in which sid is a sequence ID, and s is a sequence.

The support of a sequence X in a sequence dataset S is the number of tuples in S, which contains `X`: $support_S(X) = \{(sid, s) \vee (sid, s) \in S \leftarrow X \subseteq s\}$.

Here is a property intrinsic to sequential patterns, and it is applied to related algorithms such as the A-Priori property for the A-Priory algorithm. For a sequence `X` and its subsequence `Y`, $support(X) \leq support(Y)$.

The GSP algorithm

The **generalized sequential patterns** (**GSP**) algorithm is an A-Priori-like algorithm, but it is applied to sequence patterns. It is a level-wise algorithm and has a breadth-first approach. Here is the feature list:

- GSP is an extension of the A-Priori algorithm

  It uses the A-Priori property (downward-closed), that is, given the minimum support count, if a sequence is not accepted, all its super sequence will be discarded.

- The features require multiple passes of the initial transaction dataset
- It uses the horizontal data format
- In each pass, the candidate's set is generated by a self-join of the patterns found in the previous pass
- In the k-pass, a sequence pattern is accepted only if all its *(k-1)* subpatterns are accepted in the *(k-1)* pass

The overview of GSP algorithm goes here.

$\mathcal{F}_1$ = { frequent 1-sequences };
for $(k = 2; \mathcal{F}_{k-1} \neq \emptyset; k = k + 1)$ **do**
  C_k = Set of candidate k-sequences;
  for all input-sequences $\mathcal{E}$ in the database **do**
    Increment count of all $\alpha \in C_k$ contained in $\mathcal{E}$
  $\mathcal{F}_k = \{\alpha \in C_k | \alpha.sup \geq min_sup\}$;
Set of all frequent sequences = $\bigcup_k \mathcal{F}_k$;

Here is the pseudocode:

```
GSP(D, Σ, MIN_SUP) {
  F ← φ;
  C^(1) ← {φ};
  for(each s ∈ Σ) {
    in C^(1), add s as child of _φ
    support_count(s) ← 0
  }
  k ← 1
while(C^(k) = φ){
      ComputSupport(C^(k), D)
      for(each leaf r ∈ C^(k)){
        if(support_count(r) ≥ MIN_SUP){
          F ← F ∪ {(r, support_count(r))}
        }else{
          remove r from C^(k)
        }
      }
      C^(k+1) ← ExtendPrefixTree(C^(k));
      k ← k + 1;
  }
  return F;
}
ComputeSupport(C^(k), D){
  for(each s_i ∈ D){
    for(each r ∈ C^(k)){
      if(r ⊆ s_i){
        support_count(r) ← support_count(r) + 1
      }
```

$$
\begin{aligned}
&21: ComputeSupport\left(C^{(k)}, D\right)\{\\
&22: \quad for\left(each\, s_i \in D\right)\{\\
&23: \quad\quad for\left(each\ r \in C^{(k)}\right)\{\\
&24: \quad\quad\quad if\left(r \subseteq s_i\right)\{\\
&25: \quad\quad\quad\quad support_{count(r)} \leftarrow support_{count(r)} + 1\\
&26: \quad\quad\quad \}\\
&27: \quad\quad \}\\
&28: \quad \}\\
&29: \}\\
&30: ExtendPrefixTree\left(C^{(k)}\right)\{\\
&31: \quad for\left(each\, leaf\ r_a \in C^{(k)}\right)\{\\
&32: \quad\quad for\left(each\, leaf\ r_b \in Children\left(Parent\left(r_a\right)\right)\right)\{\\
&33: \quad\quad\quad r_{ab} \leftarrow r_a + r_b\left[k\right]\\
&34: \quad\quad\quad if\left(r_c \in C^{(k).}, \forall r_c \subset r_{ab}, \left|r_c\right| = \left|r_{ab}\right| + 1\right)\{\\
&35: \quad\quad\quad\quad add\ r_{ab} as\ child\ of\ r_a\ with\ support_count\left(r_{ab}\right) \leftarrow 0\\
&36: \quad\quad\quad \}\\
&37: \quad\quad \}\\
&38: \quad\quad if\left(no\ extensions\ from\ r_a\right)\{\\
&39: \quad\quad\quad remove\ r_a\ from\ C^{(k)}\\
&40: \quad\quad \}\\
&41: \quad \}\\
&42: \quad return\ C^{(k)}\\
&43: \}
\end{aligned}
$$

The R implementation

Here is the R source code of the main algorithm:

```
GSP  <- function (d,I,MIN_SUP){
  f <- NULL
  c[[1]] <- CreateInitalPrefixTree(NULL)
  len4I <- GetLength(I)
  for(idx in 1:len4I){
    SetSupportCount(I[idx],0)
```

```
      AddChild2Node(c[[1]], I[idx],NULL)
    }
    k <- 1
    while( !IsEmpty(c[[k]]) ){
       ComputeSupportCount(c[[k]],d)
       while(TRUE){
         r <- GetLeaf(c[[k]])
         if( r==NULL ){
           break
         }
         if(GetSupport(r)>=MIN_SUP){
           AddFrequentItemset(f,r,GetSupport(r))
         }else{
           RemoveLeaf(c[[k]],s)
         }
      }
      c[[k+1]] <- ExtendPrefixTree(c[[k]])
      k <- K+1
    }
    f
  }
```

The SPADE algorithm

Sequential Pattern Discovery using Equivalent classes (**SPADE**) is a vertical sequence-mining algorithm applied to sequence patterns; it has a depth-first approach. Here are the features of the SPADE algorithm:

- SPADE is an extension of the A-Priori algorithm
- It uses the A-Priori property
- Multiple passes of the initial transaction data set are required
- The vertical data format is used
- It uses a simple join operation
- All sequences are found in three dataset passes

The short description of SPADE algorithm goes here.

Here is the pseudocode before calling the SPADE algorithm,
$F \leftarrow \phi, \wedge k \leftarrow 0, P \leftarrow \{< s, L(s) s > \in \Sigma, \text{support_count}(s) \geq \text{MIN_SUP}\}$:

```
SPADE (min_sup, D):
   F_1 = { frequent items or 1-sequences };
   F_2 = { frequent 2-sequences };
   E = { equivalence classes [X]_θ1 };
   for all [X] ∈ E do Enumerate-Frequent-Seq([X]);

Enumerate-Frequent-Seq(S):
   for all atoms A_i ∈ S do
      T_i = ∅;
      for all atoms A_j ∈ S, with j ≥ i do
         R = A_i ∨ A_j;
         if (Prune(R) == FALSE) then
            L(R) = L(A_i) ∩ L(A_j);
            if σ(R) ≥ min_sup then
               T_i = T_i ∪ {R}; F_|R| = F_|R| ∪ {R};
      end
      if (Depth-First-Search) then Enumerate-Frequent-Seq(T_i);
   end
   if (Breadth-First-Search) then
     for all T_i ≠ ∅ do Enumerate-Frequent-Seq(T_i);
```

The R implementation

Here is the R source code of the main algorithm:

```
SPADE  <- function (p,f,k,MIN_SUP){
  len4p <- GetLength(p)
  for(idx in 1:len4p){
     AddFrequentItemset(f,p[[idx]],GetSupport(p[[idx]]))
     Pa <- GetFrequentTidSets(NULL,MIN_SUP)
     for(jdx in 1:len4p){
       xab <- CreateTidSets(p[[idx]],p[[jdx]],k)
       if(GetSupport(xab)>=MIN_SUP){
         AddFrequentTidSets(pa,xab)
       }
     }
```

```
        if(!IsEmptyTidSets(pa)){
          SPADE(p,f,k+1,MIN_SUP)
        }
      }
    }
```

Rule generation from sequential patterns

Sequential rules, label sequential rules, and class sequential rules can be generated from sequential patterns, which you will get from the previous sequential patterns discovery algorithms.

High-performance algorithms

Along with the growth of the dataset size, there is a steady requirement for high-performance associations/patterns mining algorithms.

With the introduction of Hadoop and other MapReduce-like platforms to the world, there is a chance to meet these requirements. I will discuss this further in the upcoming chapters. Depending on the size of the dataset, some algorithms should be revised and adjusted, such as the recursive algorithm that will eventually run out of space on the call stack and might present a challenge when converting to MapReduce.

Time for action

To enhance your knowledge about this chapter, here are some practice questions that'll let you understand the concepts better:

- Write an R program to find how many unique items' names are contained in the given sample market basket transaction file. Map each item's name to a unique integer ID. Find out all the closed frequent itemsets. Find out all the maximal frequent itemsets and their support count. Set a support count threshold to various values yourself.
- Write an R program to implement the A-PrioriTid algorithm.

Summary

In this chapter, we looked at the following topics:

- Market basket analysis
- As the first step of association rule mining, the frequent itemset is the key factor. Along with the algorithm design, closed itemsets and maximum frequent itemsets are defined too.
- As the target of association rule mining, association rules are mined with the measure of support count and confidence. Correlation rules mining are mined with the correlation formulae, in addition to the support count.
- Monotonicity of frequent itemset; if an itemset is frequent, then all its subsets are frequent.
- The A-Priori algorithm, which is the first efficient mining algorithm to mine frequent patterns; many variants originated from it.
- Frequent patterns in sequence.

The next chapter will cover the basic classification algorithms, which is a major application of data mining, including ID3, C4.5, and CART.

3
Classification

In this chapter, you will learn the popular classification algorithms written in the R language. Empirical classifier performance and accuracy benchmarks are also included. Along with the introduction of various classification algorithms, b will also learn various ways to improve the classifier and so on.

Classification has massive applications in modern life. With the exponential growth of the information dataset, there is a need for high performance classification algorithms to judge an event/object belonging to a predefined categories set. Such algorithms have unlimited opportunity for implementation in a wide variety of industries such as bioinformatics, cybercrime, and banking. Successful classification algorithms use predefined categories from training information datasets to predict the unknown category for a single event given a common set of features.

Along with the continual growth of computer science, the classification algorithms need to be implemented on many diverse platforms including distributed infrastructure, cloud environment, real-time devices, and parallel computing systems.

In this chapter, we will cover the following topics:

- Classification
- Generic decision tree introduction
- High-value credit card customers classification using ID3
- Web spam detection using C4.5
- Web key resource page judgment using CART
- Trojan traffic identification method and Bayes classification
- Spam e-mail identification and Naïve Bayes classification
- Rule-based classification and the player types in computer games

Classification

Given a set of predefined class labels, the task of classification is to assign each data object of the input dataset with a label using the classifier's training model. Typically, the input can be a discrete or continuous value, but the output is discrete binary or nominal value and so forth. Classification algorithms are often described as learning models or functions, in which `x` is a tuple of attribute set with discrete or continuous value, and y is an attribute with discrete value such as categorical labels.

$$f\left(x,y\right)=0, x=\left(x_1,x_2,\ldots,x_n\right)$$

This function can also be treated as a classification model. It can be used to distinguish objects belonging to different classes or to predict the class of a new tuple or `y` in the above `(x, y)`. In another point of view, classification algorithms are targeted to find a model from the input data, and apply this model to future classification usage predictions when given a common set of attributes.

Generally speaking, $x=\left(x_1,x_2,\ldots,x_n\right)$ is a set of attributes selected as the input for the classification system. There are special algorithms used to select only the useful attributes from this set to ensure the efficiency of the classification system.

Almost any classification tasks need this preprocessing procedure, but the exact means vary from case to case. Here are three mainstream methods applied:

- Data cleaning
- Relevance analysis
- Data transformation and reduction

A standard classification process often includes two steps. The classification model with the higher accepted accuracy is accepted as classifier to classify a dataset in production. The following two steps are illustrated with an example in the diagram:

- **Training** (**supervised learning**): The classification model is built upon the training dataset, that is, the (instance, class label) pairs
- **Classification validation**: The accuracy of the model is checked with the test dataset to decide whether to accept the model

In the following sections, we will introduce some classification algorithms with different designs.

Generic decision tree induction

There are various definitions of the term **decision tree**. Most commonly, a decision tree provides a representation of the process of judging the class of a given data instance or record from the root node down to some leaf node. As a major classification model, the decision tree induction builds a decision tree as a classification model using the input dataset and class label pairs. A decision tree can be applied to various combinations of the following attribute data types, but is not limited to, including nominal valued, categorical, numeric and symbolic data, and their mixture. The following list is an illustration of Hunt's decision tree definition. The Step #7 applies a selected attribute test condition to partition the records to smaller datasets.

Step #1: *Let D_t be the training set related to node t;*

Step #2: *Let $(y_1, y_2, \ldots, y_m)$ be the class labels;*

Step #3: *if all records in D_t are mapped to the same class lable y_t{*

Step #4: *t is a leaf node with the lable of y_t*

Step #5: *}*

Step #6: *if(D_t contains records belong to more than one class){*

Step #7: *$D_t' \leftarrow AttributeTestCondition(D_t)$};*

Step #8: *For each $D_{tk} \in D_t'$, Apply the step #1~#6*

The decision tree is popular for its simplicity and low computational effort compared to other algorithms. Here are some characteristics of the decision tree induction:

- The greedy strategy is usually applied to the decision tree.
- It infers a decision tree once upon the entire training dataset.
- The algorithm requires no parameters to obtain the classification model from the input dataset.
- Like many other tasks, finding an optimal decision tree is an NP-complete problem.
- The algorithm to build the decision tree enables construction of the decision tree quickly. Tree construction is efficient, even upon large datasets.
- It provides an expressive way for discrete-valued functions.
- It is robust while opposed to noise.

- Using a top-down, recursive partition, the divide-and-conquer strategy is applied to most of the decision tree algorithms.
- The size of the sample dataset usually decreases dramatically when traversed down the tree.
- A subtree can be replicated many times in the decision tree.
- The test condition usually contains only one attribute.
- The performance of decision tree algorithms is affected by the impurity measure.

It is time to consider the decision tree when the instances in the source dataset are describable by the attribute-value pair, and the target function has discrete values, while the training dataset possibly has some noise.

An example of a decision tree built with the input dataset in a table (classical play golf dataset) is shown in the following diagram. The decision tree is composed of three entities or concepts: the root node, the internal node, and the leaf node. The leaf is given a class label. Nodes other than the leaf conduct tests on the attribute set to determine which input data belongs to which branch (child) of the node.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overoast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overoast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overoast	Mild	High	True	Yes
Overoast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Given a built decision tree, a test record can be classified easily. From the root node, apply the test condition of that node to the test record and go to the next node with the corresponding test result until a leaf, by which we can decide which class the test record belongs to, is reached.

Now there will be two issues. One is how to divide the training set as per a certain node while the decision induction tree grows according to a chosen test condition upon various attribute sets. This will be a question related to attribute selection measure, which are illustrated in the following section. The second but important issue is related to model overfitting.

There are two strategies for the termination of the growth of the limiting decision induction tree node. Using the naïve strategy, for a certain node, when all the data objects within a node are assigned to it belong to the same class or all records with the same attribute values; as a result, the node related will be assigned with the class label as the majority of training records within that node. The second strategy terminates the algorithm earlier, which is meant to avoid model overfitting and will be introduced in the tree pruning section.

Attribute selection measures

The node can have more than two children or branches depending on the attribute test condition and the selected attribute. To split the node, attribute selection measures with various implementations are applied. Attribute selection measures within the same node may also vary for binary branches or multiway branches. Some common attribute selection measures are the following:

- **Entropy**: This concept is used in information theory to describe the impurity of an arbitrary collection of data. Given the target attribute class set with size of *c*, and p_i as the proportion/probability of *S* belonging to class *i*, the definition is here, and the definition *Gain* is shown in the next point. Entropy always means how disordered a dataset is. The higher the value of entropy, the more the uncertainty shown by the source dataset.

$$Entropy(S) = \sum_{i=1}^{c} - p_i log_2 p_i$$

  The size and coverage of the training set assigned to a certain node affect the correctness of the following equations. The gain is better for those situations.

- **Gain**:

$$Gain(S, A) = Entropy(S) - \sum_{\upsilon \in Values(A)} \frac{|S_\upsilon|}{|S|} Entropy(S_\upsilon)$$

- **Gain Ratio**: This is applied in the C4.5 classification algorithm using the following formula:

$$GainRatio(S,T) = \frac{Gain(S,T)}{SplitInfo(S,T)}$$

- **Information Gain**: The ID3 algorithm uses this statistical property to decide which attribute is selected to be tested at any node in the tree, and measures the association between inputs and outputs of the decision tree.

  With the concept of information gain, the definition of a decision tree can be thought of in this way:

  - A decision tree is a tree-structured plan that uses a set of attribute tests to predict output
  - To decide which attribute should be tested first, simply find the one with the highest information gain
  - It, then, recurs

- **Gini Index**: It is used in the CART classification algorithm. The Gini index for a specific split point is calculated using the following equation. It is used to gauge the purity of the split point.

$$G(D) = 1 - \sum_{i=1}^{k} P(c_i \mid D)^2$$

- **Split Info**:

$$SplitInfo(S,T) = -\sum_{\upsilon \in Values(T_S)} \frac{|T_{S,\upsilon}|}{|T_S|} * log \frac{|T_{S,\upsilon}|}{|T_S|}$$

Tree pruning

The initial decision tree is often built with many branches reflecting outliers or noise, which are also common causes of model overfitting. Usually, the direct consequent in tree pruning is needed for the after-dealt of decision tree aiming, which is required for classifying higher accuracy or lower error rates. The two types of pruning in production are as follows:

- **Post-pruning**: This approach is to perform tree pruning after the tree grows to the maximum form. The cost-complexity pruning algorithm used in CART and the pessimistic pruning in C4.5 are both examples of post-pruning.
- **Pre-pruning**: This is also known as the early stopping strategy, which avoids an over-matured tree generation, to stop the growth of a tree earlier using additional restrictions, such as a threshold.

Repetition and replication are the two major factors that make decision trees unreasonably large and inefficient.

General algorithm for the decision tree generation

Here is the pseudocode of the general decision induction tree algorithm:

```
TreeGrowth(E,F) {
  if (StoppingCondition(E,F)) {
    leaf ← CreateNode()
    leaf.label ← Classify(E)
    return leaf
  }else{
    root ← CreateNode()
    root.test_cond ← FindBestSplit(E,F)
    V ← {v | v is a possible outcome of root.test_cond}
    for(each v ∈ V) {
      E_v ← {e|root.test_cond(e) = c, and e ∈ E }
      child ← TreeGrowth(E_v,F)
      AddDescendentAndLabelEdge(root,child,v)
    }
  }
  return root;
}
```

Another variety of the algorithm is described here, with input parameters as follows:

- D denotes the training data
- The leaf size is defined by η
- The leaf purity threshold is defined by π

The output of the algorithm is a decision tree, as shown in the following screenshot:

```
   DECISION TREE (D,η,r):
1  n ⟵ |D|
2  n_i ⟵ |{ X_j | X_j ∈ D, y_j = c_i }|
3  purity (D) ⟵ max_i { n_i / n }
4  if n ≤ η or purity (D) ≥ r then
5      c* ⟵ arg max_i { n_i / n }
6      create leaf node, and label it with class c*
7      return
8  ( split-point*, score* ) ⟵ (∅, 0)
9  foreach ( attribute X_j ) do
10     if ( X_j is numeric ) then
11         ( v, score ) ⟵ Evaluate-Numeric-Attribute ( D, X_j )
12         if score > score* then (split-point*, score*) ⟵ ( X_j ≤ v, score )
13     else if ( X_j is categorial ) then
14         ( V, score ) ⟵ Evaluate-Categorial-Attribute ( D, X_j )
15         if score > score* then ( split-point*, score* ) ⟵ ( X_j ∈ V, score )
16 D_Y ⟵ { X ∈ D | X satisfies split-point* }
17 D_N ⟵ { X ∈ D | X does not satisfies split-point* }
18 create internal node split-point*, with two child nodes, D_Y and D_N
19 DecisionTree( D_Y ); DecisionTree( D_N )
```

Line 1 denotes the partition size. Line 4 denotes the stopping condition. Line 9 through line 17 try to get the two branches with the new split. Finally, line 19 applies the algorithm recursively on the two new subbranches to build the subtree. This algorithm is implemented with R in the following section.

The R implementation

The main function of the R code for the generic decision tree induction is listed as follows. Here `data` is the input dataset, `c` is the set of class labels, `x` is the set of attributes, and `yita` and `pi` have the same definitions as in the previous pseudocodes:

```
DecisionTree <- function(data,c,x,yita,pi){
        result.tree <- NULL
        if( StoppingCondition(data,c,yita,pi) ){
```

```
                result.tree <- CreateLeafNode(data,c,yita,pi)
                return(result.tree)
        }

        best.split <- GetBestSplit(data,c,x)
        newdata <- SplitData(data,best.split)

        tree.yes <- DecisionTree(newdata$yes,c,x,yita,pi)
        tree.no <- DecisionTree(newdata$no,c,x,yita,pi)
        result.tree <- CreateInternalNode(data,
                   best.split,tree.yes,tree.no)

        result.tree
}
```

One sample dataset is chosen to verify the generic decision tree induction algorithm, the weather dataset. It is from the R package Rattle, which contains 366 examples of 23 attributes, and one target or the class label. In the R language, weather is a data frame, which contains 366 observations of 24 variables. The details for the dataset can be retrieved with the following R code:

```
> Library(rattle)
> str(weather)
```

High-value credit card customers classification using ID3

The **Iterative Dichotomiser 3** (**ID3**) algorithm is one of the most popular designs of the decision induction tree. It is not tolerant of missing values or noisy, and the value of attributes must come from an infinite fixed set.

ID3 uses entropy to calculate the homogeneity of a sample and also for the split. The information gain `G` for each attribute `A` is computed using the following equation. The root of the final tree is assigned with an attribute with the highest information gain. Then the new subtree is built recursively upon each value of the attribute bound to the root.

$$Gain(S, A) = Entropy(S) - \sum_{\upsilon \in Values(A)} \frac{|S_\upsilon|}{|S|} Entropy(S_\upsilon)$$

$$Entropy(S) = \sum_{i=1}^{c} - p_i log_2 p_i$$

With the play golf dataset as the input dataset, you can calculate the information gain and list using the following formulas:

- *Entropy (root) = 0.940*
- *Gain () = 0.048, Gain (S, Humidity) = 0.151*
- *Gain (S, Temperature) = 0.029, Gain (S, Outlook) = 0.246*

ID3 (C4.5 and CART) builds the decision induction tree recursively in a top-down divide-and-conquer manner through the space of possible decision trees with a greedy strategy. Using the greedy search strategy, at each step, a decision that greatly improves the optimizing target is made. For each node, find the test condition best segment the training data assigned to it.

The characteristics of the decision induction tree in the case of ID3 include the following:

- Each node excluding the leaf of the tree corresponds to an input attribute, each arc to a possible value of that attribute
- Entropy is used to determine how informative a particular input attribute is about the output class on a given dataset
- The recursive algorithm

A quick description about the recursive algorithm can be defined as follows:

- Break the original problem into two or more smaller-sized problems with the same type
- Call the recursive algorithm on each smaller type of problem
- Group together the results of step 2 to solve the original problem

The ID3 algorithm

The input parameters for ID3 algorithm are as follows:

- *I*, denotes the set of input attributes, which may be tested by the result decision tree
- *T*, the set of training data objects, or training examples

The output parameter of the algorithm is as follows:

- O, denotes the set of output attribute, that is, the value of those attributes will be predicted by the tree

Here is the pseudocode of the general algorithm:

```
ID3(I,O,T) {
  if (T is emputy) {
    node ← CreateNode()
    node.label ← "Failure"
    return node
  }
  if (all instance in T share the save value "sameValue" for O) {
    node ← CreateNode()
    node.label ← "sameValue"
    return node
}
if (I is emputy) {
    node ← CreateNode()
    node.label ← the value of the most common value of O in T
    return node
}
```

$17: Compute\, the\, information\, gain\ \ for\ each\ attribute\ in\ I\ \ relative\ to\ T$

$18:\ let\ x\ be\ the\ attribute\ with\ lest G\left(x,T\right) of\ \ the\ attribute\ in\ I$

$19: let\ \{x_j \mid j \in \left[1,m\right]\}\ be\, the\, value\, set\, of\ x$

$20: let\left\{T_j \mid j \in \left[1,m\right]\right\}_j be\, the\, set\, of\ subset\, of\ T\ after\, it\, is\ partitioned\, with\, value\, of\ x$

$21: root \leftarrow CreateNode()$

$22: root.label \leftarrow x$

$23:\ foreach\left(\ j\ in\left[1,m\right]\right)\ \{$

$24:\quad create\, a\, new\, branch\ for\, root$

$25:\quad childNode_j \leftarrow ID3\left(I-\left\{x\right\},O,T_j\right)$

$26:\quad if\ \left(T_j\ is\, empty\right)\{$

$27:\qquad leaf. \leftarrow CreateNode()$

$28:\qquad node.label \leftarrow the\, value\, of\ the\, most\, common\, value\, of\ O\, in\, T$

$29:\qquad add\, leaf\ as\, a\, leaf\ \ and\ \ the\ j-th\, child\, of\ root, under\ the\, new\, branch$

$30:\quad \}else\{$

$31:\qquad add\, childNode_j\, as\, the\ j-th\, child\, of\ root, under\ the\, new\, branch$

$32:\quad \}$

$33:\ \}$

$34: return\, root;$

$35: \}$

The R implementation

The main function of the R code for the ID3 algorithm is listed as follows. Here `data` is the input training dataset, `ix` is the set of input attributes, and `ox` is the output attribute:

```
ID3 <- function(data,ix,ox){
   result.tree <- NULL

   if( IsEmpty(data) ){
       node.value <- "Failure"
       result.tree <- CreateNode(node.value)
       return(result.tree)
   }
   if( IsEqualAttributeValue(data,ox) ){
       node.value <- GetMajorityAttributeValue(data,ox)
       result.tree <- CreateNode(node.value)
```

```
        return(result.tree)
    }
    if( IsEmpty(ix) ){
        node.value <- GetMajorityAttributeValue(data,ox)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    gain <- GetInformationGain(data,ix)
    best.split <- GetBestSplit(data,gain,ix)

    values <- GetAttributeValues(best.split)
    values.count <- GetAttributeValuesCount(best.split)
    data.subsets <- SplitData(data,best.split)

    node.value <- best.split
    result.tree <- CreateNode(node.value)
    idx <- 0
    while( idx<=values.count ){
        idx <- idx+1
        newdata <- GetAt(data.subsets,idx)
        value <- GetAt(values,idx)
        new.ix <- RemoveAttribute(ix,best.split)
        new.child <- ID3(newdata,new.ix,ox)
        AddChildNode(result.tree,new.child,value)
    }

    result.tree
}
```

Web attack detection

Along with the development of information technology, there have emerged many systems that identify malicious usage of the built software system, web system, and so on. One of them is the **Intrusion Detection System** (**IDS**), to detect the malicious behavior, conduct content inspection without the firewall. Also includes include signature detection, anomaly detection, and so on.

Classifier-like decision tree technologies, such as ID3, C4.5, and CART, play an important role as analyzers in addition to other important components of IDS, such as sensor, manager, operator, and administrator. The classifications needed here are activity monitor, file integrity checker, host firewall, log parser, and packet pattern matching.

Many issues occur for IDS. One of them is the new variety of a known attack pattern, often with low detection rate by the existing IDS. This drives the design of new types of IDS systems integrated with artificial intelligence, especially decision tree technologies.

Among real world examples, except the ones IDS has already built, there are also competitions for applying data mining techniques to web attack detection. One of them is KDD-Cup. The topic for KDD-Cup 1999 was Computer network intrusion detection, to build a classifier to predict the unauthorized behavior.

The dataset for it came from the DARPA Intrusion Detection Evaluation Program. More than five million data instances are contained in the training dataset and more than two million for test dataset. There are about 24 attack types in the training dataset, and 14 in the test dataset. Each data instance in the dataset contains 41 attributes, 9 for TCP connection, 13 for content features contained in the TCP connection, 9 for traffic features that use a two-second time window and the left for host-related traffic features. All the attacks can be categorized into the following four groups:

- **DOS**: This refers to denial of service
- **R2L**: This refers to unauthorized access to the local machine from the remote machine
- **U2R**: This refers to unauthorized access to local super-user privileges by a local unprivileged user
- **Probing**: This refers to surveillance and probing

By specific transformation, the ID3 algorithm can be applied to various web attack detection datasets with various sizes. When the size of the dataset increases, the performance of ID3 will be kept efficient by parallelization.

For simplicity, one example only takes the following four types of attacks to label a dataset for simple IDS:

- SQL Injection
- Cross Site Scripting
- Code Injection
- Directory Traversal

All the four types of attacks behave with a common pattern, the web queries with malicious pattern. Normalizing the web queries, the URL and collection of reserved tags, label-specific patterns with the appropriate label in the four types of attacks. After training ID3 on the dataset and applying it to the existing IDS, a better detection rate can be achieved.

High-value credit card customers classification

Following the growth of credit card usage, there has been a requirement in banking industry—finding high-value credit card customers from all customers to create a more customer-oriented strategy to increase profit. There are similar requirements such as finding interesting rules from the dataset.

To achieve this target, we need to enroll more correct customer attributes (no matter what type they are) to the training data object. The possible choices are transaction records, usage behaviors, customer age, annual income, education background, financial assets, and so on.

There is no need to include all customer-related attributes; the most key attributes on this target should be adopted. The domain experts might be helpful on this.

With the appropriate attributes selected, the ID3 algorithm can be applied here to finally extract sensitive features or representatives to help to judge which customer is more likely to be profitable.

Web spam detection using C4.5

C4.5 is an extension of ID3. The major extensions include handling data with missing attribute values, and handling attributes that belong to an infinite continuous range.

It is one of the decision tree algorithms, and is also a supervised learning classification algorithm. A model is learned and the input attribute values are mapped to the mutually exclusive class labels. Moreover, the learned model will be used to further classify new unseen instances or attribute values. The attribute select measure adopted in C4.5 is the gain ratio, which avoids the possible bias:

$$GainRatio(S,T) = \frac{Gain(S,T)}{SplitInfo(S,T)}$$

$$Gain(S, A) = Entropy(S) - \sum_{\upsilon \in Values(A)} \frac{|S_\upsilon|}{|S|} Entropy(S_\upsilon)$$

$$SplitInfo(S, T) = -\sum_{\upsilon \in Values(T_S)} \frac{|T_{S,\upsilon}|}{|T_S|} * log \frac{|T_{S,\upsilon}|}{|T_S|}$$

Based on the generic C4.5 algorithm, a suite for varieties derived, C4.5, C4.5-no-pruning, C4.5-rules, and so forth; all of them are called C4.5 algorithms, which means C4.5 is a suite of algorithms.

Compared to other algorithms, there are many important characteristics of C4.5:

- Tree pruning, a post-pruning strategy, is followed to remove some of the tree structure using selected accuracy criteria
- An improved use of continuous attributes
- Missing values handling
- Inducing rule sets
- It contains a multiway test that depends on the attribute value and is not just limited to the binary test
- Information theoretic tests are applied via the gain and gain ratio
- The greedy learning algorithm, that is, along with the tree growing, test best criteria test result is chosen
- The data fits in main memory (there are many extended algorithms that can use the secondary storages, such as BOAT, Rainforest, SLIQ, SPRINT, and so forth.)

The C4.5 algorithm

Here is the pseudocode for the basic C4.5 algorithm:

```
C4.5(T)
Input: training data set T; attributes S.
Output: decision tree Tree.
if T is NULL then
    return failure
end if
if S is NULL then
    return Tree as a single node with most frequent class label in T
end if
if |S| = 1 then
    return Tree as a single node S
end if
set Tree = {}
for a ∈ S do
    set Info(a,T)=0, and SplitInfo(a,T)= 0
    compute Entropy(a)
    for v ∈ values(a,T) do
        set T_{a,v} as the subset of T with attribute a = v
    Info(a,T) += |Ta,v|/|Ta| Entroph(a_v)
        SplitInfo(a,T) += |Ta,v|/|Ta| log |Ta,v|/|Ta|
end for
Gain(a,T) = Entropy(a) − Info(a,T)
GainRatio(a,T) = Gain(a,T) / SplitInfo(a,T)
end for
set a_best = argmax_a {GainRatio(a,T)}
attach a_best into Tree
for v ∈ values(a_best, T) do
    call C4.5(Ta,v)
end for
return Tree
```

The R implementation

The R code for the ID3 is listed as follows:

```
C45 <- function(data,x){
    result.tree <- NULL

    if( IsEmpty(data) ){
        node.value <- "Failure"
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    if( IsEmpty(x) ){
        node.value <- GetMajorityClassValue(data,x)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }
    if( 1 == GetCount(x) ){
        node.value <- GetClassValue(x)
        result.tree <- CreateNode(node.value)
        return(result.tree)
    }

    gain.ratio <- GetGainRatio(data,x)
    best.split <- GetBestSplit(data,x,gain.ratio)

    data.subsets <- SplitData(data,best.split)
    values <- GetAttributeValues(data.subsets,best.split)
    values.count <- GetCount(values)

    node.value <- best.split
    result.tree <- CreateNode(node.value)
    idx <- 0
    while( idx<=values.count ){
        idx <- idx+1
        newdata <- GetAt(data.subsets,idx)
        value <- GetAt(values,idx)
        new.x <- RemoveAttribute(x,best.split)
        new.child <- C45(newdata,new.x)
        AddChildNode(result.tree,new.child,value)
    }

    result.tree
}
```

A parallel version with MapReduce

With the increase in the dataset volume or size, the C4.5 algorithm can be parallelized according to the MapReduce algorithm, the Hadoop technologies suite, and especially via RHadoop for R codes.

The MapReduce programming model is illustrated in the following diagram:

Data Preparation

1: **procedure** MAP_Attribute $(row_id, (a_1, a_2, ..., a_M, c))$
2: emit $(a_j, (row_id, c))$
3: **end procedure**
4: **procedure** REDUCE_ATTRIBUTE $(a_j, (row_id, c))$
5: emit $(a_j, c, cnt))$
6: **end procedure**

Attribute Selection

1: **procedure** REDUCE_POPULATION $a_j, c, cnt)))$
2: emit (a_j, all)
3: **end procedure**
4: **procedure** MAP_COMPUTATION $((a_j, (c, cnt, all)))$
5: compute $Entropy\ (a_j)$
6: compute $Info(a_j) = -\frac{cnt}{all}\ Entropy\ (a_j)$
7: compute $SplitInfo(a_j) = -\frac{cnt}{all}\ log\ \frac{cnt}{all}$
8: emit $(a_j, Info(a_j), SplitInfo(a_j))$
9: **end procedure**
10: **procedure** REDUCE_COMPUTATION $((a_j, (Info(a_j), SplitInfo(a_j)))$
11: emit $(a_j, GainRatio(a_j))$
12: **end procedure**

Update Tables

```
procedure MAP_UPDATE_COUNT (( a_best , (row_id, c)))
        emit ( a_best , (c,cnt' ))
end procedure
procedure MAP_HASH (( a_best , row_id ))
        compute node_id=hash (a_best )
        emit (row_id, node_id)
end procedure
```

Tree Growing

```
procedure MAP (( a_best , (row_id ))
        compute node_id=hash (a_best )
        if node_id is same with the old value then
        emit ( row_id, nod_id )
        end if
        add a new subnode
        emit (row_id, node_id, subnode_id )
    end procedure
```

Web spam detection

Spamming occurs along with the emergence of search engine technologies to pursue higher rank with the deceiving search engines relevancy algorithm, but not improve their own website technologies. It performs deliberate actions to trigger unjustifiable favorable relevance or importance for a specific web page, in contrast to the page's true value or merit. The spam page finally receives a substantial amount of score from other spam pages to boost its rank in search engine results by deliberately manipulating the search engine indexes. Finally, traffic is driven to the spammed pages. As a direct result of the Web spam, the information quality of the Web world is degraded, user experience is manipulated, and the security risk for use increases due to exploitation of user privacy.

One classic example, denoted as link-farm, is illustrated in the following diagram, in which a densely connected set of pages is created with the target of cheating a link-based ranking algorithm, which is also called **collusion**:

There are three major categories of spam from a business point of view:

- Page Spoofing
- Browser-Based Attacks
- Search Engine Manipulations

There are three major categories of spam from a technology point of view:

- **Link spam**: This consists of the creation of the link structure, often a tight-knit community of links, targeted at affecting the outcome of a link-based ranking algorithm. Possible technologies include honey pot, anchor-text spam, blog/wiki spam, link exchange, link farm, and expired domains.
- **Content spam**: This crafts the contents of web page pages. One example is inserting unrelated keywords to the web page content for higher ranking on search engines. Possible technologies include hidden text (size and color), repetition, keyword stuffing/dilution, and language-model-based technologies (phrase stealing and dumping).

- **Cloaking**: This sends content to search engines which looks different from the version viewed by human visitors.

The link-based spam detections usually rely on automatic classifiers, detecting the anomalous behavior of a link-based ranking algorithm, and so on. Classifier or language model disagreement can be adopted to detect content spam. While the cloaking detection solution is inherent, one of them is comparing the indexed page with the pages visitors saw.

How to apply a decision tree for web spam detection? The links and contents of the web spam, after statistical analysis, are unique compared to other normal pages. Some properties are valuable for detecting spam, trustworthiness of the page, neutrality (facts), and bias. Web spam detection can be a good example for illustrating the C4.5 algorithm.

Some domain-related knowledge can be applied to the classification solution. One observed phenomenon is that bad links always have links between them. The links between web pages and websites are often not randomly set but with certain rules and they can be detected by classifiers.

About the attributes for a certain data point, the dataset can be divided into two groups, link-based features and content-based features:

- **Link-based features**: These include degree-based measures such as in-degree and out-degree of the web hosts. The second is PageRank-based features, which is an algorithm to compute a score for every page. The third is TrustRank-based features, which measure the trustworthiness of certain web pages to some benchmarked, trustworthy web pages.

- **Content-based features**: These include the number of words in the page, number of words in the title, average word length, amount of anchor text, fraction of anchor text, fraction of visible text, fraction of page drawn from globally popular words, fraction of globally popular words, compressibility, corpus precision and corpus recall, query precision and query recall, independent trigram or n-gram likelihood, and entropy of trigrams or n-gram.

All these features are included in one data point to set up a preprocessed dataset, which in turn can be used by classification algorithms, especially decision tree algorithms such as C4.5 to work as web spam classifiers to distinguish spam from normal pages. Among all the classification solutions, C4.5 achieves the best performance.

Web key resource page judgment using CART

Classification and Regression Trees (**CART**) is one of the most popular decision tree algorithms. It is a binary recursive partitioning algorithm that can be used to process continuous and nominal attributes.

There are three main steps in the CART algorithm. The first is to construct the maximum tree (binary tree). The second step is to choose the right size of the tree. The last step is to classify new data using the result tree.

Compared to other algorithms, there are many important characteristics of CART:

- Binary decision tree (a binary recursive partitioning process)
- The source dataset can have continuous or nominal attributes
- No stopping rule (unless no possible splits are available)
- Tree pruning with cost-complexity pruning
- Nonparametric
- No variables to be selected in advance
- The missing value is dealt with an adaptive and better strategy
- The outlier can be easily handled
- No assumptions
- Computationally fast
- At each split point, only one variable is used

- Only one optimal tree is taken as the result tree, which is formed from a sequence of nested pruned candidate trees generated by CART
- Automatically handling the missing value in the source dataset

The weighted Gini index equation is defined as follows:

$$G\left(D_Y, D_N\right) = \frac{n_Y}{n} G\left(D_Y\right) + \frac{n_N}{n} G\left(D_N\right)$$

The CART measure is different; the goodness of the split point is proportional to the value of measure. The higher the better.

$$CART\left(D_Y, D_N\right) = 2\frac{n_Y}{n}\frac{n_N}{n}\sum_{i=1}^{k} \left| P\left(c_i | D_Y\right) - P(c_i \mid D_N) \right|$$

The CART algorithm

Splitting rules, with the omission of the following parts, is too lengthy to include in this section:

- Separate handling of continuous and categorical splitters
- Special handling for categorical splitters with multiple levels
- Missing value handling
- Tree pruning
- Tree selection

Here is the pseudocode for the CART algorithm, the simplified tree-growing algorithm:

```
BEGIN:  Assign all training data to the root node
        Define the root node as a terminal node

SPLIT:
New_splits=0
FOR every terminal node in the tree:
   If the terminal node sample size is too small or all instances in the
   node belong to the same target class goto GETNEXT
   Find the attribute that best separates the node into two child nodes
   using an allowable splitting rule
   New_splits+1
GETNEXT:
NEXT
```

The simplified pruning algorithm is as follows:

```
DEFINE:  r(t)= training data misclassification rate in node t
         p(t)= fraction of the training data in node t
         R(t)= r(t)*p(t)
         t_left=left child of node t
         t_right=right child of node t
         |T| = number of terminal nodes in tree T

BEGIN:   Tmax=largest tree grown
         Current_Tree=Tmax
         For all parents t of two terminal nodes
           Remove all splits for which R(t)=R(t_left) + R(t_right)
         Current_tree=Tmax after pruning

PRUNE:   If |Current_tree|=1 then goto DONE
         For all parents t of two terminal nodes
         Remove node(s) t for which R(t)-R(t_left) - R(t_right)
             is minimum
Current_tree=Current_Tree after pruning
```

The R implementation

Please look up the R codes file `ch_02_cart.R` from the bundle of R codes for the previously mentioned algorithms. One example is chosen to apply the CART algorithm to, in the following section.

Web key resource page judgment

Web key resource judgment arises from the domain of web information retrieval and web search engines. The original concept is from the authority value, the hub value, and the HITS algorithm. During queries of information from IR systems or search engines, finding important and related information from an overwhelmingly increasing volume of information is a challenging task. A better judgment leads to less indexing storage and a more informative querying result.

A key resource page is a high quality web page with much more information per selected topic compared to an ordinary web page on the same topic. In order to measure the importance of a certain web page, feature selection is the first thing required in the design.

The link-based features used in current search technologies can't resolve such issues at an acceptable accuracy rate. To improve the accuracy rate, more global information across many data instances can be adopted in addition to the single-data-instance-related attributes or features, which means local attributes.

Experimental results show that the key web page should contain in-site out-links with anchor text to other pages. Non-content attributes, such as web page links related attributes and content structures of pages, can be applied to judge key resource pages. The possible attributes are listed as follows:

- **In-degree or in-links**: This denotes the number of links pointing to the page. Observation shows that the higher the number of in-links related to the key page, the more the links from other sites to that page, which means more recommendations to a certain extent.
- **URL length or the depth of a page's URL**: There are four types of URLs defined in the following box: root, subroot, path, and filename. The four types of URLs map to four levels of length, that is, 1 to 4 respectively. A lower prior probability with a lower level and a higher prior probability mean a bigger possibility to be a key resource page.
- **In-site out-link anchor text rate**: This refers to the rate of the length of the anchor text to the document or page content length.
- **In-site out-link number**: This refers to the number of links embedded in a page.
- **Document length (in words)**: This filters out specified predefined non-usable characters from the document. This attribute can predict the relevance of a page because of the non-uniform distribution.

With the attributes just mentioned, the uniform sampling problem can be bypassed to a certain extent. The dataset can be easily built and used by decision tree induction algorithms such as CART.

Trojan traffic identification method and Bayes classification

Among probabilistic classification algorithms is the Bayes classification, which is based on Bayes' theorem. It predicts the instance or the class as the one that makes the posterior probability maximal. The risk for Bayes classification is that it needs enough data to estimate the joint probability density more reliably.

Given a dataset D with a size n, each instance or point x belonging to D with a dimension of *m*, for each $x_i = (x_{i1}, x_{i2}, \ldots, x_{im},) \in D, y_i \; is \; the \; class \; for \; x_i, y_i \in \{c_1, c_2, \ldots, c_k\}$. To predict the class y' of any x, we use the following formula:

$$\hat{y} = arg\,max\{P(c_i|x)|\forall i\}$$

Basing on Bayes' theorem, $P\ (x\,|\,c_i)$ is the likelihood:

$$P(c_i|x) = \frac{P(x\,|\,c_i)P(c_i)}{P(x)} \quad ,P(x) = \sum_{j=1}^{k} P(x|c_j)P(c_j)$$

Then we get the following new equations for predicting y' for x:

$$\hat{y} = arg\,max\{P(c_i|x)|\forall i\} = arg\,max\left\{\frac{P(x|c_i)P(c_i)}{P(x)}\right\} = arg\max\{P(x\,|\,c_i)P(c_i)\}$$

Estimating

With new definitions to predict a class, the prior probability and its likelihood needs to be estimated.

Prior probability estimation

Given the dataset D, if the number of instances in D labeled with class c_i is n_i and the size of D is n, we get the estimation for the prior probability of the class c_i as follows:

$$\hat{P}(c_i) = \frac{n_i}{n}$$

Likelihood estimation

For numeric attributes, assuming all attributes are numeric, here is the estimation equation. One presumption is declared: each class c_i is normally distributed around some mean μ_i with the corresponding covariance matrix $\sum_i$. $\hat{\mu}_i$ is used to estimate μ_i, $\hat{\sum}_i$ for $\sum_i$:

$$\hat{y} = arg\max\{f_i(x)P(c_i) \mid \forall i\}$$

$$f_i(x) = f(x|\mu_i, \Sigma_i) = \frac{1}{\left(\sqrt{2\pi}\right)^d \sqrt{|\Sigma_i|}} \exp\left\{-\frac{(x-\mu_i)^T \sum_i^{-1}(x-\mu_i)}{2}\right\}$$

$$\hat{\mu}_i = \frac{1}{n_i} \sum_{x_j \in D_i} x_j$$

$$\hat{\Sigma}_i = \frac{1}{n_i} Z_i^T Z_i$$

For categorical attributes, it can also be dealt with similarly but with minor difference.

The Bayes classification

The pseudocode of the Bayes classification algorithm is as follows:

```
BayesClassifier (D = {(x_j, y_j)}_{j=1}^n):
for i = 1, ··· , k do
    D_i ← {x_j | y_j = c_i, j = 1, ··· , n}
    n_i ← |D_i|
    P̂(c_i) ← n_i/n
    μ̂_i ← (1/n_i) Σ_{x_j ∈ D_i} x_j
    Z_i ← D_i − 1_{n_i} μ̂_i^T
    Σ̂_i ← (1/n_i) Z_i^T Z_i
return P̂(c_i), μ̂_i, Σ̂_i for all i = 1, ··· , k
Testing (x and P̂(c_i), μ̂_i, Σ̂_i, for all i ∈ [1, k]):
ŷ ← arg max_i {f(x|μ̂_i, Σ̂_i) · P(c_i)}
return ŷ
```

The R implementation

The R code for the Bayes classification is listed as follows:

```
BayesClassifier <- function(data,classes){
    bayes.model <- NULL

    data.subsets <- SplitData(data,classes)
    cards <- GetCardinality(data.subsets)
    prior.p <- GetPriorProbability(cards)
    means <- GetMeans(data.subsets,cards)
   cov.m <-GetCovarianceMatrix(data.subsets,cards,means)

    AddCardinality(bayes.model,cards)
    AddPriorProbability(bayes.model,prior.p)
    AddMeans(bayes.model,means)
    AddCovarianceMatrix(bayes.model,cov.m)

    return(bayes.model)
}

TestClassifier <- function(x){
    data <- GetTrainingData()
    classes <- GetClasses()
    bayes.model <- BayesClassifier(data,classes)

    y <- GetLabelForMaxPostProbability(bayes.model,x)

    return(y)
}
```

One example is chosen to apply the Bayes classification algorithm, in the following section.

Trojan traffic identification method

A Trojan horse, which is a malicious program, surreptitiously performs its operation under the guise of a legitimate program. It has a specific pattern and unique malicious behavior (such as traffic and other operations). For example, it may obtain account information and sensitive system information for further attacks. It can also fork processes for dynamic ports, impersonate software and redirect traffic of affected services to other systems, make them available to attackers to hijack connections, intercept valuable data, and inject fake information or phishing.

Depending on the purpose of Trojans, there are many versatile types of designs for Trojans, each with a certain traffic behavior. With the ability to identify the Trojan traffic, further processing can be performed to protect information. As a result, detecting the traffic of Trojans is one of the main tasks to detect Trojans on system. The behavior of Trojans is an outlier compared to the normal software. So the classification algorithms such as the Bayesian classification algorithm can be applied to detect the outliers. Here is a diagram showing the Trojan traffic behavior:

The malicious traffic behaviors include but are not limited to spoofing the source IP addresses and (short and long) term scanning the flow of the address/port that serves as the survey for successive attacks. Known Trojan traffic behaviors are used as the positive training data instances. The normal traffic behaviors are used as the negative data instances in the training dataset. These kinds of datasets are continuously collected by NGOs.

The attributes used for a dataset may include the latest DNS request, the NetBIOS name table on the host machine, ARP cache, intranet router table, socket connections, process image, system ports behavior, opened files updates, remote files updates, shell history, packet TCP/IP headers information, identification fields (IPID) of the IP header, **Time To Live** (**TTL**), and so forth. One possible attribute set for a dataset is source IP, port, target IP, target port, number of flows, number of packets, number of bytes, timestamp at certain checkpoint, and the class label for the type of detection. The DNS traffic plays an important role in the Trojans' detection too; the traffics of Trojans has certain a relation with DNS traffic.

The traditional technologies for detecting a Trojan often rely on the Trojan's signature and can be deceived by dynamic ports, encrypted messages, and so on. This led to the introduction of mining technologies for the classification of Trojan traffic. The Bayesian classifier is one of the better solutions among others. The preceding diagram is one such possible structure.

Identify spam e-mail and Naïve Bayes classification

The Naïve Bayes classification presumes that all attributes are independent; it simplifies the Bayes classification and doesn't need the related probability computation. The likelihood can be defined with the following equation:

$$P\left(x|c_i\right) = P\left(x_1, x_2, .., x_d|c_i\right) = \prod_{j=1}^{d} P(x_j \mid c_i)$$

Some of the characteristics of the Naïve Bayes classification are as follows:

- Robust to isolated noise
- Robust to irrelevant attributes
- Its performance might degrade due to correlated attributes in the input dataset

The Naïve Bayes classification

The pseudocode of the Naïve Bayes classification algorithm, with minor differences from the Bayes classification algorithm, is as follows:

$$
\begin{array}{l}
\textbf{NaiveBayes } (\mathbf{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{n}): \\
\textbf{for } i = 1, \cdots, k \textbf{ do} \\
\quad \mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \cdots, n\} \\
\quad n_i \leftarrow |\mathbf{D}_i| \\
\quad \hat{P}(c_i) \leftarrow n_i / n \\
\quad \hat{\boldsymbol{\mu}}_i \leftarrow \frac{1}{n_i} \sum_{\mathbf{x}_j \in \mathbf{D}_i} \mathbf{x}_j \\
\quad \mathbf{Z}_i = \mathbf{D}_i - \mathbf{1} \cdot \hat{\boldsymbol{\mu}}_i^T \\
\quad \textbf{for } j = 1, .., d \textbf{ do} \\
\quad\quad \hat{\sigma}_{ij}^2 \leftarrow \frac{1}{n_i} Z_{ij}^T Z_{ij} \\
\quad \hat{\boldsymbol{\sigma}}_i = \left(\hat{\sigma}_{i1}^2, \ldots, \hat{\sigma}_{id}^2\right)^T \\
\textbf{return } \hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i \textit{ for all } i = 1, \cdots, k \\
\\
\textbf{Testing } (\mathbf{x} \textbf{ and } \hat{P}(c_i), \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\sigma}}_i, \textbf{ for all } i \in [1, k]): \\
\hat{y} \leftarrow \arg\max_i \left\{ \hat{P}(c_i) \prod_{j=1}^{d} f(x_j | \hat{\mu}_{ij}, \hat{\sigma}_{ij}^2) \right\} \\
\textbf{return } \hat{y}
\end{array}
$$

The R implementation

The R code for the Naïve Bayes classification is listed as follows:

```
NaiveBayesClassifier <- function(data,classes){
    naive.bayes.model <- NULL

    data.subsets <- SplitData(data,classes)
    cards <- GetCardinality(data.subsets)
    prior.p <- GetPriorProbability(cards)
    means <- GetMeans(data.subsets,cards)
    variances.m <- GetVariancesMatrix
      (data.subsets,cards,means)

    AddCardinality(naive.bayes.model,cards)
    AddPriorProbability(naive.bayes.model,prior.p)
    AddMeans(naive.bayes.model,means)
    AddVariancesMatrix(naive.bayes.model,variances.m)

    return(naive.bayes.model)
}

TestClassifier <- function(x){
    data <- GetTrainingData()
    classes <- GetClasses()
    naive.bayes.model <- NaiveBayesClassifier(data,classes)

    y <- GetLabelForMaxPostProbability(bayes.model,x)

    return(y)
}
```

One example is chosen to apply the Naïve Bayes classification algorithm, in the following section.

Identify spam e-mail

E-mail spam is one of the major issues on the Internet. It refers to irrelevant, inappropriate, and unsolicited emails to irrelevant receivers, pursuing advertisement and promotion, spreading malware, and so on.

> *Unsolicited, unwanted e-mail that was sent indiscriminately, directly or indirectly, by a sender having no current relationship with the recipient* is the formal definition of e-mail spam from spam track at the **Text Retrieval Conference** (**TREC**).

The increase in e-mail users, business e-mail campaigns, and suspicious usage of e-mail have resulted in a massive dataset of spam e-mails, which in turn necessitate high-efficiency solutions to detect e-mail spam:

E-mail spam filters are automated tools that recognize spam and prevent further delivery. The classifier serves as a spam detector here. One solution is to combine inputs of a couple of e-mail spam classifiers to present improved classification effectiveness and robustness.

Spam e-mail can be judged from its content, title, and so on. As a result, the attributes of the e-mails, such as subject, content, sender address, IP address, time-related attributes, in-count /out-count, and communication interaction average, can be selected into the attributes set of the data instance in the dataset. Example attributes include the occurrence of HTML form tags, IP-based URLs, age of link-to domains, nonmatching URLs, HTML e-mail, number of links in the e-mail body, and so on. The candidate attributes include discrete and continuous types.

The training dataset for the Naïve Bayes classifier will be composed of the labeled spam e-mails and legitimate e-mails.

Rule-based classification of player types in computer games and rule-based classification

Compared to other classification algorithms, the learned model for a rule-based classification is set up by an IF-THEN rules set. The rules set can be transformed from the decision tree or by the following algorithm. An IF-THEN rule has the following format:

```
IF condition_holds_true THEN make_a_conclusion
```

An alternative format is as follows:

$$RULE\ Antecedent \leftarrow RULE\ Consequent$$

For a given instance or record in the source dataset, if the `RULE` antecedent holds true, the rule is defined to cover the instance, and it is satisfied.

Given a rule `R`, the coverage and accuracy are defined as follows:

$$Coverage(R) = \frac{|D_{cover}|}{|D|}$$

$$Accuracy(R) = \frac{|D_{correct}|}{|D_{cover}|}$$

Transformation from decision tree to decision rules

It is very convenient to transform the decision tree into a decision rules set for further processing. Along with every path from the root to a leaf in the decision tree, a rule can be written. The left-hand side, the rule antecedent, of any rule is constructed by the combination of the label of the nodes and the labels of the arcs, then the rule consequent by the leaf node. One example of extracting classification rules from the decision tree is illustrated in the following diagram:

One important question is the pruning of the resulting rules set.

Rule-based classification

Rules are learned sequentially, one at a time. Here is the pseudocode of the algorithm to build a rule-based classifier. The `LearnOneRule` function is designed with the greedy strategy. Its target is to cover the positive instance in the source dataset as much as possible, and none or as few as possible of the negative instance at the same time. All the instances in the source dataset with a specific class are defined as positive, and those that belong to other classes are considered to be negative. An initial rule `r` is generated, which keeps refining until the stop condition is met.

Sequential covering algorithm

The pseudocode of the generic sequential covering algorithm is as follows. The input parameters include the dataset with class-labeled tuples and the attributes set with all of their possible values. The output is a set of IF-THEN rules as follows:

Sequential covering

Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set of class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

```
Rule_set = {};
for each class c do
    repeat
        Rule = Learn_One_Rule(D, Att_vals, c);
        remove tuples covered by Rule from D;
        Rule_set = Rule_set + Rule;
    until terminating condition;
endfor
return Rule_Set;
```

The RIPPER algorithm

Repeated Incremental Pruning to Produce Error Reduction (**RIPPER**) is a direct rule-based classifier, in which the rule set is relatively convenient to interpret and the most practical for imbalance problems.

As per the growth of a rule, the algorithm starts from an empty rule and adds conjuncts, which maximize or improve the information gain measure, that is, the FOIL. It stops at the situation so that the rule does not cover negative rules. The resulting rule is pruned immediately with incremental reduced error pruning. Any final sequence of conditions is removed once it maximizes the measure of pruning v, which is calculated as follows:

$$v = \frac{p-n}{p+n}$$

The sequential covering algorithm is used to build a rule set; the new **description length** (**DL**) is computed once a new rule is added to the rule set. The rule set is then optimized.

Given are p as the number of positive examples covered by this rule and n as the number of negative rules covered by this rule. P denotes the number of positive examples of this class, and N the number of the negative examples of this class.

$$G = p\left[\log\left(\frac{p}{t}\right) - \log\left(\frac{P}{T}\right)\right]$$

$$W = \frac{p+1}{t+2}$$

$$A = \frac{p+n'}{T}, current\ rule's accuracy$$

The pseudocode of the RIPPER algorithm is as follows:

```
Initialize E to the instance set
For each class C, from smallest to largest
    BUILD:
        Split E into Growing and Pruning sets in the ratio 2:1
        Repeat until (a) there are no more uncovered examples of C; or (b) the
             description length (DL) of ruleset and examples is 64 bits greater
             than the smallest DL found so far, or (c) the error rate exceeds
             50%:
           GROW phase: Grow a rule by greedily adding conditions until the rule
             is 100% accurate by testing every possible value of each attribute
             and selecting the condition with greatest information gain G
           PRUNE phase: Prune conditions in last-to-first order. Continue as long
             as the worth W of the rule increases
    OPTIMIZE:
        GENERATE VARIANTS:
        For each rule R for class C,
           Split E afresh into Growing and Pruning sets
           Remove all instances from the Pruning set that are covered by other
             rules for C
           Use GROW and PRUNE to generate and prune two competing rules from the
             newly-split data:
               R1 is a new rule, rebuilt from scratch;
               R2 is generated by greedily adding antecedents to R.
           Prune using the metric A (instead of W) on this reduced data
        SELECT REPRESENTATIVE:
        Replace R by whichever of R, R1 and R2 has the smallest DL.
    MOP UP:
        If there are residual uncovered instances of class C, return to the
             BUILD stage to generate more rules based on these instances.
    CLEAN UP:
        Calculate DL for the whole ruleset and for the ruleset with each rule in
             turn omitted; delete any rule that increases the DL
        Remove instances covered by the rules just generated
Continue
```

The R implementation

The R code for the rule-based classification is listed as follows:

```
SequentialCovering <- function(data,x,classes){
    rule.set <- NULL

    classes.size <- GetCount(classes)
     idx <- 0
    while( idx <= classes.size ){
        idx <- idx+1
        one.class <- GetAt(classes,idx)
        repeat{
            one.rule <- LearnOneRule(newdata,x,one.class)
            data <- FilterData(data,one.rule)
            AddRule(rule.set,one.rule)
            if(CheckTermination(data,x,classes,rule.set)){
                break;
            }
        }
    }
    return(rule.set)
}
```

One example is chosen to apply the rule-based classification algorithm, in the following section.

Rule-based classification of player types in computer games

During computer game progressing and in the game context, improving the experience of a game is always a continual task. Classification of player types is one major task, which in turn brings more improvements including game design.

One of the popular player models of typological of temperature is the DGD player topology, which is illustrated in the following diagram. Given this model, the game players can be labeled with appropriate types, the game can be explained, it helps in designing new games, and so forth.

Based on the player behaviors or models, we can train the decision tree model with the dataset, and the rules set from the trained decision tree model. The dataset will come from the game log and some predefined domain knowledge.

Time for action

Here are some practices for you to check what you've learned so far:

- Running the R code of the ID3 algorithm step by step upon a minor dataset to trace the values of the important factors at each step
- Preparing the dataset related to web logs and creating an application that detects web attacks using ID3
- Implementing an R code to generate decision rules from a decision tree
- What is Gain Ratio?

Summary

In this chapter, we learned the following facts:

- Classification is a class of dispatch instances to one of predefined categories
- Decision tree induction is to learn the decision tree from the source dataset with the (instance and class-label) pairs under the supervised learning mode
- ID3 is a decision tree induction algorithm
- C4.5 is an extension of ID3
- CART is a decision tree induction
- Bayes classification is a statistical classification algorithm
- Naïve Bayes classification is a simplified version of Bayes classification in which there is a presumption of independence
- Rule-based classification is a classification model applying the rule set, which can be collections by direct algorithm, the sequential covering algorithm, and the indirect method by decision tree transforming

In the next chapter, you'll cover the more-advanced classification algorithms, including Bayesian Belief Network, SVM, k-Nearest Neighbors algorithm, and so on.

4
Advanced Classification

In this chapter, you will learn about the top classification algorithms written in the R language. You will also learn the ways to improve the classifier.

We will cover the following topics:

- Ensemble methods
- Biological traits and Bayesian belief network
- Protein classification and the k-Nearest Neighbors algorithm
- Document retrieval and Support Vector Machine
- Text classification using sentential frequent itemsets and classification using frequent patterns
- Classification using the backpropagation algorithm

Ensemble (EM) methods

To improve the accuracy of classification, EM methods are developed. The accuracy is dramatically improved by at least one grade compared to its base classifiers, because the EM methods make mistakes only when at least half of the result of the base classifiers are wrong.

The concept structure of EM methods is illustrated in the following diagram:

The label for the new data tuple is the result of the voting of a group of base classifiers. A combined classifier is created based on several base classifiers. Each classifier is trained with a different dataset or training set re-sampled with the replacement of the original training dataset.

Three popular EM methods are discussed in the successive sections:

- Bagging
- Boosting
- Random forests

The bagging algorithm

Here is a concise description of the bagging algorithm (noted as the bootstrap aggregation), followed by the summarized pseudocode. For iteration i ($i \in [1, k]$), a training set, D_i, of d tuples is sampled with replacement from the original set of tuples, D. Any training set is sampled by employing bootstrap sampling (with replacement) for it, which in turn is used to learn a classifier model, M_i. To classify an unknown or test tuple, X, each classifier, M_i, returns its class prediction, which counts as one vote. Assume the number of classifiers as follows, which predicts the same class, c_j, given the test tuple X:

$$v_j(X) = \left| \{ M_i(x) = \mathrm{c}_j \,\middle|\, i \in [1, K] \} \right|$$

The bagged classifier, M_*, counts the votes and assigns the class with the most votes to X. Each vote has the same weight in the equation;

$$M_*(X) = \underbrace{argmax}_{c_j}\{v_j(X) | j \in [1, K]\}$$

About the prediction of continuous values, the average value of each prediction for a given test tuple is used as the result. The algorithm reduces the variance, given a more correct result than the base classifiers.

The input parameters for bagging algorithm are as follows:

- D: This is the training tuples dataset
- K: This is the number of classifiers combined
- S: This is a classification learning algorithm or scheme to learning base classifier
- M_*: This is the ensemble classifier, which is the output of the algorithm

The summarized pseudocode for the bagging algorithm is as follows:

```
Bagging (D, K, S) {
  for (j ← 1; j ≤ K; j++) {
    Create bootstrap sample, Dj, by sampling D with replacement
    Learning a model Mj with Dj and S;
  }
}
```

The boosting and AdaBoost algorithms

As opposed to an ensemble algorithm, the bagging algorithm is the weighted voting and weighted sample training tuple dataset for each base classifier. The base classifiers are learned iteratively. Once a classifier is learned, the relative weights are updated with a certain algorithm for the next learning of the base classifiers. The successive model learning will emphasize the tuples misclassified by the former classifier. As a direct result, the accuracy of a certain classifier will play an important role in the voting of the final label once an unknown test tuple is provided for the combined classifier.

Adaptive Boosting or AdaBoost is one of the boosting algorithms. If it contains *K* base classifiers, then the AdaBoost will be performed as *K* passes. Given the tuple dataset D_i and its corresponding classifier M_i, the error rate $error(M_i)$ of classifier M_i is defined as follows:

$$error(M_i) = \sum_{j=1}^{d} w_j * error\left(X_j\right)$$

The new classifier M_i will be discarded once the error (M_i) is bigger than 0.5. The training tuples set will be resampled for this classifier and perform the training of this classifier from scratch.

For tuple $\forall X_j \in D_i$, the error function is as follows:

$$error\left(X_j\right) = 1\ for\ mis-classified, 0\ for\ others$$

All the weights of the tuples in the training tuples dataset D_i are initialized with $\frac{1}{d}, d = |D_i|$. When the classifier M_i is learned from the training tuples set D_i, the weight for the tuple, which is correctly classified, is multiplied by $\frac{error\left(M_i\right)}{1-error\left(M_i\right)}$. After the updates, the weights for all tuples are normalized, which means the weight of the classified tuple increases and the weight of the others decreases.

The weight of the vote of classifier M_i is as follows:

$$\log \frac{1-error\left(M_i\right)}{error\left(M_i\right)}$$

We defined $v_j\left(x\right)$ as the weight of the vote for class c_j upon the K classifiers, α_i representing the weight of the *i*th classifier:

$$v_j\left(X\right) = \sum_{i=1}^{K} \alpha_i * I\left(M_t\left(x\right) = c_j\right), I\left(M_t\left(x\right) = c_j\right) = 1\ when\ M_t\left(x\right) = c_j, 0\ for\ others$$

The AdaBoost combined classifier, M_*, counts the votes with their respective weights multiplied and assigns the class with the most votes to x. Each vote has the same weight in the equation:

$$M_*(X) = \underbrace{argmax}_{c_j} \{v_j(X) | j \in [1, K]\}$$

The input parameters for the AdaBoost algorithm are as follows:

- D, which denotes a set of training tuples
- k, which is the number of rounds
- A classification learning algorithm

The output of the algorithm is a composite model. The pseudocode of AdaBoost is listed here:

$$
\begin{array}{ll}
1: & Adaboost(D, K, S) \ \{ \\
2: & \quad set\ the\ initial\ value\ as\ \frac{1}{d}\ for\ the\ weight\ of\ each\ training\ tuple; \\
3: & \quad for(j \leftarrow 1; j \leq k; j++)\ \{ \\
4: & \qquad Create\ bootstrap\ sample, D_j, by\ sampling\ D\ with\ replacement \\
5: & \qquad Sample\ D\ with\ replacement\ according\ to\ the\ tuple\ weights\ to\ get\ D_j; \\
6: & \qquad Learning\ a\ model\ M_j\ with\ D_j; \\
7: & \qquad compute\ the\ error\ rate\ of\ M_j, i.e., error(D_j); \\
8: & \qquad if\ (error(M_j) > 0.5\ )\{ \\
9: & \qquad\quad go\ back\ to\ step\ 3\ and\ try\ again; \\
10: & \qquad \} \\
11: & \qquad for(each\ tuple\ in\ D_j\ that\ was\ corretly\ classified)\{ \\
12: & \qquad\quad updates\ its\ weight\ value\ by\ multiplying\ with\ \frac{error(M_j)}{1 - error(M_j)}; \\
13: & \qquad \} \\
14: & \quad normalize\ the\ weight\ of\ each\ tuple; \\
15: & \quad \} \\
16: & \}
\end{array}
$$

The Random forests algorithm

Random forests algorithm is an ensemble method to combine a group of decision trees, which is generated by a strategy of applying a random selection of attributes at each node that will be split. Given an unknown tuple, each classifier votes, and the most popular one decides the final result. The pseudocode for the ForestRI algorithm to generate a forest is as follows:

1: $GenerateForestRI(T, CPT)$ {

2: $T \leftarrow (X_1, X_2, \ldots, X_d)$; `//T denotes a total order of the variables`

3: $for\ (j \leftarrow 1;\ j \leq d; j++)$ {

4: $Let\ X_{T(j)}\ denote\ the\ j-th\ highesst\ order\ variable\ in\ T$

5: $Let\ \pi(X_{T(j)}) \leftarrow \{X_{T(1)}, X_{T(2)}, \ldots, X_{T(j-1)}\}$; `//denotes the set of variables preceding` $X_{T(j)}$

6: $Remove\ the\ variables\ from\ \pi(X_{T(j)})\ that\ do\ not\ affect\ X_j$ `(Using prior knowledge)`

7: $Create\ an\ arc\ between\ X_{T(j)}\ and\ the\ remaining\ variables\ in\ \pi(X_{T(j)})$.

8: }

9: }

`T` denotes a total order of the variables in line 2. In line 5, $\pi\left(X_{T(j)}\right)$ denotes the set of variables preceding $X_{T(j)}$. Prior knowledge is required for line 6.

Instead of random selection of attributes on splitting the node, for another algorithm, ForestRC, the random linear combination strategy of the existing attributes is used to split the task. New attributes are built by the random linear combination of the original attributes set. With a couple of new attributes added, the best split is searched over the updated attributes set including the new and original attributes.

The R implementation

Here we provide three R implementations, bagging, AdaBoost, and Random forests. Please look up the R codes file `ch_04_bagging.R`, `ch_04_adaboost.R`, `ch_04_forestrc.R`, and `ch_04_forestri.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following commands:

```
> source("ch_04_bagging.R")
> source("ch_04_adaboost.R")
```

```
> source("ch_04_forestrc.R")
> source("ch_04_forestri.R")
```

Parallel version with MapReduce

The following algorithm is the parallelized AdaBoost algorithm, which depends on a couple of workers to construct boosting classifiers. The dataset for the *p*th worker is defined using the following formula, where, n^p denoting its size is $p \in \{1,...,M\}$:

$$D^p_{n^p} = \{(\mathrm{x}^p_1, \mathrm{y}^p_1), (\mathrm{x}^p_2, \mathrm{y}^p_2), ..., (\mathrm{x}^p_{n^p}, \mathrm{y}^p_{n^p})\}$$

The classifier H^p is defined in the following format, with $\alpha^{p(\mathrm{t})}$ as the weight:

$$\{(h^{p(1)}, \alpha^{p(1)}), (h^{p(2)}, \alpha^{p(2)}), ..., (h^{p(T)}, \alpha^{p(T)})\}$$

The output is the final classifier. The input is the training dataset of M workers $(D_{n^1}{}^1, ..., D_{n^M}{}^M)$.

ADABOOST.PL($D^1_{n^1}, \ldots, D^M_{n^M}, T$)
1: **for** $p \leftarrow 1$ to M **do**
2: $\quad H^p \leftarrow$ ADABOOST($D^P_{N^P}, T$)
3: $\quad H^{p^*} \leftarrow$ the weak classifiers in H^p sorted w.r.t. $\alpha^{p(t)}$
4: **end for**
5: **for** $t \leftarrow 1$ to T **do**
6: $\quad h^{(t)} \leftarrow$ MERGE($h^{1^*(t)}, \ldots, h^{M^*(t)}$)
7: $\quad \alpha^t \leftarrow \frac{1}{M} \sum_{p=1}^{M} \alpha^{p^*(t)}$
8: **end for**
9: return $H = \sum_{t=1}^{T} \alpha^t h^{(t)}$

Biological traits and the Bayesian belief network

The Bayesian belief network, once trained, can be used for classification. Based on the Bayes' theorem, which is defined in the *The Bayes classification* section of *Chapter 3*, *Classification*, it is defined with two parts, one directed acyclic graph and **conditional probability tables** (**CPT**) for each variable; this is in turn represented by one node in the graph and models the uncertainty by graphically representing the conditional dependencies between distinct components. The arcs in the image give a representation of causal knowledge. The interaction among the diverse sources of uncertainty is also graphically illustrated.

The uncertainty comes from various sources:

- The way to associate the knowledge by the expert
- The domain intrinsic uncertainty
- The requirement of the knowledge to be translated
- The accuracy and availability of knowledge

Here is an example of the Bayesian belief network with four Boolean variables and the corresponding arcs. *Whether the grass is wet* is influenced by the work results of *sprinkler* and *whether it has just rained*, and so on. Each arc has a certain probability.

Let us have a look at the CPT representation of $P(WetGrass = T | Sprinkler = F, Rain = T) = 0.9$:

In the network, each variable is conditionally independent of its non-descendants. Here is the definition of the joint probability distribution:

$$P\left(x_1, x_2, \ldots, x_n\right) = \sum_{i=1}^{n} P(x_i \mid Parents\left(Y_i\right))$$

The Bayesian belief network (BBN) algorithm

Before the application of the BBN algorithm to classification, we need to train it first. In the process of training the network, the expert knowledge, that is, the prior knowledge, can be used in the training process to help the design of the network. For the variables that participated in direct dependency, experts must specify their conditional probability. There are many algorithms to learn the network from the training dataset; we will introduce an adaptive probabilistic networks algorithm.

The input parameters for the BBN algorithm are as follows:

- *T*, denotes a total order of the variables
- *CPT*

The output of the algorithm is the topology structure of BBN, which is as follows:

$$1: GenerateBBN\left(T, CPT\right)\{$$
$$2: T \leftarrow \left(X_1, X_2, \ldots, X_d\right);$$
$$3: for\ (j \leftarrow 1; j \le d; j++)\{$$
$$4: \quad Let\ X_{T(j)}\ denote\ the\ j-th\ highesst\ order\ variable\ in\ T$$
$$5: \quad Let\ \pi\left(X_{T(j)}\right) \leftarrow \left\{X_{T(1)}, X_{T(2)}, \ldots, X_{T(j-1)}\right\};$$
$$6: \quad Remove\ the\ variables\ from\ \pi\left(X_{T(j)}\right) that\ do\ not\ affect\ X_j$$
$$7: \quad Create\ an\ arc\ between\ X_{T(j)}\ and\ the\ remaining\ variables\ in\ \pi\left(X_{T(j)}\right).$$
$$8: \quad \}$$
$$9: \}$$

T denotes a total order of the variables in line 2. In line 5, $\pi\left(X_{T(j)}\right)$ denotes the set of variables preceding $X_{T(j)}$.

The R implementation

Please look up the R codes file `ch_04_bnn.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_04_bnn.R")
```

Biological traits

A biological trait is one of the important applications of the BBN algorithm.

Protein classification and the k-Nearest Neighbors algorithm

The **k-Nearest Neighbors** (**kNN**) algorithm is one of the lazy learners that postpones the learning until the test tuple or test instance is provided.

A single training tuple is represented by a point in an *n*-dimensional space. In other words, *n* attributes' combinations are used to represent the specific training tuple. There is no specific training before the arrival of the test tuple that needs to be classified. Some preprocessing steps are needed, such as normalization for some attributes with large values compared to other attributes' values. Data normalization approaches in the data transformation can be applied here for preprocessing.

When a test tuple is given, the k-nearest training tuples are found from the training tuples space by a specific measure to calculate the distance between test tuple and the training tuple. The k-nearest training tuples are also known as the kNN. One popular solution is the Euclidean distance in real space, illustrated in the following equation. This method is only applicable to numeric attributes:

$$distance\left(X_p, X_q\right) = \sqrt{\sum_{i=1}^{n}\left(x_{qi} - x_{qi}\right)^2}, X_p, X_q \in R^n$$

For nominal attributes, one solution is that the difference between two attribute values is defined as *1*, or as *0*. We already know that many approaches deal with missing values in the attributes. With a predefined threshold, the value of *k* is selected with the number of tuples with the lowest error-rate among all the training tuples.

The class label of the test tuple is defined by the voting of the most common class in the kNN.

The kNN algorithm

The input parameters for kNN algorithm are as follows:

- D, the set of training objects
- z, the test object, which is a vector of attribute values
- L, the set of classes used to label the objects

The output of the algorithm is the class of z, represented as $c_z \in L$.

The pseudocode snippet for kNN is illustrated here:

1: $GenerateKNN(D, z, L)$ {

2: $\quad for\ (each\ object\ y \in D)$ {

3: $\quad\quad Compute\ d(z, y), the\ distance\ between\ z\ and\ y;$

4: }

5: $\quad Select\ N \subseteq D, the\ set\ (neighborhood)\ of\ k\ closest\ training\ object\ for\ z;$

6: $\quad \boldsymbol{c_z} = \text{argmax}_{v \in L} \sum_{y \in L} I\left(v = class\left(c_y\right)\right), I\left(v = class\left(c_y\right)\right),$ I() is an indicator function that returns the value 1 if its argument is true and 1 otherwise.

7: }

The I function in line 6 denotes an indicator function that returns the value `1` if its argument is true and `0` otherwise.

The R implementation

Please look up the R codes file `ch_04_knn.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_04_knn.R")
```

Document retrieval and Support Vector Machine

Support Vector Machine (**SVM**) is a classification algorithm applicable to both linear and nonlinear data classification. It is based on an assumption: if two classes of data cannot be divided by a hyper-plane, then after mapping the source dataset to sufficient higher dimension spaces, the optimal separating hyper-plane must exist.

Here are two concepts that need to be clearly defined:

- **Linearly separable**: This means that a dataset can be divided into the target classes with a linear equation with the input of a training tuple.
- **Nonlinearly separable**: This means that none of the linear equations exist in the space with the same dimension as that of the training tuple.

The linear hyper-plane can be represented as the linear discriminant equation, given the weight vector `w` and the training tuple `x`,

$$\boldsymbol{W} = \left(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\right)$$

$$\boldsymbol{X} = \left(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\right)$$

$$g\left(\boldsymbol{X}\right) = \boldsymbol{W} * \boldsymbol{X}^T + b = 0$$

With the preceding equation, we have the following image to illustrate a hyper-plane:

The target of SVM is to find the optimal hyper-plane, by which the margin between data points belonging to different classes are maximized.

There are two hyper-planes with equal distance and that are parallel to the $g(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}^T + b = 0$ hyper-plane. They are boundary hyper-planes and all support vectors are on them. This is illustrated in the following diagram:

In the following diagram, the case of a nonlinearly separable case is illustrated:

In the following diagram, after mapping the vector from a low-dimensional space to high-dimensional space, the nonlinearly separable case will be transformed into to a linearly separable case:

The SVM algorithm

The input parameters for a dual SVM algorithm are as follows:

- *D*, the set of training objects
- *K*
- *C*
- ε

The output of the algorithm is the SVM algorithm. The pseudocode snippet for this algorithm is illustrated here:

$$
\begin{aligned}
&1: DualSVM(D, K, C, \in)\ \{\\
&2: \quad for(each\ x_j \in \mathrm{D})\ \{\\
&3: \qquad x_j \leftarrow \begin{pmatrix} x_j \\ 1 \end{pmatrix}\\
&4: \quad \}\\
&5: \quad if\,(loss = quadratic)\{\\
&6: \quad K \leftarrow \{K(x_i, x_j) + \frac{1}{2*C}\delta_{ij}\}, i, j = 1, \ldots, n\\
&7: \quad \}else\ if\,(loss = longe)\{\\
&8: \quad K \leftarrow \{K(x_i, x_j)\}, i, j = 1, .., n\\
&9: \quad \}\\
&10: \ for(each\ j \in [1, n])\ \{\\
&11: \qquad \eta_j \leftarrow \frac{1}{K\left(x_{j,} x_j\right)}\\
&12: \ \}
\end{aligned}
$$

$$
\begin{aligned}
&13: t \leftarrow 0 \\
&14: \alpha_0 \leftarrow (0,...,0)^T \\
&15: do\{ \\
&16: \quad \alpha \leftarrow \alpha_t \\
&17: \quad for(k=1; k \le 1; k++)\{ \\
&18: \quad\quad \alpha_k \leftarrow \alpha_k + \eta_k * \left(1 - y_k * \left(\sum_{i=1}^{n} \alpha_i * y_i * K(x_i, x_k)\right)\right) \\
&19: \quad\quad if(\alpha_k < 0)\alpha_k \leftarrow 0 \\
&20: \quad\quad if(\alpha_k < C)\alpha_k \leftarrow C \\
&21: \quad \} \\
&22: \quad \alpha_{t+1} \leftarrow \alpha \\
&23: \quad t \leftarrow t+1 \\
&24: \}until(\|\alpha_t - \alpha_{t-1}\| \le \varepsilon) \\
&25: \}
\end{aligned}
$$

Here is the pseudocode for another version of the SVM algorithm, which is the primal kernel SVM algorithm. The input parameters for the primal kernel SVM algorithm are as follows:

- D, the set of training objects
- K
- C
- ε

The output of the algorithm is the SVM model. The pseudocode snippet for this algorithm is illustrated here:

$$
\begin{aligned}
&1: PrimalKernalSVM(D,K,C,\varepsilon)\{\\
&2:\ \ for(each\ x_j \in D)\{\\
&3:\qquad x_j \leftarrow \left(\frac{x_j}{1}\right)\\
&4:\ \}\\
&5:\ K \leftarrow \{K(x_i,x_i)\}, i, j = 1,..,n\\
&6: t \leftarrow 0\\
&7: \beta_0 \leftarrow (0,...,0)^T, \beta_0, \beta_t \in R^n\\
&8: do\{\\
&9:\quad \mathrm{v} \leftarrow \sum_{yi(K_i^T * \beta_t)<1} y_i * K_i\\
&10:\quad s \leftarrow \sum_{yi(K_i^T * \beta_t)<1} y_i * K_i\\
&11:\quad \nabla \leftarrow (K + 2CS)\beta_t - 2C * v\\
&12:\quad H \leftarrow K + 2CS\\
&13:\quad 22: \beta_{t+1} \leftarrow \beta_t - \eta_t * H^{-1} * \nabla\\
&14:\quad t \leftarrow t+1\\
&15:\ \ \}until(\|\beta_t - \beta_{t-1}\| \le \varepsilon)\\
&16:\}
\end{aligned}
$$

The R implementation

Please look up the R codes file `ch_04_svm.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_04_svm.R")
```

Parallel version with MapReduce

Along with the online computation requests, such as mobile cloud applications for classification, a high-performance, robust, and high-accuracy classification platform or algorithm is widely required. The parallelized SVM distributes the computing of optimization by the MapReduce technique to achieve working on a large scale of datasets and high performance at the same time. There are many implementations of various versions of parallelized SVM. One of them is shown here.

The `t` denotes for iteration number, `l` for MapReduce function size, h^t for the best hypothesis at iteration `t`, D_l for sub-dataset for node `l`, SV_l for support vectors at node `l` and SV_{Global} for global support vector:

1. As initialization, the global support vector set as $t = 0, SV^t = \emptyset$
2. t = t + 1
3. For any computer in $l, l = 1, \ldots, L$ reads global SVs and merge them with subset of training data
4. Train SVM algorithm with merged new dataset
5. Find out support vectors
6. After all computers in cloud system complete their training phase, merge all calculated SVs and save the result to the global SVs
7. If $h^t = h^{t-1}$ stop, otherwise go to step 2

Next, the mapper algorithm is designed and the for `loop` is immediately behind the while `loop`, which is looped for each subset:

Map Function

$SV_{Global} = \emptyset$
while $h^t \neq h^{t-1}$
  for $l \in L$ **do**
    $D_l^t \leftarrow D_l^t \cup SV_{Global}^t$
  end for
end while

Finally, the reducer algorithm is designed. On line 3, the code inside the for `loop` immediately follows the while `loop`. This is for training by merging the datasets to obtain support vectors and binary-class hypothesis:

Reduce Function

while $h^t \neq h^{t-1}$ **do**
  for $l \in \boldsymbol{L}$
    $SV_l, h^t \leftarrow binarySvm(D_l)$
  end for
  for $l \in \boldsymbol{L}$
    $SV_{Global} \leftarrow SV_{Global} \cup SV_l$
  end for
end while

Document retrieval

One of the important applications of SVM is document retrieval, which has a static information source; the task is to fetch a ranking of documents as a response to a user request. The vector model is a widely implemented document-retrieval or information-retrieval model.

Classification using frequent patterns

There are two types of classification using frequent patterns:

- Associative classification model as well as association rules, which are generated from frequent patterns and used for classifications
- Discriminative frequent pattern-based classification

The associative classification

The generic association classification algorithm is defined here. The input parameters for the kNN algorithm are as follows:

- *D*, which is a set of training objects
- *F*, which is the itemset
- *MIN_SUP*, which is the minimal support
- *MIN_CONF*, which is the minimal confidence

The output of the algorithm is a rule-based classifier and is shown as follows:

```
GenerateGenericAC(D, F, MIN_SUP, MIN_CONF){
  Perform association mining, satisfying the minimal support and confidence too,
  Build a rule-based classifier based on the transformation of result rules
}
```

Two popular algorithms are illustrated in the successive sections, one is **Classification Based on Association** (**CBA**), and the other is **Classification Based on Multiple Association Rules** (**CMAR**).

CBA

Here is the pseudocode for CBA:

```
GenerateCBA(D, F, MIN_SUP, MIN_CONF){
  Perform association mining
  Let R is the resulting rules set
  Let C be a empty rule – based classifier
  for(each r in R){
     Intergrating r into the C with an improved rule-based classifier
  }
}
```

Discriminative frequent pattern-based classification

Here is the pseudocode for discriminative frequent pattern-based classification:

```
GenerateDFPC(D, F, MIN_SUP, MIN_CONF){
  Perform association mining, satisfying the minimal support and confidence
  Let R is the resulting rules set, R is ordered in decreasing precedence based,
on the MIN_SUP and MIN_CONF, for the rules with same antecedent,
the one with highest confidence is kept, others discarded
  Let C be a empty rule-based classifier
  for(each r in R){
     Intergrating r into the C with an improved rule-based classifier
  }
}
```

The R implementation

Please look up the R codes files `ch_04_associative_classification.R`, `ch_04_cba.R`, and `ch_04_frequent_pattern_based_classification.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following commands:

```
> source("ch_04_associative_classification.R")
> source("ch_04_cba.R")
> source("ch_04_frequent_pattern_based_classification.R")
```

Text classification using sentential frequent itemsets

One application of CBA is text classification. The key here is to build a matrix with a document or text term and labels. With the matrix built, any classification algorithm can be applied to it. One example of document matrix is illustrated here. The term might include a character, word, phrase, and concept and so on.

Y	have	inc	its	last	mln	new	...
1	0.00	1.00	1.00	0.00	0.00	0.00	
1	0.00	0.00	0.00	0.00	1.00	1.00	
0	1.00	0.00	9.00	1.00	2.00	2.00	
0	0.00	0.00	2.00	0.00	1.00	1.00	
1	0.00	0.00	1.00	0.00	0.00	0.00	

Classification using the backpropagation algorithm

The **backpropagation** (**BP**) algorithm learns the classification model by training a multilayer feed-forward neural network. The generic architecture of the neural network for BP is shown in the following diagrams, with one input layer, some hidden layers, and one output layer. Each layer contains some units or perceptron. Each unit might be linked to others by weighted connections. The values of the weights are initialized before the training. The number of units in each layer, number of hidden layers, and the connections will be empirically defined at the very start.

The training tuples are assigned to the input layer; each unit in the input layer calculates the result with certain functions and the input attributes from the training tuple, and then the output is served as the input parameter for the hidden layer; the calculation here happened layer by layer. As a consequence, the output of the network contains all the output of each unit in the output layer. The training is performed iteratively by updating the weights of the connections with the feedback of errors, that is, back propagation of the errors.

The prototype for the unit in the hidden/output layer is shown here, the one in the input layer with only one input and other minor differences compared to this. W_{1j} denotes the weight related to that link or connection. Each unit is bound with a bias, which is θ. The threshold or activation function bound with each unit is f.

For a certain unit or perceptron in the hidden or output layer, the net input is a combination of the linear combination of each of its input from the previous unit, that is, the output of it, which is O_p. Let k denote the number of input connections of the unit q:

$$I_q = \sum_{p=1}^{k} w_{pq} * O_p + \theta_q$$

The output of this unit q is O_q.

$$O_q = \frac{1}{1+e^{-I_q}}$$

If the unit q is in the output layer and T_q denotes the expected or known output value in the training tuple, then its error Err_q can be calculated as follows:

$$Err_q = O * (1 - O_q) * (T_q - Q_q)$$

If the unit q is in the hidden layer, then let w_{qp} denote the weight of the connection from unit q to one unit with the error Err_p in the next layer or the known output value in the training tuple. We can then calculate its error as Err_q. Let M denote the number of output connections of the unit q:

$$Err_q = O * (1 - O_q) * \sum_{p=1}^{M} Err_p * w_{qp}$$

After this preprocessing or preparation, the weights and biases can be updated accordingly with the backpropagation strategy. η denotes the learning rate; it is an empirical parameter, which belongs to `(0,1)`:

$$\Delta w_{pq} = (\eta) * Err_q * O_p$$

$$w_{pq} = w_{pq} + \Delta w_{pq}$$

$$\Delta \theta_q = (\eta) + Err_q$$

$$\theta_q = \theta_q + \Delta \theta_q$$

The weights and biases are updated per tuple of the training tuple dataset.

The BP algorithm

The input parameters for the BP algorithm, the topology of the neural network, the number of hidden layers, and the connections, are defined before the start of the training:

- D, which is the training tuples set
- W, which is the initial values for all the weights.
- θ, which is the bias for each units
- I, which is the learning rate

The output of the algorithm is the topology structure of BP, which consists of:

- *BPNN*, which is the trained BP neural network
- *W*, which is a set of weights of the connections in the neural network

Here is the pseudocode for the training of the backpropagation network:

```
GenerateBPNN(D,W,θ,I){
  Intialized all the weights and biases in network;
  While(termination condition is not true){
    for(each tuple X in D){
    for(each input layer unit j){
      O_j ← I_j
    }
    for(each children or output layer unit j){
      I_j ← Σ_i w_ij O_i + θ_j
      O_j ← 1 / (1 − e^(−I_j))
    }
    for(each output layer unit j){
      Err_j ← O_j * (1 − O_j) * (T_j − O_j)
    }
    for(each output layer unit j){
      Err_j ← O_j * (1 − O_j) * (T_j − O_j)
    }
    for(each unit in the hidden layers, from last to first hidden layer){
      Err_j ← O_j * (1 − O_j) * Σ_k Err_k * w_jk
    }
    for(each weight w_ij in network){
      Δw_ij ← (l) * Err_j * O_i
      w_ij ← w_ij + Δw_ij
    }
    for(each bias θ_j in network){
```

$$26: \quad \Delta\theta_j \leftarrow (l) * Err_j$$

$$27: \quad \theta_j \leftarrow \theta_j + \Delta\theta_j$$

$$28: \quad \}$$

$$29: \quad \}$$

$$30: \}$$

$$31: \}$$

The R implementation

Please look up the R codes file `ch_04_bp.R` from the bundle of R codes for the BP algorithm. The codes can be tested with the following command:

```
> source("ch_04_bp.R")
```

Parallel version with MapReduce

Massive data makes the training process of BP dramatically slow. The parallelized version of BP is proven to accelerate the speed in an amazing way. There are many versions of parallelized BPNN algorithms implemented on the MapReduce architecture. Here is one implementation, the **MapReduce-based Backpropagation Neural Network** (**MBNN**) algorithm.

Given the training dataset, each mapper is fed with one single training item. During the mapper tasks stage, new values for weights are calculated. Then, within the reducer tasks stage, new values for one weight are collected, and this results in an average value of these values to output for the weights. After the new values are provided, all the weights are updated, batch-wise. These steps are executed iteratively until the termination condition is matched.

The backpropagation main algorithm is listed as follows:

	Backpropagation Reducer
1	Input key/value pair < *key* = *w*, *value* = Δw >
2	*sum* ←0, *count* ← 0
3	*sum* ← *sum* + *value*
4	*count* ← *count* + 1
5	**If** more pairs <*key*, *value*> are collected, go to step 1
	Else output key/value pair <*key*, *sum/count*>

Four steps compose the backpropagation mapper algorithm. They are listed here:

	Backpropagation Mapper
1	Input key/value pair < *key*, *value* = inputItem >
2	Compute the update-value for all the weights using *value* as the input of the network
3	For every weight *w*, get its weight Δw
4	Emit intermediate key/value pair <*w*, Δw>

Five steps compose the backpropagation reducer algorithm. They are listed here:

	Backpropagation Reducer
1	Input key/value pair < *key* = *w*, *value* = Δw >
2	*sum* ←0, *count* ← 0
3	*sum* ← *sum* + *value*
4	*count* ← *count* + 1
5	**If** more pairs <*key*, *value*> are collected, go to step 1
	Else output key/value pair <*key*, *sum/count*>

Time for action

Here are some practice questions for you to check your understanding about the concepts covered:

- Find one application of the classification using frequent patterns
- What is the BBN algorithm?
- What is the kNN algorithm?
- What is the backpropagation algorithm?
- What is SVM?

Summary

In this chapter, we looked at the following topics:

- Ensemble methods
- Bagging
- AdaBoost
- Random forests
- BBN, which provides a topology model of causal relationship; it is an eager learner.

> An eager learner is one that behaves in an opposite way to the lazy learner. The lazy learner postpones the learning until the test tuple or test instance is provided.

- The kNN algorithm, which is a lazy learner
- SVM, by which the original data is transformed to a higher dimension, and is separated with a hyper-plane; it is an eager learner
- Classification using frequent patterns; this is an eager learner
- Classification using the BP algorithm, which is a neural network trained with a descent gradient

In the next chapter, we will learn the clustering algorithm, which is also a kind of unsupervised classification with no predefined labels.

5
Cluster Analysis

Clustering is defined as an unsupervised classification of a dataset. The objective of the clustering algorithm is to divide the given dataset (a set of points or objects) into groups of data instances or objects (or points) with distance or probabilistic measures. Members in the same groups are closer by distance or similarity or by other measures. In other words, maximize the similarity of the intracluster (internal homogeneity) and minimize the similarity of the intercluster (external separation).

In this chapter, you will learn how to implement the top algorithms for clusters with R; these algorithms are listed here:

- Search engine and the k-means algorithm
- Automatic abstracting of document texts and the k-medoids algorithm
- The CLARA algorithm
- CLARANS
- Unsupervised image categorization and affinity propagation clustering
- Web page clustering and spectral clustering
- News categorization and hierarchical clustering

A clustering algorithm is used for the preparation of further analysis, while the other objective is to understand the nature of the dataset. The most common clustering process is illustrated in the following diagram:

The key steps in this process include the following:

- **Feature selection**: This step chooses distinguished features from the original dataset
- **Clustering algorithm design**: This step designs an appropriate algorithm based on the currently available clustering algorithms, or it just builds one from scratch
- **Cluster validation**: This step evaluates the clusters and provides a degree of confidence about the result
- **Result interpretation**: This step gives an intrinsic idea of the input dataset

There are many categorization methods to categorize clustering algorithms; the major categories are partition-based methods, density-based methods, hierarchical methods, spectral methods, grid-based methods, and so on.

Every clustering algorithm has its limitations and best practices for certain conditions or datasets. Once an algorithm is chosen, the parameters and distance measures (only for some algorithms) related to that algorithm need careful consideration as well. Then, we will list the most popular clustering algorithms and their corresponding parallel versions (if available).

Here is the short list of clustering algorithms and their complexities:

Cluster algorithm	**Capability of tracking high dimensional data**
k-means	No
Fuzzy c-means	No
Hierarchical clustering	No
CLARA	No
CLARANS	No
BIRCH	No
DBSCAN	No
CURE	Yes
WaveCluster	No
DENCLUE	Yes
FC	Yes
CLIQUE	Yes
OptiGrid	Yes
ORCLUS	Yes

The difficulties or targets in another aspect of clustering algorithms are the arbitrary shape of clusters, large volume of datasets, high-dimensional datasets, insensitivity to noise or outlier, low reliance to user-defined parameters, capability of dealing with new data without learning after the initial learning or clustering, sensitivity to the intrinsic, or nature of the number of clusters, nice data visualization, and application to both numeric or nominal data types.

Search engines and the k-means algorithm

The general process of partition-based clustering is iterative. The first step defines or chooses a predefined number of representatives of the cluster and updates the representative after each iteration if the measure for the clustering quality has improved. The following diagram shows the typical process, that is, the partition of the given dataset into disjoint clusters:

(a) Initial clustering **(b)** Iterate **(c)** Final clustering

The characteristics of partition-based clustering methods are as follows:

- The resulting clusters are exclusive in most of the circumstances
- The shape of the clusters are spherical, because of most of the measures adopted are distance-based measures
- The representative of each cluster is usually the mean or medoid of the corresponding group (cluster) of points
- A partition represents a cluster
- These clusters are applicable for small-to-medium datasets
- The algorithm will converge under certain convergence object functions, and the result clusters are often local optimum

The k-means clustering algorithm is basically a greedy approach that defines the centroid of each cluster with its mean value. It is efficient for dealing with a large dataset. The k-means algorithm is a kind of exclusive clustering algorithm in which data is clustered in an exclusive way, and one object only belongs to, at the most, one group or cluster. It is also a type of partitional clustering algorithm in which clusters are created in one step, in contrast to a couple of steps.

The value of k is often determined by the domain knowledge, the dataset, and so on. At the start, k objects in the initial dataset D (the size of D is *n*, where $k \le n$) are randomly selected as the initial centers for the initial k clusters. In the iteration of the k-means algorithm, each object is assigned to the most similar or closest (various measures are used for distance or similarity) cluster (mean). Once the iteration ends, the mean for each cluster is updated or relocated. The k-means algorithm performs as much iteration as possible until there is no change that the clusters get from the previous clustering.

The quality of the specific cluster of the clustering algorithm is measured by various merits. One of them is formulated in the following equation. This is within the cluster variation measure, where c_i stands for the centroid of the cluster C_i. Here, *k* is the number of clusters, whereas $dist(p, c_i)$ represents the Euclidean distance of the two points. The minimum value of E is the needed value and depicts the best quality clusters. This evaluation or assessing objective function serves as the ideal one, though not practical for concrete problems. The k-means algorithm provides easy methods for an approximate-to-ideal value. The k-means algorithm is also known as the **squared error-based clustering algorithm**.

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist^2 \left(p, c_i \right)$$

$$\frac{1}{n} \sum_{j=1}^{n} \left[\min_j d^2 \left(p, c_i \right) \right]$$

In practice, the k-means algorithm can run several times with a different initial set of centroids to find a relatively better result.

The varieties of the k-means clustering algorithm are designed with different solutions for the selection of initial k centroids or means. The similarity or dissimilarity is measured, and the way to calculate the means is established.

The shortages of the k-means clustering method are listed as follows:

- The means of clusters must be defined with a function
- It is only applicable to numeric data type
- The value of k needs to be predefined by users, and it is difficult

The guidelines or thumb rules for the k-means clustering method are as follows:

- Remember this method is sensitive to noise and outlier
- This method is only applicable to clusters with closer sizes
- This method finds it difficult to find nonconvex shapes

The k-means clustering algorithm

The input parameters for a dual SVM algorithm is as follows:

- D, which is the set of training objects
- K
- $\mathcal{E}$

These parameters are used as depicted in the summarized pseudocodes of the k-means clustering algorithm given as follows:

K-MEANS $(\mathbf{D}, k, \epsilon)$:

$t = 0$

Randomly initialize k centroids: $\boldsymbol{\mu}_1^t, \boldsymbol{\mu}_2^t, \ldots, \boldsymbol{\mu}_k^t \in \mathbb{R}^d$

repeat

  $t \leftarrow t + 1$

  `// Cluster Assignment Step`

  foreach $\mathbf{x}_j \in \mathbf{D}$ **do**

    $j^* \leftarrow \arg\min_i \left\{ \left\| \mathbf{x}_j - \boldsymbol{\mu}_i^t \right\|^2 \right\}$ `// Assign` $\mathbf{x}_j$ `to closest centroid`

    $C_{j^*} \leftarrow C_{j^*} \cup \{\mathbf{x}_j\}$

  `// Centroid Update Step`

  foreach $i = 1$ *to* k **do**

    $\boldsymbol{\mu}_i^t \leftarrow \frac{1}{|C_i|} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j$

until $\sum_{i=1}^{k} \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\| \leq \epsilon$

The kernel k-means algorithm

The summarized pseudocode for the kernel k-means clustering algorithm is as follows:

```
KERNEL-KMEANS(K, k, ε):
t ← 0
C^t ← {C_1^t, ··· , C_k^t}// Randomly partition points into k clusters
repeat
    t ← t + 1
    foreach C_i ∈ C^{t-1} do // Compute squared norm of cluster means
        sqnorm_i ← (1/n_i^2) Σ_{x_a ∈ C_i} Σ_{x_b ∈ C_i} K(x_a, x_b)
    foreach x_j ∈ D do // Average kernel value for x_j and C_i
        foreach C_i ∈ C^{t-1} do
            avg_ji ← (1/n_i) Σ_{x_a ∈ C_i} K(x_a, x_j)
    // Find closest cluster for each point
    foreach x_j ∈ D do
        foreach C_i ∈ C^{t-1} do
            d(x_j, C_i) ← sqnorm_i − 2 · avg_ji
        j* ← arg min_{C_i} {d(x_j, C_i)}
        C_{j*}^t ← C_{j*}^t ∪ {x_j} // Cluster reassignment
    C^t ← {C_1^t, ..., C_k^t}
until 1 − (1/n) Σ_{i=1}^{k} |C_i^t ∩ C_i^{t-1}| ≤ ε
```

The k-modes algorithm

This algorithm is a variant of the k-means algorithm; it can deal with categorical data types and large datasets. It can be integrated with the k-means method to deal with the clustering of the dataset that contains all data types. The algorithm is mentioned as follows:

The k-Modes algorithm with the proposed dissimilarity measure.

```
Initialize the variable oldmodes as a k × |P|-ary empty array;
Randomly choose k distinct objects x1, x2,...,xk from U
and assign [x1,x2,...,xk] to the k × |P|-ary array variable newmodes;
for l = 1 to k
  for j = 1 to |P|
    calculate the similarity Sim_aj(x_l, x_l) according to Definition 2;
  end;
end;
while oldmodes< >newmodes do
   oldmodes = newmodes;
   for i = 1 to |U|
     for l = 1 to k
       calculate the dissimilarity between the ith object and
       the lth mode according to Definition 5, and classify the ith
       object into the cluster whose mode is closest to it;
     end;
   end;
   for l = 1 to k
     find the mode z_l of each cluster and assign to newmodes;
     for j = 1 to |P|
       calculate the similarity Sim_aj(z_l, z_l) according to Definition 2;
       calculate m_aj of Definition 5;
     end;
   end;
   if oldmodes==newmodes
     break;
   end;
 end.
```

The R implementation

Take a look at the `ch_05_kmeans.R`, `ch_05_kernel_kmeans.R`, and `ch_05_kmedoids.R` R code files from the bundle of R code for the previously mentioned algorithms. The code can be tested using the following commands:

```
> source("ch_05_kmeans.R")
> source("ch_05_kernel_kmeans.R")
> source("ch_05_kmedoids.R")
```

Parallel version with MapReduce

The parallelized k-means is listed as follows:

map $(key,\ value)$

```
1. Construct the sample instance from value;
2. minDis = Double.MAX_VALUE;
3. index = -1;
4. For i=0 to centers.length do
        dis= ComputeDist(instance, centers[i]);
        If dis < minDis {
            minDis = dis;
            index = i;
        }
5. End For
6. Take index as key';
7. Construct value' as a string comprise of the values of different dimensions;
8. output < key', value' > pair;
9. End
```

combine $(key,\ V)$

```
1. Initialize one array to record the sum of value of each dimensions of the samples contained in
   the same cluster, i.e. the samples in the list V;
2. Initialize a counter num as 0 to record the sum of sample number in the same cluster;
3. while(V.hasNext()){
        Construct the sample instance from V.next();
        Add the values of different dimensions of instance to the array
        num++;
4. }
5. Take key as key';
6. Construct value' as a string comprised of the sum values of different dimensions and num;
7. output < key', value' > pair;
8. End
```

reduce $(key,\ V)$

```
1. Initialize one array record the sum of value of each dimensions of the samples contained in the
   same cluster, e.g. the samples in the list V;
2. Initialize a counter NUM as 0 to record the sum of sample number in the same cluster;
3. while(V.hasNext()){
        Construct the sample instance from V.next();
        Add the values of different dimensions of instance to the array
        NUM += num;
4. }
5. Divide the entries of the array by NUM to get the new center's coordinates;
6. Take key as key';
7. Construct value' as a string comprise of the center's coordinates;
8. output < key', value' > pair;
9. End
```

Search engine and web page clustering

Along with the nonstop accumulation of Internet documents, the difficulties in finding some useful information keeps increasing. To find information in these documents or web pages or from the Web, four search methodologies are provided to us:

- The unassisted keyword search
- The assisted keyword search
- The directory-based search
- The query-by-example search

Web page clustering is an important preprocessing step to web data mining, which is one solution among the many possible solutions. The document clustering occurs in the progress of IR and text clustering. Many web clustering criteria are provided, such as the semantic, the structure, the usage-based criteria, and so on. Domain knowledge plays an important role in web page clustering.

Term Frequency-Inverse Document Frequency (**TF-IDF**) is applied during the preprocessing of the document dataset. One modeling method to represent a data instance for document clustering is the **vector-space model**. Given a term space, each dimension with a specific term in the documents and any document in the original document dataset can be represented by a vector, as depicted in the following equation. This definition implies that mostly, frequently used terms play an important role in the document:

$$d_{tf} = \left(tf_1, tf_2, \ldots, tf_n\right)$$

Each value in a dimension denotes the frequency of the term that labels the dimension appearing in the document. For simplicity, the vector needs to be normalized as a unit length before processing it further, the stop words should be removed, and so on. As a potential and popular solution, the TF often weighs every term using the inverse document frequency among the document datasets. The inverse document frequency is denoted by the following formula in which *n* is the dataset size, and $\mathrm{df}\left(t_i\right)$ is the document's number that contains the term, t_i:

$$idf\left(t_i\right) = \log\left(\frac{n}{df(t_i)}\right)$$

Given the definition of inverse document frequency, there is another popular definition of the vector model to denote a document in the document collection for further processing using a clustering algorithm; $tf\left(d_i, t_i\right)$ is the frequency of term t_i in the document d_i:

$$d_j = (w_{j1}, \ldots, w_{jm})$$

$$w_{ji} = tf(d_j, t_i) \cdot idf(t_i)$$

The measures used in the document clustering are versatile and tremendous. Cosine method is one among them, and we will use the vector dot production in the equation while the corresponding centroid is defined as *c* in the successive equation:

$$cosine(d_1, d_2) = d_1 \cdot d_2 / \| d_1 \| \| d_2 \|$$

$$c = \frac{1}{|\mathrm{S}|} \sum_{d \in s} d$$

Reuters-21578 is one publicly-available document dataset for further research. TREC-5, 6, and 7 are also open source datasets.

With this measure definition, the k-means algorithm is used for web page clustering, as it displays its efficiency and always acts as a component for a practical web search engine.

Automatic abstraction of document texts and the k-medoids algorithm

The k-medoids algorithm is extended from the k-means algorithm to decrease the sensitivity to the outlier data points.

Given the dataset *D* and the predefined parameter *k*, the k-medoids algorithm or the PAM algorithm can be described as shown in the upcoming paragraphs.

As per a clustering related to a set of k medoids, the quality is measured by the average distance between the members in each cluster and the corresponding representative or medoids.

An arbitrary selection of *k* objects from the initial dataset of objects is the first step to find the k medoids. In each step, for a selected object O_i and a nonselected node O_h, if the quality of the cluster is improved as a result of swapping them, then a swap is performed.

The cluster quality should be the sum of all the differences in the distance between the members and medoids, before and after the swap. For each nonselected object O_h, there are four difference cases under consideration (they are marked out in the following diagram). Given a set of medoids, one of them is O_i, and the cluster related to it is C_i.

- **Case 1**: The first difference case is $O_j \in C_i$, given that $O_{j,2}$ is the representative of the second medoid that is closer or more similar to O_j:

$$d\left(O_j, O_h\right) \geq d\left(O_j, O_{j,2}\right)$$

After swapping, O_j will be relocated to $c_{j,2}$, and the cost of swapping will be defined as follows:

$$C_{jih} = d\left(O_j, O_{j,2}\right) - d\left(O_j, O_i\right)$$

- **Case 2**: The second difference case is $O_j \in C_i$, $d\left(O_j, O_h\right) < d\left(O_j, O_{j,2}\right)$. After swapping, O_j will be relocated to C_h, and the cost of swapping is defined as follows:

$$C_{jih} = d\left(O_j, O_h\right) - d\left(O_j, O_i\right)$$

- **Case 3**: Here, $O_j \in C_t, t \neq i, O_t \; represents \; C_t$ and $d\left(O_j, O_h\right) \geq d\left(O_j, O_t\right)$. After swapping, the cost of swapping is defined as follows:

$$C_{jih} = 0$$

- **Case 4**: Here, $O_j \in C_t, t \neq i, O_t \; represents \; C_t$ and $d\left(O_j, O_h\right) < d\left(O_j, O_t\right)$. After swapping, O_j will be relocated to C_h, and the cost of swapping is defined as follows:

$$C_{jih} = d\left(O_j, O_h\right) - d\left(O_j, O_t\right)$$

At the end of each swapping step, the total cost of swapping is defined as follows:

$$\sum_j C_{jih}$$

The PAM algorithm

The **PAM** (**Partitioning Around Medoids**) algorithm is a partitional clustering algorithm. The summarized pseudocode for the PAM algorithm is as follows:

Algorithm PAM

1. Select k representative objects arbitrarily.
2. Compute TC_{ih} for *all* pairs of objects O_i, O_h where O_i is currently selected, and O_h is not.
3. Select the pair O_i, O_h which corresponds to $min_{O_i,O_h}\ TC_{ih}$. If the minimum TC_{ih} is negative, replace O_i with O_h, and go back to Step (2).
4. Otherwise, for each non-selected object, find the most similar representative object. Halt. □

The R implementation

Take a look at the `ch_05_kmedoids.R` R code file from the bundle of R code for the previous algorithms. The code can be tested with the following command:

```
> source("ch_05_kmedoids.R")
```

Automatic abstraction and summarization of document text

Along with the increase in the size and quantity of documents on the Internet, the efficient algorithms are always in urgent need to get usable summarization or distill the key information. The documents will in versatile formats, structured or unstructured. The tasks include summarization of a single document or multiple documents. More extended target to extract summarization from multimedia files. Other challenges include summarizing multilingual documents. Abstraction requires the tool support from KNLP for grammar and lexicons for analyses and generation. One possible process for the abstraction is illustrated as follows:

Many approaches are suggested for automatic abstraction of document text, such as automatic extraction, automatic abstraction based on understanding, information extraction, and automatic abstraction based on structures. Possible features to be adapted to design the summarization system include the sentence length cutoff, fixed-phrase, paragraph, thematic word, and uppercase word features.

Abstraction or summarization popularly has been defined as a process with two steps. An intermediate representation of some sort is retrieved by the extraction of important concepts from the source texts. Given the intermediate representation, the summary is generated.

For the first step of the summarization process, it can largely be treated as part of automatic indexing. Lexical chains to extract important concepts from a document are one possible solution. Lexical chains exploit the cohesion among an arbitrary number of related words, and they are calculated by grouping (chaining) sets of words that are semantically related.

The CLARA algorithm

Instead of taking the whole set of data into consideration, the **CLARA** (**Clustering LARge Application**) algorithm randomly chooses a small portion of the actual data as a representative of the data. Medoids are then chosen from this sample using PAM. If the sample is selected in a fairly random manner, it should closely represent the original dataset.

CLARA draws multiple samples of the dataset, applies PAM to each sample, finds the medoids, and then returns its best clustering as the output. At first, a sample dataset D' is drawn from the original dataset D, and the PAM algorithm is applied to D' to find the k medoids. Use these k medoids and the dataset D to calculate the current dissimilarity. If it is smaller than the one you get in the previous iteration, then these k medoids are kept as the best k medoids. The whole process is performed a specified number of times.

The CLARA algorithm

The summarized pseudocode for the CLARA algorithm is as follows:

```
CLARA(X, d, k)
  bestDissim ← ∞
  for t ← 1 to S
  do X' ← RANDOM-SUBSET(X, s)
     D ← BUILD-DISSIM-MATRIX(X', d)
     (C', M) ← PAM(X', D, k)
     C ← ASSIGN-MEDOIDS(X, M, D)
     dissim ← TOTAL-DISSIM(C, M, D)
     if dissim < bestDissim
       then bestDissim ← dissim
            C_best ← C
            M_best ← M

  return (C_best, M_best)
```

Figure 1: Sequential CLARA algorithm

The R implementation

Take a look at the `ch_05_clara.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_clara.R")
```

CLARANS

CLARANS (**Clustering Large Applications based on RANdomized Search**) is efficient and effective and is the best practice for spatial data mining. CLARANS applies a strategy to search in a certain graph. A node in this graph, denoting it as $G_{n,k}$, is represented by a set of objects, $\left\{O_{m_1},\ldots,O_{m_k}\right\}, O_{m_1},\ldots,O_{m_k} \in D$. Here, *k* is the predefined value to choose the k medoids; as a result, the nodes in the graph are a set of $\left\{\left\{O_{m_1},\ldots,O_{m_k}\right\} \mid O_{m_1},\ldots,O_{m_k} \in D\right\}$. If two nodes, $S_1 = \left\{O_{m_1},\ldots,O_{m_k}\right\}$, and $S_2 = \left\{O_{w_1},\ldots,O_{w_k}\right\}$ are neighbors, then $\left|S_1 \cap S_2\right| = k-1$. Each node in the $G_{n,k}$ graph represents a set of medoids and the cluster related to it. As a result, a cost is related to each node; this cost is the total distance between any objects and the medoid represents its cluster. The cost differential of two neighbors can be calculated with the cost measure function introduced in the PAM algorithm.

The CLARANS algorithm

The input parameters for the CLARANS algorithm are as follows:

- `D`, which is the training tuples dataset
- `numlocal`
- `maxneighbor`

The output of the algorithm is:

- `bestnode`

The summarized pseudocode for the CLARANS algorithm is as follows:

```
1: Input parameters numlocal and maxneighbor. Initialize i to 1,
and mincost to a large number.

2: Set current to an arbitrary node in G_{n,k}.

#3: set j to 1.

#4: Consider a random neighbor S of current, and based on S,
calculate the cost differential of the two nodes.

#5: If S has a lower cost, set current to S, and go to Step (#3).

#6: Otherwise, increment j by 1. If j ≤ maxneighbor, go to step (#4).

#7: Otherwise, when j > maxneighbor, compare the cost of current
with mincost. If the cost < mincost, set mincost to the cost of
current, and set bestnode to current.

#8: Increment I by 1. If i > numlocal, output bestnode and halt.
Otherwise, go to Step (#2).
```

The R implementation

Take a look at the `ch_05_clarans.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_clarans.R")
```

Unsupervised image categorization and affinity propagation clustering

AP (**Affinity Propagation**) finds a set of exemplars, $X_e = \{x_{e_1}, \ldots, x_{e_k}\}$, in the dataset and assigns nonselected points to the exemplars. An exemplar is the representative of a cluster.

Two types of messages are exchanged between data objects or points; they are explained here:

- The *r(i,k)* message, which is called **responsibility**, represents the accumulated evidence sent from x_i to x_k. It informs us that x_k is suitable to serve as the exemplar of point x_i. Every candidate is counted in.
- The *a(i,k)* message, which is called **availability**, denotes the accumulated evidence sent from x_i to x_k. It informs us that x_k should be the exemplar. Every support from other points is well considered.

Both *r(i, k)* and *a(i,k)* are initialized as 0 at the beginning of the algorithm:

$$r(i,k) = s(i,k) - \max_{k',k'\neq k}\{a(i,k') + s(i,k')\}$$

$$r(k,k) = s(k,k) - \max_{k',k'\neq k}\{s(k,k')\}$$

$s(i,k) = -\|x_i - x_k\|^2$, $i \neq k$, for `s(k, k)` is initialized with the same value (typically defined with heuristic knowledge) for each point at the start and updated in the following description to recur the affection. `s(i, k)` denotes the extent to which x_j is suited to be the exemplar of x_i. Here is a possible value for `s(i, i)` to be set as a constant:

$$s(l,l) = \frac{\sum_{i,j=1;\, i\neq j}^{N} s(i,j)}{N*(N-1)}, 1 \leq l \leq N$$

$$a(i,k) = \min\{0, r(k,k) + \sum_{i',i'\neq i,k} \max\{0, r(i',k)\}\}$$

$$a(k,k) = \sum_{i',i'\neq i} \max\{0, r(i',k)\}$$

The index of exemplar, $e(x_i)$, which is for data point x_i, is defined with the following formula:

```
arg max {a(i,k) + r(i,k), k = 1,…, N}
```

Given `R = (r(i, j))` as the responsibility matrix and `A = (a(i, j))` as the availability matrix, t represents the iteration counts, where a damping factor $\lambda \in [0,1]$ is set to depress numerical oscillations that might arise:

$$R_{t+1} = (1-\lambda)R_t + \lambda R_{t-1}$$

$$A_{t+1} = (1-\lambda)A_t + \lambda A_{t-1}$$

Affinity propagation clustering

Affinity propagation itself is summarized as follows:

AFFINITY PROPAGATION

INPUT: $\{s(i,j)\}_{i,j\in\{1,\ldots,N\}}$ (data similarities and preferences)

INITIALIZE: set 'availabilities' to zero *i.e.* $\forall i,k\colon a(i,k)=0$

REPEAT: responsibility and availability updates until convergence

$$\forall i,k\colon\ r(i,k) = s(i,k) - \max_{k':k'\neq k}\left[s(i,k') + a(i,k')\right]$$

$$\forall i,k\colon\ a(i,k) = \begin{cases} \sum_{i':i'\neq i} \max[0,\ r(i',k)], & \text{for } k=i \\ \min\left[0,\ r(k,k) + \sum_{i':i'\notin\{i,k\}} \max[0,\ r(i',k)]\right], & \text{for } k\neq i \end{cases}$$

OUTPUT: cluster assignments $\hat{\mathbf{c}} = (\hat{c}_1,\ldots,\hat{c}_N)$, $\hat{c}_i = \operatorname{argmax}_k\left[a(i,k)+r(i,k)\right]$

Note: $\hat{c}$ may violate $\{f_k\}$ constraints, so initialize k-medoids with $\hat{c}$ and run to convergence for a coherent solution.

The R implementation

Take a look at the `ch_05_affinity_clustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_affinity_clustering.R")
```

Unsupervised image categorization

Due to the massive number of images and other multimedia documents, the task to classify images becomes even harder than before. Unsupervised image categorization is frequently utilized by image and video summarization, or it just serves as a preprocessing step in supervised methods for classification.

One major issue related to unsupervised image categorization is to estimate the distribution of image categories. Further on, finding the most descriptive prototypes of the image categories is another main issue of image categorization.

Each image can be represented as a high-dimensional data instance, including features related to color, texture, and shape. The exemplar technique is applied here; it represents image categories by a small set of image or its fragments. Given exemplar concepts, the dimension of an image data instance reduces to a relatively small size and eases further processing. The measures applied here include the Chamfer, Hausdorff, and shuffle distances.

Natural categories of the dataset can be of various complex types; overlapping might be a frequent shape.

Unsupervised image categorization or classification is a clustering problem. Image clustering is to identify a set of similar image primitives, such as pixels, line elements, regions, and so on. Given the complex dataset, the recommended way is to use the prototype-based clustering algorithm. Affinity propagation algorithms can be applied to unsupervised categorization by finding a subset of representative exemplars.

The spectral clustering algorithm

The summarized pseudocode for the spectral clustering algorithm is as follows:

Algorithm 16.1: Spectral Clustering Algorithm

```
SPECTRAL CLUSTERING (D, k):
Compute the similarity matrix A ∈ ℝ^{n×n}
if ratio cut then B ← L
else if normalized cut then B ← L^s or L^a
Solve Bu_i = λ_i u_i for i = n, ..., n − k + 1, where λ_n ≤ λ_{n−1} ≤ ··· ≤ λ_{n−k+1}
U ← (u_n  u_{n−1}  ···  u_{n−k+1})
Y ← normalize rows of U using (16.19)
C ← {C_1, ..., C_k} via K-means on Y
```

The R implementation

Take a look at the `ch_05_ spectral_clustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_ spectral_clustering.R")
```

News categorization and hierarchical clustering

Hierarchical clustering divides the target dataset into multilevels or a hierarchy of clusters. It segments data points along with successive partitions.

There are two strategies for hierarchical clustering. **Agglomerative clustering** starts with each data object in the input dataset as a cluster, and then in the following steps, clusters are merged according to certain similarity measures that end in only one cluster. **Divisive clustering**, in contrast, starts with one cluster with all the data objects in the input dataset as members, and then, the cluster splits according to certain similarity measures in the following steps, and at the end, singleton clusters of individual data objects are left.

The characteristics of hierarchical clustering methods are as follows:

- Multilevel decomposition.
- The merges or splits cannot perform a rollback. The error in the algorithm that is introduced by merging or splitting cannot be corrected.
- The hybrid algorithm.

Agglomerative hierarchical clustering

The summarized pseudocode for the agglomerative hierarchical clustering algorithm is as follows:

Algorithm 14.1: Agglomerative Hierarchical Clustering Algorithm

```
AGGLOMERATIVECLUSTERING(D, k):
C ← {C_i = {x_i} | x_i ∈ D} // Each point in separate cluster
Δ ← {δ(x_i, x_j): x_i, x_j ∈ D} // Compute distance matrix
repeat
    Find the closest pair of clusters C_i, C_j ∈ C
    C_ij ← C_i ∪ C_j // Merge the clusters
    C ← C \ {{C_i} ∪ {C_j}} ∪ {C_ij} // Update the clustering
    Update distance matrix Δ to reflect new clustering
until |C| = k
```

The BIRCH algorithm

The **BIRCH** (**Balanced Iterative Reducing and Clustering using Hierarchies**) algorithm is designed for large dynamic datasets and can also be applied to incremental and dynamic clustering. Only one pass of dataset is required, and this means that there is no need to read the whole dataset in advance.

CF-Tree is a helper data structure that is used to store the summary of clusters, and it is also a representative of a cluster. Each node in the CF-Tree is declared as a list of entries, $[CF_i, child_i], i \in [1, B]$, and B is the predefined entry limitation. $child_i$ denotes the link to the *i*th child.

Given CF_i as the representative of cluster *i*, is defined as $CF_i = (N_i, LS, SS)$, where N_i is the member count of the cluster, *LS* is the linear sum of the member objects, and *SS* is the squared sum of the member objects.

The leave of the CF-Tree must conform to the diameter limitation. The diameter of the $\left(\sum_{l=1}^{N_1}\sum_{m=1}^{N_1}(x_l - x_m)^2 / N_i(N_i - 1)\right)^{1/2}$ cluster must be less than the predefined threshold value, *T*.

The main helper algorithms in BIRCH are **CF-Tree insertion** and **CF-Tree rebuilding**.

The summarized pseudocode for the BIRCH algorithm is as follows:

The chameleon algorithm

The overall framework for the chameleon algorithm is illustrated in the following diagram:

There are three main steps in the chameleon algorithm, which are as follows:

- Sparsification: This step is to generate a k-Nearest Neighbor graph, which is derived from a proximity graph.
- Graph partitioning: This step applies a multilevel graph-partitioning algorithm to partition the dataset. The initial graph is an all-inclusive graph or cluster. This then bisects the largest graph in each successive step, and finally, we get a group of roughly, equally-sized clusters (with high intrasimilarity). Each resulting cluster has a size not more than a predefined size, that is, `MIN_SIZE`.

- Agglomerative hierarchical clustering: This last step is where the clusters are merged based on self-similarity. The notion of self-similarity is defined with the RI and RC; they can be combined in various ways to be a measure of self-similarity.

Relative Closeness (**RC**), given the C_i and C_j clusters with sizes m_i and m_j, respectively. $\overline{S}_{EC}\left(C_i, C_j\right)$ denotes the average weight of the edges that connect the two clusters. $EC\left(C_i\right)\left(EC\left(C_j\right)\right)$ denotes the average weight of the edges that bisect the $C_i\left(C_j\right)$ cluster:

$$RC(C,C) = \overline{S}_{EC}\left(C_i, C_j\right) / \left(\frac{m_i}{m_i + m_j} S_{EC}\left(C_i\right) + \frac{m_i}{m_i + m_j} S_{EC}\left(C_j\right) \right)$$

Relative Interconnectivity (**RI**) is defined in the following equation. Here, $EC\left(C_i, C_j\right)$ is the sum of the edges to connect the two clusters, and $EC\left(C_i\right)\left(EC\left(C_j\right)\right)$ is the minimum sum of the cut edges that bisect the $C_i\left(C_j\right)$ cluster:

$$RI\left(C_i, C_j\right) = EC(C_{i,} C_j) / \frac{1}{2}\left(EC(C_i) + EC(C_j)\right)$$

The summarized pseudocode for the chameleon algorithm is as follows:

```
1: Build a k-nearest neighbor graph.

2: Partition the graph using a multilevel graph-partitioning
algorithm.

3: do {

4:   Merge the clusters that best preserve the cluster
self-similarity with respect to relative interconnectivity and
relative closeness.

5:} while No more clusters can be merged.
```

The Bayesian hierarchical clustering algorithm

The summarized pseudocode for the Bayesian hierarchical clustering algorithm is as follows:

input: data $\mathcal{D} = \{x^{(1)} \ldots x^{(n)}\}$, model $p(x|\theta)$,
prior $p(\theta \mid \beta)$
initialize: number of clusters c=n, and
$\mathcal{D}_i = \{x^{(i)}\}$ for $i=1 \ldots n$
while $c > 1$ **do**
Find the pair $\mathcal{D}_i$ and $\mathcal{D}_j$ with the highest probability of the merged hypothesis:

$$r_k = \frac{\pi_k p(\mathcal{D}_k | \mathcal{H}_1^k)}{p(\mathcal{D}_k | T_k)}$$

Merge $\mathcal{D}_k \leftarrow \mathcal{D}_i \cup \mathcal{D}_j$, $T_k \leftarrow (T_i, T_j)$
Delete $\mathcal{D}_i$ and $\mathcal{D}_j$, $c \leftarrow c - 1$
end while
output: Bayesian mixture model where each tree node is a mixture component
The tree can be cut at points where $r_k < 0.5$

Figure 2. Bayesian Hierarchical Clustering Algotithm

The probabilistic hierarchical clustering algorithm

The probabilistic hierarchical clustering algorithm is a hybrid of hierarchical clustering and probabilistic clustering algorithms. As a probabilistic clustering algorithm, it applies a completely probabilistic approach, as depicted here:

Algorithm: A probabilistic hierarchical clustering algorithm.

Input:

- $D = \{\boldsymbol{o_1}, \ldots, \boldsymbol{o_n}\}$: a data set containing n objects;

Output: A hierarchy of clusters.

Method:

(1) **create** a cluster for each object $C_i = \{\boldsymbol{o_i}\}$, $1 \leq i \leq n$;
(2) **for** $i = 1$ to n
(3) **find** pair of clusters C_i and C_j such that $C_i, C_j = \arg\max_{i \neq j} \log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)}$;
(4) **if** $\log \frac{P(C_i \cup C_j)}{P(C_i)P(C_j)} > 0$ then merge C_i and C_j;
(5) **else** stop;

The R implementation

Take a look at the `ch_05_ahclustering.R` R code file from the bundle of R code for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_05_ahclustering.R")
```

News categorization

News portals provide visitors with tremendous news categorized with a predefined set of topics. With the exponential growth of information we receive from the Internet, clustering technologies are widely used for web documents categorization, which include online news. These are often news streams or news feeds.

One advantage of a clustering algorithm for this task is that there is no need for prior domain knowledge. News can be summarized by clustering news that covers specific events. Unsupervised clustering can also play a pivotal role in discovering the intrinsic topical structure of the existing text collections, while new categorization is used to classify the documents according to a predefined set of classes.

Services such as Google News are provided. For a story published by a couple of new agencies or even the same agency, there are often various versions that exist at the same time or span the same range of days. Here, clustering can help aggregate the news related to the same story, and the result is a better overview of the current story for visitors.

As preprocessing steps, plain texts are extracted from the web pages or news pages. Reuters-22173 and -21578 are two popular documents dataset used for research. The representation of a data instance in the dataset can either be classical term-document vectors or term-sentence vectors, as the dataset is a collection of documents, and the cosine-like measures are applied here.

Time for action

Here are some practice questions for you to check whether you understood the concepts we covered in this chapter:

1. What is the PAM clustering algorithm?
2. What is the k-means algorithm?
3. What is the k-medoids algorithm?
4. What is the CLARA algorithm?
5. What is the CLARANS algorithm?

Summary

In this chapter, we looked at:

- Partition-based clustering.
- The k-means algorithm is a partition-based clustering algorithm. The centroids of clusters are defined as a representative of each cluster. In k-means clustering, a set of n data points in a D-dimensional space and an integer k are given. The problem is to distribute a set of k points in the centers to minimize the SSE.
- The k-medoids algorithm is a partition-based clustering algorithm. The representatives of each resulting clusters are chosen from the dataset itself, that is, the data objects belong to it.
- CLARA depends on sampling. It draws a sample from the original dataset instead of the entire dataset. PAM is then applied to each sampling. Then, the best result is kept during all the iterations.
- CLARANS is a clustering algorithm based on randomized search.

- The affinity propagation clustering algorithm recursively passes affinity messages between objects or points and converges to exemplars adaptively.
- Spectral clustering is used to construct graph partitions based on eigenvectors of the adjacency matrix.
- Hierarchical clustering decomposes a dataset *D* into levels of nested clusters; this is represented by a **dendrogram**, a tree that iteratively splits the dataset *D* into smaller subsets. The process stops only after each subset consists of only one object.

The next chapter will cover more advanced topics related to clustering algorithms, density-based algorithms, grid-based algorithms, the EM algorithm, high-dimensional algorithms, constraint-based clustering algorithms, and so on.

6
Advanced Cluster Analysis

In this chapter, you will learn how to implement the top algorithms for clusters with R. The evaluation/benchmark/measure tools are also provided.

In this chapter, we will cover the following topics:

- Customer categorization analysis of e-commerce and DBSCAN
- Clustering web pages and OPTICS
- Visitor analysis in the browser cache and DENCLUE
- Recommendation system and STING
- Web sentiment analysis and CLIQUE
- Opinion mining and WAVE CLUSTER
- User search intent and the EM algorithm
- Customer purchase data analysis and clustering high-dimensional data
- SNS and clustering graph and network data

Customer categorization analysis of e-commerce and DBSCAN

By defining the density and density measures of data point space, the clusters can be modeled as sections with certain density in the data space.

Density Based Spatial Clustering of Applications with Noise (**DBSCAN**) is one of the most popular density-based clustering algorithms. The major characteristics of DBSCAN are:

- Good at dealing with large datasets with noises
- Clusters of various shapes can be dealt with

DBSCAN is based on the classification of the data points in the dataset as core data points, border data points, and noise data points, with the support of the use of density relations between points, including directly density-reachable, density-reachable, density-connected points. Before we provide a detailed description of DBSCAN, let's illustrate these ideas.

A point is defined as a core point if the number of data points within the distance of the predefined parameter, **Eps** or $\in$, is greater than that of the predefined parameter, **MinPts**. The space within the Eps is called Eps-neighborhood or $N_{\in}(q)$. An object, *o*, is noise only if there is no cluster that contains *o*. A border data object is any object that belongs to a cluster, but not a core data object.

Given a core object, *p*, and an object, *q*, the object, *q*, is **directly density-reachable** from *p* if $q \in N_{\in}(p)$.

Given a core object, *p*, and an object, *q*, the object *q* is **density-reachable** from *p* if $\exists a\, chain\, of\ data\, object\ p_1,..,p_n, p_1 = q, p_n = p, \forall 1 \le i < n$ and $p_i\ is\ directly\ density-reachable\ from\ p$.

For two data objects, q_1 and q_2, they are **density-connected** if $\exists a\, core\, object\ p, q_1\ and\ q_2\ are\, density-reachable\, to\ p$.

The **density-based cluster** denotes a set of density-connected data objects that are maximal according to density reachability.

Here is an example illustrated in the following diagram:

The DBSCAN algorithm

The summarized pseudocode for the DBSCAN algorithm is as follows, with the following input/output parameters defined.

The input parameters for the DBSCAN algorithm are as follows:

- *D*, which is the training tuples dataset
- *k*, which is the neighbor list size
- *Eps*, which is the radius parameter that denotes the neighborhood area of a data point
- *MinPts*, which is the minimum number (the neighborhood density threshold) of points that must exist in the Eps-neighborhood
- The output of the algorithm
- A set of density-based clusters

$\textbf{DBSCAN}\ (\mathbf{D}, \varepsilon,\ minpts)$:

$Core \leftarrow \emptyset$

$\textbf{foreach}\ \mathbf{x}_i \in \mathbf{D}\ \ \textbf{do}$

  Compute $N_\epsilon(\mathbf{x}_i)$

  $id(\mathbf{x}_i) \leftarrow \emptyset$

  $\textbf{if}\ N_\varepsilon(\mathbf{x}_i) \geq minpts\ \textbf{then}\ \ Cores \leftarrow Cores \cup \{\mathbf{x}_i\}$

$k \leftarrow 0$

$\textbf{foreach}\ \mathbf{x}_i \in Core,\ such\ that\ id(\mathbf{x}_i) = \emptyset\ \textbf{do}$

  $k \leftarrow k + 1$

  $id(\mathbf{x}_i) \leftarrow k$

  $\textsc{DensityConnected}\ (\mathbf{x}_i, k)$

$\mathcal{C} \leftarrow \{C_i\}_{i=1}^{k}$, where $C_i \leftarrow \{\mathbf{x} \in \mathbf{D} \mid\ id(\mathbf{x}) = i\}$

$Noise \leftarrow \{\mathbf{x} \in \mathbf{D} \mid\ id(\mathbf{x}) = \emptyset\}$

$Border \leftarrow \mathbf{D} \setminus \{Core \cup Noise\}$

$\textbf{return}\ \mathcal{C}, Core, Border, Noise$

$\textbf{DensityConnected}\ (\mathbf{x},\ k)$:

$\textbf{foreach}\ \mathbf{y} \in N_\varepsilon(\mathbf{x})\ \textbf{do}$

  $id(\mathbf{y}) \leftarrow k$

  $\textbf{if}\ \mathbf{y} \in Core\ \textbf{then}\ \ \textsc{DensityConnected}\ (\mathbf{y}, k)$

Customer categorization analysis of e-commerce

Customers of e-commerce can be categorized by the psychographic, culturally specific purchasing behavior. The result of the customer categorization can make the storeowner efficiently and effectively respond to the customer. The e-commerce general analysis process is illustrated as follows:

Clustering web pages and OPTICS

Ordering points to identify the clustering structure, OPTICS, extends the DBSCAN algorithm and is based on the phenomenon that density-based clusters, with respect to a higher density, are completely contained in density-connected sets with respect to lower density.

To construct density-based clusters with different densities simultaneously, the objects are dealt with in a specific order when expanding a cluster, that is, according to the order, an object that is density-reachable with respect to the lowest $\in$ for the higher density clusters will be finished first.

Two major concepts are introduced to illustrate the OPTICS algorithm: **core-distance** of an object, *p*, and **reachability-distance** of object, *p*. An example is illustrated in the following image. During which, core-distance (○), reachability-distances, $r(p_1, o)$, $r(p_2, o)$, for **MinPts=4**.

The **core-distance** of an object, `p`, is denoted as the minimal value, $\in'$, considering that the $\in'$-neighborhood at least contains `MinPts` data objects; otherwise, this is undefined.

Given two data objects, *p* and *q*, the **reachability-distance** of object, *p*, from *q* represents the smallest radius value that makes *p* density-reachable from *q* if *q* is a core data object; otherwise, this is undefined.

OPTICS outputs an ordering of all data objects in a given dataset in which the data objects are processed. For each object, the core-distance and a suitable reachability-distance for each data object are calculated and output together.

The OPTICS algorithm

OPTICS randomly selects an object from the input dataset as the current data object, *p*. In addition, for the object, *p*, the $\in$-neighborhood is fetched, the core-distance is calculated, and the reachability-distance is undefined.

Given *p* as a core data object, OPTICS recalculates the reachability-distance for any object, *q* (in the neighborhood of *p*), from *p* and inserts *q* into OrderSeeds if *q* has not been processed.

Once the augmented cluster ordering of the input dataset is generated with respect to $\in$, `MinPts` and a clustering-distance $\in' \leq \in$, the density-based clustering is performed with this order.

The summarized pseudocode for the OPTICS algorithm with a couple of supporter functions are as follows:

```
OPTICS (SetOfObjects, ε, MinPts, OrderedFile)
   OrderedFile.open();
   FOR i FROM 1 TO SetOfObjects.size Do
     Object := SetOfObject.get(i);
     IF NOT Object.Processed THEN
        ExpandClusterOrder(SetOfObjects, Object, ε,
              MinPts, OrderedFile)
   OrderedFile.close();
END; //OPTICS
```

```
ExpandClusterOrder (SetOfObjects, Objects, MinPts,)
OrderedFile);
  neighbors := SetOfObjects.neighbors(Object, ε);
  Object.Processed := TRUE;
  Object.reachability_distance :=UNDEFINED;
  Object.setCoreDistance(neighbors,   , MinPts)
  OrderedFile.write(Object);
  IF Object.core_distance<> UNDEFINED THEN
     OrderSeeds.update(neighbors,Object);
     WHILE NOT OrderSeed.empty()DO
        currentObject :=OrderSeeds.next()
        neighbors:=SetOfObject.neighbors(currentObject, ε);
        currentObject.Processed :=TRUE;
        currentObject.setCoreDistance(neighbors, ε,MinPts)
        OrderdFile.write(currentObject);
        IF currentObject.core_distance<>UNDEFINED THEN
          OrderSeeds.update(neighbore, currentObject);
END; // ExpandClusterOrder
```

```
OrderSeeds::update(neightbors, CenterObject);
  c_dist :=CenterObject.core_distance;
  FORALL Object FROM neighbors DO
     IF NOT Object.Processed THEN
        new_r_dist:=max(c_dist,CenterObject.dist(Object));
        IF Object.reachability_distance=UNDEFINED THEN
           Object.reachability_distance :=new_r_dist;
           insert(Object, new_r_dist);
        ELSE // Object already inOrderSeeds
           IF new_r_dist<Object.reachability_distance THEN
              Object.reachability_distance :=new_r_dist;
              decrease(Object, new_r_dist);
END; //OrderSeeds::update
```

```
ExractDBSCAN-Clustering(ClusterOrderedObjs, ε', MinPts)
// Precondition: ε '≤ generating dist ε for ClusterOrderedObjs
   ClusterId := NOISE;
   FOR i FROM 1 TO ClusterOrderedObjs.size Do
     Object :=ClusterOrderedObjs.get(i);
     IF Object.reachability_distance>ε' THEN
       // UNDEFINED >ε
       IF Object.core_distance ≤ ε' THEN
           ClusterId := nextId(ClusterId);
           Object.clusterId := ClusterId;
        ELSE
           Object.clusterId := NOISE;
      ELSE     // Object.reachability_distance ≤ ε'
           Object.clusterId := ClusterId;
 END; // ExtractDBSCAN-Clustering
```

The R implementation

Please take a look at the R codes file `ch_06_optics.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_optics.R")
```

Clustering web pages

Clustering web pages can be used to group related texts/articles and serve as the preprocessing steps for supervised learning. It enables automated categorization.

Web pages are versatile and have a different structure (well-formed or badly formed without structure) and contents.

The Yahoo! industry web page data (CMUWeb KBProject, 1998) is used as the test data here.

Visitor analysis in the browser cache and DENCLUE

DENsity-based CLUstEring (**DENCLUE**) is a density-based clustering algorithm that depends on the support of density-distribution functions.

Before a detailed explanation on the DENCLUE algorithm, some concepts need to be introduced; they are **influence function**, **density function**, **gradient**, and **density attractor**.

The influence function of a specific data object can be any function for which the Gaussian kernel is usually used as the kernel at the data point.

The density function at a point, x, is defined as the sum of the influence functions of all the data objects at this data point.

A point is defined as a density attractor if it is a local maximum of the density function and is computed as

$$x^0 = x$$

$$x^{i+1} = x^i + \delta * \frac{\nabla f_B^D\left(x^i\right)}{\left\|\nabla f_B^D\left(x^i\right)\right\|}$$

A gradient of the density function is defined in the following equation, given the density function, $f_B^D\left(x\right)$.

$$\nabla f_B^D\left(x\right) = \sum_{i=1}^{N}\left(x_i - x\right) * f_B^{x_i}\left(x\right)$$

DENCLUE defines a density function for the data point space at first. All the local maxima data points are searched and found. Assign each data point to the nearest local maxima point to maximize the density related to it. Each group of data points bound with a local maxima point is defined as a cluster. As a postprocess, the cluster is discarded if its bound local maxima density is lower than the user-predefined value. The clusters are merged if there exists a path such that each point on the path has a higher density value than the user-predefined value.

The DENCLUE algorithm

The summarized pseudocodes for the DENCLUE algorithm is as follows:

$\textsc{denclue}\ (\mathbf{D}, h, \xi, \varepsilon)$:
$\mathcal{A} \leftarrow \emptyset$
foreach $\mathbf{x} \in \mathbf{D}$ **do**
  $\mathbf{x}^* \leftarrow \textsc{FindAttractor}(\mathbf{x}, \mathbf{D}, h, \varepsilon)$
  if $\hat{f}(\mathbf{x}^*) \geq \xi$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathbf{x}^*\}$
    $R(\mathbf{x}^*) \leftarrow R(\mathbf{x}^*) \cup \{\mathbf{x}\}$

$\mathcal{C} \leftarrow \{\text{maximal } C \subseteq \mathcal{A} \mid \forall \mathbf{x}_i^*, \mathbf{x}_j^* \in C, \mathbf{x}_i^* \text{ and } \mathbf{x}_j^* \text{ are density reachable}\}$
foreach $C \in \mathcal{C}$ **do**
  foreach $\mathbf{x}^* \in C$ **do** $C \leftarrow C \cup R(\mathbf{x}^*)$

return $\mathcal{C}$

$\textsc{FindAttractor}\ (\mathbf{x}, \mathbf{D}, h, \epsilon)$:
$t \leftarrow 0$
$\mathbf{x}_t \leftarrow \mathbf{x}$
repeat
  $\mathbf{x}_{t+1} \leftarrow \frac{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right) \cdot \mathbf{x}_t}{\sum_{i=1}^{n} K\left(\frac{\mathbf{x}_t - \mathbf{x}_i}{h}\right)}$
  $t \leftarrow t + 1$
until $\|\mathbf{x}_t - \mathbf{x}_{t-1}\| \leq \varepsilon$
return $\mathbf{x}_t$

The R implementation

Please take a look at the R codes file `ch_06_denclue.R` from the bundle of R codes for previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_denclue.R")
```

Visitor analysis in the browser cache

The browser-cache analysis provides the website owner with the convenience that shows the best matched part to the visitors, and at the same time, it is related to their privacy protection. The data instances in this context are browser caches, sessions, cookies, various logs, and so on.

The possible factors included in certain data instances can be the Web address, IP address (denotes the position where the visitor comes from), the duration for which the visitor stayed on a specific page, the pages the user visited, the sequence of the visited pages, the date and time of every visit, and so on. The log can be specific to a certain website or to various websites. A more detailed description is given in the following table:

Hit	This refers to each element of a web page downloaded to a viewer's web browser (such as Internet Explorer, Mozilla, or Netscape). Hits do not correspond in any direct fashion to the number of pages viewed or number of visitors to a site. For example, if a viewer downloads a web page with three graphics, the web logfile will show four hits: one for the web page and one for each of the three graphics.
Unique Visitors	The actual number of viewers to the website that came from a unique IP address (see IP address in this table).
New/Return Visitors	The number of first-time visitors to the site compared to returning visitors.
Page views	The number of times a specified web page has been viewed; shows exactly what content people are (or are not) viewing at a website. Every time a visitor hits the page refresh button, another page view is logged.
Page views per visitor	The number of page views divided by the number of visitors; measures how many pages viewers look at each time they visit a website.
IP address	A numeric identifier for a computer. (The format of an IP address is a 32-bit numeric address written as four numbers separated by periods; each number can be zero to 255. For example, 1.160.10.240 could be an IP address.) The IP address can be used to determine a viewer's origin (that is, by country); it also can be used to determine the particular computer network a website's visitors are coming from.
Visitor location	The geographic location of the visitor.
Visitor language	The language setting on the visitor's computer.
Referring pages/sites (URLs)	Indicates how visitors get to a website (that is, whether they type the URL, or web address, directly into a web browser or if they click through from a link at another site).

Keywords	If the referring URL is a search engine, the keywords (search string) that the visitor used can be determined.
Browser type	The type of browser software a visitor is using (that is, Netscape, Mozilla, Internet Explorer, and so on)
Operating system version	The specific operating system the site visitor uses.
Screen resolution	The display settings for the visitor's computer.
Java or Flash-enabled	Whether or not the visitor's computer allows Java (a programming language for applications on the Web) and/or Flash (a software tool that allows web pages to be displayed with animation, or motion).
Connection speed	Whether visitors are accessing the website from a slower dial-up connection, high-speed broadband, or Tl.
Errors	The number of errors recorded by the server, such as a "404-file not found" error; can be used to identify broken links and other problems at the website.
Visit duration	Average time spent on the site (length the visitor stays on the site before leaving). Sites that retain visitors longer are referred to as "sticky" sites.
Visitor paths/navigation	How visitors navigate the website, by specific pages, most common entry pages (the first page accessed by a visitor at a website) and exit points (the page from which a visitor exits a Website), and so on. For example, if a large number of visitors leave the site after looking at a particular page, the analyst might infer that they either found the information they needed, or alternatively, there might be a problem with that page (is it the page where shipping and handling fees are posted, which maybe are large enough to turn visitors away?).
Bounce rate	The percentage of visitors who leave the site after the first page; calculated by the number of visitors who visit only a single page divided by the number of total visits. The bounce rate is sometimes used as another indicator of "stickiness."

The analysis of a visitor is basically history sniffing, which is used for user-behavior analysis.

Recommendation system and STING

STatistical Information Grid (**STING**) is a grid-based clustering algorithm. The dataset is recursively divided into a hierarchy structure. The whole input dataset serves as the root node in the hierarchy structure. Each cell/unit in a layer is composed of a couple of cells/units in the lower layer. An example is shown in the following diagram:

To support the query for a dataset, the statistical information of each unit is calculated in advance for further processing; this information is also called statistics parameters.

The characteristics of STING algorithms are (but not limited to) the following:

- A query-independent structure
- Intrinsically parallelizable
- Efficiency

The STING algorithm

The summarized pseudocodes for the STING algorithm are as follows:

1. Determine a layer to begin with.
2. For each cell of this layer, we calculate the confidence interval (or estimated range) of probability that this cell is relevant to the query.
3. From the interval calculated above, we label the cell as *relevant* or *not relevant*.
4. If this layer is the bottom layer, go to Step 6; otherwise, go to Step 5.
5. We go down the hierarchy structure by one level. Go to Step 2 for those cells that form the *relevant* cells of the higher level layer.
6. If the specification of the query is met, go to Step 8; otherwise, go to Step 7.
7. Retrieve those data fall into the *relevant* cells and do further processing. Return the result that meet the requirement of the query. Go to Step 9.
8. Find the regions of *relevant* cells. Return those regions that meet the requirement of the query. Go to Step 9.
9. Stop.

The R implementation

Please take a look at the R codes file `ch_06_sting.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_sting.R")
```

Recommendation systems

Depending on statistical, data-mining, and knowledge-discovery techniques, recommendation systems are being used by most of the e-commerce sites to make it easy for consumers to find products to purchase. Three main parts: representation of input data, neighborhood formation, and recommendation generation are shown in the following diagram:

Web sentiment analysis and CLIQUE

CLustering In QUEst (**CLIQUE**) is a bottom-up and grid-based clustering algorithm. The idea behind this algorithm is the Apriori feature, that is, the monotonicity of dense units with respect to dimensionality. If a set of data points, *S*, is a cluster in a k-dimensional projection of the space, then *S* is also contained in a cluster in any (k-1)-dimensional projections of this space.

The algorithm proceeds by passes. The one-dimensional dense units are produced by one pass through the data. The candidate k-dimensional units are generated using the candidate-generation procedure and the determined (k-l)-dimensional dense units that are fetched at the (k-1) pass.

The characteristics of the CLIQUE algorithm are as follows:

- Efficient for high-dimensional datasets
- Interpretability of results
- Scalability and usability

The CLIQUE algorithm contains three steps to cluster a dataset. First, a group of subspaces is selected to cluster the dataset. Then, clustering is independently executed in every subspace. Finally, a concise summary of every cluster is produced in the form of a **disjunctive normal form** (**DNF**) expression.

The CLIQUE algorithm

The summarized pseudocode for the CLIQUE algorithm is as follows:

1. Indentification of subspaces that contain clusters.
2. Indentification of cluster.
3. Generation minimal description for the cluster.

The candidate-generation algorithm is illustrated as follows:

insert into C_k
Select $u_1.[l_1, h_1)$, $u_1.[l_2, h_2)$,...,
$u_1.[l_{k-1}, h_{k-1})$, $u_2.[l_{k-1}, h_{k-1})$
from D_{k-1} u_1, D_{k-1} u_2
Where $u_1.a_1 = u_2.a_1$, $u_1.l_1 = u_2.l_1$, $u_1.h_1 = u_2.h_1$,
$u_1.a_2 = u_2.a_2$, $u_1.l_2 = u_2.l_2$, $u_1.h_2 = u_2.h_2$,...,
$u_1.a_{k-2} = u_2.a_{k-2}$, $u_1.l_{k-2} = u_2.l_{k-2}$, $u_1.h_{k-2} = u_2.h_{k-2}$,
$u_1.a_{k-1} < u_2.a_{k-2}$

Here is the algorithm to find the connected components of the graph; this is equivalent to finding clusters:

```
input: starting unit u= {[l_1, h_1),...[l_k, h_k)}
        clusternumber n
dfs (u, n)
u. num=n
for ( j = 1; j < k; j ++) do begin
    ul = {[ l1, h1 ),... [(l_j^l), (h_j^l)),..., [l_k , h_k )}
    if( u^l is dense ) and ( u^l. num is undefined )
    dfs (u^l, n)
    ur = {[ l1, h1 ), ..., [( l_j^r), (h_j^r)),...,[ l_k , h_k )}
    if( u^r is dense ) and ( u^r. num is undefined )
    dfs (u^r, n)
end
```

The R implementation

Please take a look at the R codes file `ch_06_clique.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_clique.R")
```

Web sentiment analysis

Web sentiment analysis is used to identify the idea or thought behind the text, for example, the sentiment analysis of microblogs such as Twitter. One simple example is comparing the post with a predefined labeled word list for sentiment judging. Another example is that we can judge a movie review as thumbs up or thumbs down.

Web sentiment analyses are used in news article analyses of biases pertaining to specific views, newsgroup evaluation, and so on.

Opinion mining and WAVE clustering

The WAVE clustering algorithm is a grid-based clustering algorithm. It depends on the relation between spatial dataset and multidimensional signals. The idea is that the cluster in a multidimensional spatial dataset turns out to be more distinguishable after a wavelet transformation, that is, after applying wavelets to the input data or the preprocessed input dataset. The dense part segmented by the sparse area in the transformed result represents clusters.

The characteristics of the WAVE cluster algorithm are as follows:

- Efficient for a large dataset
- Efficient for finding various shapes of clusters
- Insensitive to noise or outlier
- Insensitive with respect to the input order of a dataset
- Multiresolution, which is introduced by wavelet transforms
- Applicable to any numerical dataset

The WAVE cluster algorithm performs a few steps. In the first step, it creates a grid and assigns each data object from the input dataset to a unit in the grid. And then, transform the data to a new space by applying wavelet transform functions. Third, find the connected component in the new space. Map the cluster label for related data object to the original data space.

The WAVE cluster algorithm

The summarized pseudocodes for the WAVE cluster algorithm are as follows:

Input: Multidimentional data objects' fealure vectors
Output: clustered objects

1. Quantize feature space, then assign objects to the units.
2. Apply wavelet transform on the feature space.
3. Find the connected componebts (clusters) in the subbands of transformed feature space, at different levels.
4. Assign label to the units.
5. Make the lookup table.
6. Map the object to the clusters.

The R implementation

Please take a look at the R codes file `ch_06_wave.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_wave.R")
```

Opinion mining

An entity possesses a couple of features. Features might be explicit or implicit. If a person or organization expresses an opinion, then the person or organization should be an opinion holder. An opinion specific to a feature is a positive or negative view, attitude, emotion, or appraisal of the feature from the opinion holder. Whether the opinion on a feature is positive, negative, or neutral denotes opinion orientation.

Opinion mining means mining the opinion on certain features of the object or entity under research. The simplest case is to judge the opinion as positive or negative.

An opinion-orientation algorithm can be listed as follows:

- Identify opinion words and phrases
- Handle negation
- The But clause
- Aggregating opinion

$$Score\left(f_i, s\right) = \sum_{op_j \in s} \frac{op_j \cdot oo}{d\left(op_j, f_i\right)}$$

The preceding formula denotes the opinion orientation on a certain feature, f_i, where op_j is an opinion word in s, $d\left(op_j, f_i\right)$ is the distance between f_i and op_j. $op_j \cdot oo$ is the opinion orientation.

User search intent and the EM algorithm

The **Expectation Maximization** (**EM**) algorithm is a probabilistic-model-based clustering algorithm that depends on the mixture model in which the data is modeled by a mixture of simple models. The parameters related to these models are estimated by **Maximum Likelihood Estimation** (**MLE**).

Mixture models assume that the data is the result of the combination of various simple probabilistic distribution functions. Given *K* distribution functions and the *j*th distribution with the parameter, θ_j, Θ is the set of θ_j of all distributions:

$$p\left(x \middle| \Theta\right) = \sum_{j=1}^{K} w_j p_j(x \mid \theta_j)$$

The EM algorithm performs in the following way. In the first step, an initial group of model parameters are selected. The expectation step is the second step that performs the calculation of the probability:

$$P\left(distribution\ \ j \mid x_i, \theta\right) = \frac{P\left(x_i \mid \theta_j\right)}{\sum_{l=1}^{K} P\left(x_i \mid \theta_l\right)}$$

The previous equation represents the probability of each data object belonging to each distribution. Maximization is the third step. With the result of the expectation step, update the estimation of the parameters with the ones that maximize the expected likelihood:

$$\mu_j = \frac{1}{k} \sum_{i=1}^{n} o_i \frac{P(\Theta_j \middle| \mathrm{o}_i, \Theta)}{\sum_{l=1}^{n} P(\Theta_j \middle| \mathrm{o}_l, \Theta)} = \frac{1}{k} \frac{\sum_{i=1}^{n} o_i P(\Theta_j \middle| \mathrm{o}_i, \Theta)}{\sum_{i=1}^{n} P(\Theta_j \middle| \mathrm{o}_i, \Theta)}$$

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{n} p(\Theta_j \middle| \mathrm{o}_i, \Theta)(\mathrm{o}_i - u_j)^2}{\sum_{i=1}^{n} p(\Theta_j \middle| \mathrm{o}_i, \Theta)}}$$

The expectation step and maximization step are performed repeatedly until the output matches the ending condition, that is, the changes of the parameter estimations below a certain threshold.

The EM algorithm

The summarized pseudocode for the EM algorithm are as follows:

$$
\begin{array}{l}
\textbf{Expectation-Maximization } (\mathbf{D}, k, \varepsilon): \\
t \leftarrow 0 \\
\text{Randomly initialize } \boldsymbol{\mu}_1^t, \cdots, \boldsymbol{\mu}_k^t \\
\boldsymbol{\Sigma}_i^t \leftarrow \mathbf{I},\ \forall i = 1, \ldots, k \\
P^t(C_i) \leftarrow \frac{1}{k},\ \forall i = 1, \ldots, k \\
\textbf{repeat} \\
\quad t \leftarrow t + 1 \\
\quad \texttt{// Expectation Step} \\
\quad \textbf{for } i = 1, \ldots, k \ \textit{and}\ j = 1, \ldots, n \textbf{ do} \\
\qquad w_{ij}^t \leftarrow \frac{f(\mathbf{x}_j | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \cdot P(C_i)}{\sum_{a=1}^k f(\mathbf{x}_j | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_a) \cdot P(C_a)} \\
\quad \texttt{// Maximization Step} \\
\quad \textbf{for } i = 1, \ldots, k \textbf{ do} \\
\qquad \boldsymbol{\mu}_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} \cdot \mathbf{x}_j}{\sum_{j=1}^n w_{ij}} \\
\qquad \boldsymbol{\Sigma}_i^t \leftarrow \frac{\sum_{j=1}^n w_{ij} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T}{\sum_{j=1}^n w_{ij}} \\
\qquad P^t(C_i) \leftarrow \frac{\sum_{j=1}^n w_{ij}}{n} \\
\textbf{until } \sum_{i=1}^k \left\| \boldsymbol{\mu}_i^t - \boldsymbol{\mu}_i^{t-1} \right\| \leq \varepsilon
\end{array}
$$

The R implementation

Please take a look at the R codes file `ch_06_em.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_em.R")
```

The user search intent

Determining the user search intent is an important yet a difficult issue with respect to the sparse data available concerning the searcher and queries.

User intent has a wide range of applications, cluster query refinements, user intention profiles, and web search intent induction. Given Web search engine queries, finding user intention is also a key and requirement.

To determine the user interest and preferences, the clicks' sequence upon the search result can be used as a good base.

Web search personalization is another important application with the user search intention. It is related to user context and intention. With the user intention applied, more effective and efficient information will be provided.

Customer purchase data analysis and clustering high-dimensional data

For high-dimensional data-space clustering, two major issues occur: efficiency and quality. New algorithms are needed to deal with this type of dataset. Two popular strategies are applied to it. One is the subspace-clustering strategy to find the cluster in the subspace of the original dataset space. Another is the dimensionality-reduction strategy, which is a lower dimensional data space created for further clustering.

MAFIA is an efficient and scalable subspace-clustering algorithm for high-dimensional and large datasets.

The MAFIA algorithm

The summarized pseudocode for the MAFIA algorithm is as follows:

```
Adaptive Grid Computation
|A_i| - Cardinality of A_i
for each dimension A_i, i ∈ d
    Divide |A_i| into windows of some small size x
    Compute the histogram for each unit of A_i, and set the value of the window to the maximum
    in the window
    From left to right merge two adjacent units if they are within a threshold β
    /* If number of bins is one, we have an equi-distributed dimension */
    if(number of bins == 1)
        Divide the dimension A_i into a fixed number of equal partitions and set a threshold β'
        for it
end
```

The summarized pseudocode for the parallel MAFIA algorithm is as follows:

Parallel MAFIA Algorithm

N - Number of records
p - Number of processors
d - Dimensionality of data
A_i - i^{th} attribute $i \in d$
B - Number of records that fit in memory buffer allocated at each processor
/* Each processor reads $\frac{N}{p}$ data from its local disk */
On each processor
    Read $\frac{N}{pB}$ chunks of B records from local disk and build a histogram in each dimension $A_i, i \in d$
    Reduce communication to get the global histogram
    Determine adaptive intervals using the histogram in each dimension $A_i, i \in d$ and also fix the threshold level
    Set candidate dense units to the bins found in each dimension
    Set current dimensionality, k to 1
    while (no more dense units are found)
        if $(k > 1)$
            Build candidate dense units in k from the dense units in $(k-1)$ which share the $(k-2)$-dimensional edges
        Read $\frac{N}{pB}$ chunks of B records from local disk and for every record populate the candidate dense units
        Reduce communication to get the global candidate dense unit population
        /* Now divide the task of finding dense units between processors */
        Pick the appropriate $\frac{1}{p}^{th}$ portion of the populated candidate dense units to check if it qualifies to be a dense unit
        Reduce communication to get the global information of identified dense units
        Pick the appropriate $\frac{1}{p}^{th}$ of the dense units to find their bounds and build their data structures for use in the $(k+1)^{th}$ dimension
        Reduce communication to get the global information of the dense unit bounds
end

The SURFING algorithm

The summarized pseudocodes for the SURFING algorithm are as follows. It selects interesting features from the original attributes of the dataset.

```
SURFING(Database DB, Integer k)
S_1 := {{a_1}, ..., {a_d}};
compute quality of all subspaces S ∈ S_1;
S_l := S ∈ S_1 with lowest quality;
S_h := S ∈ S_1 with highest quality;
if quality(S_l) > 2/3 · quality(S_h) then
   τ := quality(S_h)/2;
else
   τ := quality(S_l);
   S_1 = S_1 − {S_l};
end if
k := 2;
create S_2 from S_1;
while not S_k = ∅ do
   compute quality of all subspaces S in S_k;
   Interesting := {S ∈ S_k | quality(S) ↑};
   Neutral := {S ∈ S_k | quality(s) ↓ ∧ quality(S) > τ};
   Irrelevant := {S ∈ S_k | quality(S) ≤ τ};
   S_l := S ∈ S_k with lowest quality;
   S_h := S ∈ S_k − Interesting with highest quality;
   if quality(S_l) > 2/3 · quality(S_h) then
      τ := quality(S_h)/2;
   else
      τ := quality(s_l);
   end if
   if not all subspaces irrelevant then
      S_k := S_k − Irrelevant;
   end if
   create S_{k+1} from S_k;
   k := k + 1;
end while
end
```

The R implementation

Please take a look at the R codes file `ch_06_surfing.R` from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_06_surfing.R")
```

Customer purchase data analysis

Customer purchase data analysis contains many applications such as the customer-satisfaction analysis.

From the customer purchase data analysis, one application helps find the unwanted consumption or user's purchase behavior.

SNS and clustering graph and network data

The clustering for graph and network data has a wide application in modern life, such as social networking. However, more challenges crop up along with the needs. High computational cost, sophisticated graphs, and high dimensionality and sparsity are the major concerns. With some special transformations, the issues can be transformed into graph cut issues.

Structural Clustering Algorithm for Network (**SCAN**) is one of the algorithms that searches for well-connected components in the graph as clusters.

Fig. 3: An example of overlaping communities [26]

The SCAN algorithm

The summarized pseudocodes for the SCAN algorithm are as follows:

```
SCAN for clusters on graph data.
Input: a graph G = ( V, E ), a similarity threshold ε, and a
  population threshold μ
Output: a set of clusters
Method: set all vertices in V unlabeled
  for all unlabeled vertex u do
     if u is a core then
       generate a new cluster-id c
       insert all v∈N_ε(u) into a queue Q
     while Q ≠ do
       w ← the first vertices in Q
       R ← the set of vertices that can be directly reached from w
     for all s∈R do
          if s is not unlabeled or labeld as nonmember then
               assign the current cluster-id c to s
          endif
          if s is not unlabeled or labeled as nonmember then
              assign the current cluster-id c to s
          endif
     endfor
     remove w from Q
  end while

  else
     label u as nonmember
  endif
  endfor
  for all vertex n labeled nonmember do
      if ∃ x,y ∈ Γ(u) : x and y have different cluster-ids then
           label u as hub
  else
     label u as outlier
  endif
  endfor
```

The R implementation

Please take a look at the R codes file `ch_06_scan.R` from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_06_scan.R")
```

Social networking service (SNS)

Social network has become the most popular online communication method nowadays. The analysis of SNS becomes important, because of the requirement of security, business, control, and so on.

The foundation of SNS is a graph theory, especially for SNS mining, such as finding social communities and abuse of SNS for bad purpose.

The clustering for SNS is an inherent application to find a community (or community detection). Random walk is another key technology for SNS analysis and is used to find communities.

Time for action

Here are some questions for you to know whether you have understood the concepts:

- What is the DBSCAN algorithm?
- What is the SCAN algorithm?
- What is the STING algorithm?
- What is the OPTICS algorithm?
- What is the constraint-based cluster method?

Summary

In this chapter, we covered the following facts:

- DBSCAN depends on the density-based description of clusters. With searching and measuring the density of data points, high density means high possibility of the existence of clusters, and others mean outliers or noise.
- OPTICS produces the cluster ordering that consists of the order of the data objects together with the corresponding reachability values and core values.
- You learned that DENCLUE is a clustering algorithm based on a set of specific density-distribution functions and can find arbitrary shape clusters. It first segments the dataset as cubes and identifies the local density-distribution function. You also learned that a hill-climbing algorithm is performed to retrieve the local maximum for each item in the related cube with which a cluster will be built.
- We saw that STING is based on the grid-like data structure, and it segments the embedding spatial area of the input data points into rectangular units. It is mainly used for a spatial dataset.

- You learned that CLIQUE is a grid-based clustering algorithm that finds subspaces of high-dimensional data, and it can find dense units in the high-dimensional data too.
- You know that the WAVE cluster is a grid-based clustering algorithm based on the wavelet transformations. It has a multiresolution and is efficient for large dataset.
- You learned that EM is a probabilistic-model-based clustering algorithm, where each data point with a probability indicates that it belongs to a cluster. It is based on the assumption that the attribute of the data point has values that is the linear combination of simple distributions.
- Clustering high-dimensional data.
- Clustering graph and network data.

In the next chapter, we will cover the major topics related to outlier detection and algorithms, and look at some of their examples.

7
Outlier Detection

In this chapter, you will learn how to write R codes to detect outliers in real-world cases. Generally speaking, outliers arise for various reasons, such as the dataset being compromised with data from different classes and data measurement system errors.

As per their characteristics, outliers differ dramatically from the usual data in the original dataset. Versatile solutions are developed to detect them, which include model-based methods, proximity-based methods, density-based methods, and so on.

In this chapter, we will cover the following topics:

- Credit card fraud detection and statistical methods
- Activity monitoring—the detection of fraud of mobile phones and proximity-based methods
- Intrusion detection and density-based methods
- Intrusion detection and clustering-based methods
- Monitoring the performance of network-based and classification-based methods
- Detecting novelty in text, topic detection, and mining contextual outliers
- Collective outliers on spatial data
- Outlier detection in high-dimensional data

Here is a diagram illustrating a classification of outlier detection methods:

The output of an outlier detection system can be categorized into two groups: one is the **labeled result** and the other is the **scored result** (or an ordered list).

Credit card fraud detection and statistical methods

One major solution to detect outliers is the model-based method or statistical method. The outlier is defined as the object not belonging to the model that is used to represent the original dataset. In other words, that model does not generate the outlier.

Among the accurate models to be adopted for the specific dataset, there are many choices available such as Gaussian and Poisson. If the wrong model is used to detect outliers, the normal data point may wrongly be recognized as an outlier. In addition to applying the single distribution model, the mixture of distribution models is practical too.

The log-likelihood function is adopted to find the estimation of parameters of a model:

$$\text{In}\,\mathcal{L}(\mu,\sigma^2)=\sum_{i=1}^{n}\text{In}\ f(x_i\,|(\mu,\sigma^2))=-\frac{n}{2}\text{In}\ (2\pi)-\frac{n}{2}\text{In}\sigma^2-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(\text{x}_i-\mu)^2$$

$$\hat{\mu}=\overline{x}=\frac{1}{n}\sum_{i=1}^{n}\text{x}_i$$

$$\hat{\sigma}^2-\frac{1}{n}\sum_{i=1}^{n}\left(x_i-\overline{x}\right)^2$$

The likelihood-based outlier detection algorithm

The summarized pseudocode of the likelihood-based outlier detection algorithm is as follows:

```
LBOutlierDetection (D, M, A, c) {
  At time t = 0, let M_t contains all the data objects, while A_t is empty set
  LL_t(D) ← LL(M_t) + LL(A_t)
  foreach (x ∈ M_t) {
    M_{t+1} ← M_t − {x} , A_{t+1} ← A_t ∪ {x}
    LL_{t+1}(D) ← LL(M_{t+1}) + LL(A_{t+1})
    Δ= LL_t(D) − LL_{t+1}(D)
    if(Δ> c){
      x is an outlier, M, A keeps unchanged.
    }
  }
}
```

The R implementation

Look up the file of R codes, `ch_07_lboutlier_detection.R`, from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_07_lboutlier_detection.R")
```

Credit card fraud detection

Fraud denotes the criminal activities that happen in various commercial companies, such as credit card, banking, or insurance companies. For credit card fraud detection, two major applications are covered, fraudulent application of credit card and fraudulent usage of credit card. The fraud represents behavior anomalous to the average usage of credit cards to certain users, that is, transaction records of the users.

This kind of outlier statistically denotes credit card theft, which deviates from the normal nature of criminal activities. Some examples of outliers in this case are high rate of purchase, very high payments, and so on.

The location of payment, the user, and the context are possible attributes in the dataset. The clustering algorithms are the possible solutions.

Activity monitoring – the detection of fraud involving mobile phones and proximity-based methods

Two major approaches of proximity-based methods are distance-based and density-based outlier detection algorithms.

The NL algorithm

The summarized pseudocodes of the NL algorithm are as follows:

Algorithm NL

1. Fill the first array (of size $\frac{B}{2}\%$ of the dataset) with a block of tuples from T.
2. For each tuple t_i in the first array, do:
   a. $count_i \leftarrow 0$
   b. For each tuple t_j in the first array, if dist(t_i, t_j) $\leq D$:

      Increment $count_i$ by 1. If $count_i > M$, mark t_i as a non-outlier and proceed to next t_i.
3. While blocks remain to be compared to the first array, do:
   a. Fill the second array with another block (but save a block which has never served as the first array, for last).
   b. For each unmarked tuple t_i in the first array do:

      For each tuple t_j in the second array, if dist(t_i, t_j) $\leq D$:

      Increment $count_i$ by 1. If $count_i > M$, mark t_i as a non-outlier and proceed to next t_i.
4. For each unmarked tuple t_i in the first array, report t_i as an outlier.
5. If the second array has served as the first array anytime before, stop; otherwise, swap the names of the first and second arrays and goto step 2.

The FindAllOutsM algorithm

The following are the summarized pseudocode of the FindAllOutsM algorithm:

Algorithm FindAllOutsM

1. For $q \leftarrow 1, 2, \ldots m, \quad Count_q \leftarrow 0$
2. For each object P, map P to an appropriate cell C_q, store P, and increment $Count_q$ by 1.
3. For $q \leftarrow 1, 2, \ldots, m, \quad$ if $Count_q > M$, label C_q *red*.
4. For each *red* cell C_r, label each of the L_1 neighbours of C_r *pink*, provided the neighbour has not already been labelled *red*.
5. For each non-empty *white* (i.e., uncoloured) cell C_w, do:
   a. $Count_{w2} \leftarrow Count_w + \sum_{i \in L_1(C_w)} Count_i$
   b. If $Count_{w2} > M$, label C_w *pink*.
   c. else
      1. $Count_{w3} \leftarrow Count_{w2} + \sum_{i \in L_2(C_w)} Count_i$
      2. If $Count_{w3} \leq M$, mark all objects in C_w as outliers.
      3. else for each object $P \in C_w$, do:
         i. $Count_P \leftarrow Count_{w2}$
         ii. For each object $Q \in L_2(C_w)$, if $dist(P, Q) \leq D$:
            Increment $Count_P$ by 1. If $Count_P > M$, P cannot be an outlier, so goto 5(c)(3).
         iii. Mark P as an outlier.

The FindAllOutsD algorithm

The summarized pseudocodes of the FindAllOutsD algorithm are as follows:

Algorithm FindAllOutsD

1. For $q \leftarrow 1, 2, \ldots m, \quad Count_q \leftarrow 0$
2. For each object P in the dataset, do:
   a. Map P to its appropriate cell C_q but do not store P.
   b. Increment $Count_q$ by 1.
   c. Note that C_q references P's page.
3. For $q \leftarrow 1, 2, \ldots, m, \quad$ if $Count_q > M$, label C_q *red*.
4. For each *red* cell C_r, label each of the L_1 neighbours of C_r *pink*, provided the neighbour has not already been labelled *red*.
5. For each non-empty *white* (i.e., uncoloured) cell C_w, do:
   a. $Count_{w2} \leftarrow Count_w + \sum_{i \in L_1(C_w)} Count_i$
   b. If $Count_{w2} > M$, label C_w *pink*.
   c. else
      1. $Count_{w3} \leftarrow Count_{w2} + \sum_{i \in L_2(C_w)} Count_i$
      2. If $Count_{w3} \leq M$, label C_w *yellow* to indicate that all tuples mapping to C_w are outliers.
      3. else $Sum_w \leftarrow Count_{w2}$
6. For each page i containing at least 1 *white* or *yellow* tuple, do:
   a. Read page i.
   b. For each *white* or *yellow* cell C_q having tuples in page i, do:
      1. For each object P in page i mapped to C_q, do:
         i. Store P in C_q.
         ii. $Kount_P \leftarrow Sum_q$
7. For each object P in each non-empty *white* cell C_w, do:
   a. For each *white* or *yellow* cell $C_L \in L_2(C_w)$, do:
      1. For each object $Q \in C_L$, if $dist(P, Q) \leq D$:
         Increment $Kount_P$ by 1. If $Kount_P > M$, mark P as a non-outlier, and goto next P.
8. For each object Q in each *yellow* cell, report Q as an outlier.
9. For each page i containing at least 1 tuple that (i) is both *non-white* and *non-yellow*, and (ii) maps to an L_2 neighbour of some white cell C, do:
   a. Read page i.
   b. For each cell $C_q \in L_2(C)$ that is both *non-white* and *non-yellow*, and has tuples in page i, do:
      1. For each object Q in page i mapped to C_q, do:
         i. For each non-empty *white* cell $C_w \in L_2(C_q)$, do:
            For each object $P \in C_w$, if $dist(P, Q) \leq D$:
            Increment $Kount_P$ by 1. If $Kount_P > M$, mark P as a non-outlier.
10. For each object P in each non-empty *white* cell, if P has not been marked as a non-outlier, then report P as an outlier.

The distance-based algorithm

The summarized pseudocodes of the distance-based outlier detection algorithm are as follows, given a dataset *D*, size of the input dataset *n*, threshold *r* (`r > 0`), and $\pi \in (0,1]$:

```
for i in n
    count = 0
    for j in n
        if i ≠ j and dist (o_i, o_j) ≤ r
            count++
            if count ≥ π {
                exit (o_i cannot be a DB (r,π) outlier)
            }
        }
    }
    print o_i, which is determined as a DB (r,π) outlier
}
```

A $DB(r,\pi)$ outlier is defined as a data point, `o`, and subjected to this formula:

$$\left\|\{o' \mid dist(o,o') \le r\}\right\| / \left\|D\right\| \le \pi$$

Let's now learn the pseudocodes for a variety of distance-based outlier detection algorithms, which are summarized in the following list. The input parameters are `k`, `n`, and `D`, which represent the neighbors' number, outlier number to be identified, and input dataset, respectively. A few supporter functions also are defined. `Nearest (o, S, k)` returns `k` nearest objects in `S` to `o`, `Maxdist (o, S)` returns the maximum distance between `o` and points from `S`, and `TopOutlier (S, n)` returns the top *n* outliers in `S` according to the distance to their *k*th nearest neighbor.

```
O ← ∅
D^k_min ← 0
for each object o in D do
    Neighbours(o) ← ∅

    D^k(o) ← 0

    for each object v in D, where v ≠ o do
        Neighbours(o) = Nearest(o, Neighbours(o)∪ v, k)
        D^k(o) = Maxdist(o, Neighbours(o))
        if |Neighbours(o)| = k and D^k_min > D^k(o) then
            break
        end if
    end for
    O = TopOutliers(O ∪ o, n)
    if | O | = n then
        D^k_min = min(D^k(o) for all o in O)
    end if
end for
```

The Dolphin algorithm

The Dolphin algorithm is a distance-based outlier detection algorithm. The summarized pseudocodes of this algorithm are listed as follows:

```
Algorithm DOLPHIN
  initialize an empty DBO-index INDEX

  for each object obj of DS do
    associate a DBO-node n_curr with the object obj
    if not isInlier(n_curr) then
      insert n_curr into INDEX
  remove from INDEX all the nodes n such that n.rad ≤ R
  set to zero all the entries of the n.nn arrays of the nodes n in INDEX

  for each object obj of DS do
    execute the procedure pruneInliers(obj)
  the objects remaining in INDEX are the outliers of DS
```

Function isInlier(n_{curr})

perform a range query search in INDEX with center $n_{curr}.obj$ and radius R
for each node n_{index} returned by the range query **do**
  $dst = \text{dist}(n_{curr}.obj, n_{index}.obj)$
  if $dst \leq R - n_{index}.rad$ **then**
    stop the search and report $n_{curr}.obj$ as an inlier [PR1] (return TRUE)
  if $dst \leq R$ **then**
    $oldrad = n_{index}.rad$
    update the array $n_{index}.nn$ with the distance dst
    if $oldrad > R$ and $n_{index}.rad \leq R$ **then**
      remove, with probability $1 - p_{inliers}$, the node n_{index} from INDEX [PR2]
    update the array $n_{curr}.nn$ with the distance dst
    if $n_{curr}.rad \leq R$ **then**
      stop the search and report $n_{curr}.obj$ as an inlier [PR3] (return TRUE)
report $n_{curr}.obj$ as not an inlier (return FALSE)

Procedure pruneInliers(obj)

perform a range query search in INDEX with center obj and radius R
for each node n_{index} returned by the range query **do**
  if $\text{dist}(obj, n_{index}.obj) \leq R$ **then**
    update the array $n_{index}.nn$ with obj
    if $n_{index}.rad \leq R$ **then**
      delete n_{index} from INDEX

The R implementation

Look up the file of R codes, `ch_07_proximity_based.R`, from the bundle of R codes for the preceding algorithms. The codes can be tested with the following command:

```
> source("ch_07_proximity_based.R")
```

Activity monitoring and the detection of mobile fraud

The purpose of outlier detection is to find the patterns in source datasets that do not conform to the standard behavior. The dataset here consists of the calling records, and the patterns exist in the calling records.

There are many special algorithms developed for each specific domain. Misuse of a mobile is termed as mobile fraud. The subject under research is the calling activity or call records. The related attributes include, but are not limited to, call duration, calling city, call day, and various services' ratios.

Intrusion detection and density-based methods

Here is a formal definition of outliers formalized based on concepts such as, LOF, LRD, and so on. Generally speaking, an outlier is a data point biased from others so much that it seems as if it has not been generated from the same distribution functions as others have been.

Given a dataset, `D`, a DB (x, y)-outlier, `p`, is defined like this:

$$\left|\{q \in D | d(p,q) \le y\}\right| \le x$$

The k-distance of the `p` data point denotes the distance between `p` and the data point, `o`, which is member of `D`:

$$dist_k(p) \, denotes \, k - distance \, of \, data \, point \, p$$

$$\left|\{o' \in D \setminus \{p\} | d(p,o') \le d(p,o)\}\right| \ge k$$

$$\left|\{o' \in D \setminus \{p\} | d(p,o') < d(p,o)\}\right| \le k-1$$

The k-distance neighborhood of the `p` object is defined as follows, `q` being the k-Nearest Neighbor of `p`:

$$N(p) = \{q \in D \setminus \{p\} | d(p,q) \le dist_k(p)\}$$

The following formula gives the reachability distance of an object, `p`, with respect to an object, `o`:

$$reachdist_k(p,o) = \max\{dist_k(o), d(p,o)\}$$

The **Local Reachability Density** (**LRD**) of a data object, `o`, is defined like this:

$$lrd_{MinPts}(p) = 1 / \left(\frac{\sum_{o \in N_{MinPts}(\mathrm{p})} reach\text{-}dist_{MinPts}(p,o)}{\left| N_{MinPts}(p) \right|} \right)$$

The **Local Outlier Factor** (**LOF**) is defined as follows, and it measures the degree of the outlierness:

$$LOF_{MinPts}(p) = \frac{\sum_{o \in N_{MinPts}(\mathrm{p})} \frac{lrd_{MinPts}(o)}{lrd_{MinPts}(p)}}{\left| N_{MinPts}(p) \right|}$$

A property of `LOF (p)` is defined as shown in the following equation:

$$\frac{direct_{min}(p)}{indirect_{max}(p)} \le LOF(p) \le \frac{direct_{max}(p)}{indirect_{min}(p)}$$

$$direct_{min}(p) = \min\{reachdist(p,q) \mid q \in N_{MinPts}(p)\}$$

$$indirect_{min}(p) = \min\{reachdist(q,o) \mid q \in N_{MinPts}(p), o \in N_{MinPts}(q)\}$$

These equations are illustrated as follows:

The OPTICS-OF algorithm

The input parameters for the bagging algorithm are:

- $D(A_1, \ldots, A_m)$, the dataset
- α, the parameter
- β, another parameter

The output of the algorithm is the value of CBLOF, for all records.

The summarized pseudocodes of the OPTICS-OF algorithm are as follows:

$1: OPTICS-OF\ (D, K, S)\ \{$

$2:\quad clustering\ the\ dataset\ D(A_1, ..., A_m)\ by\ squeezer\ algorithm;$

$the\ prodused\ clusters\ are\ C = \{C_1, ..., C_\kappa\}, \text{and} |C_1| \geq ... \geq |C_k|$

$3:\quad get\ LC\ and\ SC\ with\ the\ two\ parameters;$

$4:\quad for(each\ record\ t\ in\ the\ dataset\ D)\{$

$5:\quad\quad if\ (\mathrm{t} \in C_i\ and\ C_i \in SC)\{$

$6:\quad\quad\quad \mathrm{CBLOF} = |C_i| * \min\left(\mathrm{distance}(t, C_i)\right), C_j \in LC;$

$7:\quad\quad \}\ \mathrm{else}\ \{$

$8:\quad\quad\quad \mathrm{CBLOF} = |C_i| * \mathrm{distance}(\mathrm{t}, \mathrm{C}_i), C_i \in LC;;$

$9:\quad\quad \}$

$10:\quad\quad return\ CBLOF$

$11:\quad \}$

$12: \}$

The High Contrast Subspace algorithm

The summarized pseudocodes of the **High Contrast Subspace** (**HiCS**) algorithm are as follows, where the input parameters are `S`, `M`, and $\propto$. The output is a contrast, `|S|`.

calculation of subspace contrast

for $i = 1 \rightarrow M$ **do**
  Permute list of subspace attributes $s \in S$
  Initialize boolean vector *selected_objects* for all objects: *true*
  for $i = 1 \rightarrow |S| - 1$ **do**
    Select random index block of attribute s_i with a size of $N \cdot \sqrt[|S|]{\alpha}$
    Mask index block with *selected_objects*
  end for
  Compare distributions: $deviation\left(\hat{p}_{s_i},\ \hat{p}_{s_i|selected_objects}\right)$ for the remaining attribute with $i = |S|$.
end for
Combine the results of all statistical test

The R implementation

Look up the file of R codes, `ch_07_ density _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested using the following command:

```
> source("ch_07_ density _based.R")
```

Intrusion detection

Any malicious activity against systems, networks, and servers can be treated as intrusion, and finding such activities is called **intrusion detection**.

The characteristics of situations where you can detect intrusion are high volume of data, missing labeled data in the dataset (which can be training data for some specific solution), time series data, and false alarm rate in the input dataset.

An intrusion detection system is of two types: host-based and network-based intrusion detection systems. A popular architecture for intrusion detection based on data mining is illustrated in the following diagram:

The core algorithms applied in an outlier detection system are usually semi-supervised or unsupervised according to the characteristics of intrusion detection.

Intrusion detection and clustering-based methods

The strategy of outlier detection technologies based on the clustering algorithm is focused on the relation between data objects and clusters.

Hierarchical clustering to detect outliers

Outlier detection that uses the hierarchical clustering algorithm is based on the k-Nearest Neighbor graph. The input parameters include the input dataset, `DATA`, of size, `n`, and each data point with `k` variables, the distance measure function (`d`), one hierarchical algorithm (`h`), threshold (`t`), and cluster number (`nc`).

$Out \leftarrow \phi$
Obtain the distance matrix D by applying the distance function d
  to the observations in $DATA$
Use algorithm h to grow an hierarchy using the distance matrix D
Cut the hierarchy at the level l that leads to nc clusters
FOR each resulting cluster c DO
    IF sizeof(c) $< t$ THEN
        $Out \leftarrow Out \cup \{obs \in c\}$

The k-means-based algorithm

The process of the outlier detection based on the k-means algorithm is illustrated in the following diagram:

The summarized pseudocodes of outlier detection using the k-means algorithm are as follows:

- **Phase 1** (**Data preparation**):
    1. The target observations and attributes should be aligned to improve the accuracy of the result and performance of the k-means clustering algorithm.
    2. If the original dataset has missing data, its handling activity must be carried out. The data of maximum likelihood that is anticipated by the EM algorithm is fed as input into the missing data.
- **Phase 2** (**The outlier detection procedure**):
    1. The *k* value should be determined in order to run the k-means clustering algorithm. The proper *k* value is decided by referring to the value of the **Cubic Clustering Criterion**.
    2. The k-means clustering algorithm runs with the decided *k* value. On completion, the expert checks the external and internal outliers in the clustering results. If the other groups' elimination of the outliers is more meaningful, then he/she stops this procedure. If the other groups need to be recalculated, he/she again runs the k-means clustering algorithms but without the detected outliers.
- **Phase 3** (**Review and validation**):
    1. The result of the previous phase is only a candidate for this phase. By considering the domain knowledge, we can find the true outliers.

The ODIN algorithm

Outlier detection using the indegree number with the ODIN algorithm is based on the k-Nearest Neighbor graph.

```
ODIN
T is indegree threshold
Calculate kNN graph of S
for i= 1 to |S| do
  if indegree of v_i ≤ T then
    Mark v_i as outlier
  end if
end for
```

The R implementation

Look up the file of R codes, `ch_07_ clustering _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ clustering _based.R")
```

Monitoring the performance of the web server and classification-based methods

Classification algorithms can be used to detect outliers. The ordinary strategy is to train a one-class model only for the normal data point in the training dataset. Once you set up the model, any data point that is not accepted by the model is marked as an outlier.

The OCSVM algorithm

The **OCSVM** (**One Class SVM**) algorithm projects input data into a high-dimensional feature space. Along with this process, it iteratively finds the maximum-margin hyperplane. The **hyperplane** defined in a **Gaussian reproducing kernel Hilbert** space best separates the training data from the origin. When $v \in [0,1]$, the major portion of outliers or the solution of OCSVM can be represented by the solution of the following equation (subject to $\sum_{k=1}^{n} w_k = 1$ and $0 \le w_1, ..., w_n \le \frac{1}{vn}$):

$$\min_{\{w_k\}_{k=1}^n} \frac{1}{2} \sum_{k,k'=1}^{n} w_k w_{k'} K_\sigma(\boldsymbol{x}_k, \boldsymbol{x}_{k'})$$

Initialization of the algorithm. We start by setting a random fraction ν of all α_i to $1/(\nu\ell)$. If $\nu\ell$ is not an integer, then one of the examples is set to a value in $(0, 1/(\nu\ell))$ to ensure that $\sum_i \alpha_i = 1$. Moreover, we set the initial ρ to $\max\{O_i : i \in [\ell], \alpha_i > 0\}$.

Optimization algorithm. We then select a first variable for the elementry optimization step in one of the two following ways. Here, we use the shorthand SV_{nb} for the indices of variables which are not at bound, that is, $SV_{nb} := \{i : i \in [\ell], 0 < \alpha_i < 1/(\nu\ell)\}$. At the end, these correspond to points that will sit exactly on the hyperplane, and that will therefore have a strong influence on its precise position.

(i) We scan over the entire data set[1] until we find a variable violating a KKT condition(Bertsekas, 1995, e.g.), that is, a point such that $(O_i - \rho).\alpha_i > 0$ or $(\rho - O_i).(1/(\nu\ell)) - \alpha_i) > 0$. Once we have found one,say α_i, we pick α_j according to

$$j = \arg\max_{n \in} sv_{nb} \left| O_i - O_n \right|.$$

(ii) Same as (i), but the scan is only performed over sv_{nb}

The one-class nearest neighbor algorithm

This algorithm is based on the k-Nearest Neighbor algorithm. A couple of formulas are added.

The local density is denoted as follows:

$$P_n(x) = k_n / N / V_n$$

The distance between the test object, x, and its nearest neighbor in the training set, $NN^{tr}(x)$, is defined like this:

$$d_1 = \left\| x - NN^{tr}(x) \right\|$$

The distance between this nearest neighbor ($NN^{tr}(x)$) and its nearest neighbor in the training set ($NN^{tr}\left(NN^{tr}(x)\right)$) is defined as follows:

$$d_2 = \left\| NN^{tr}(x) - NN^{tr}\left(NN^{tr}(x)\right) \right\|$$

One data object is marked as an outlier once $d_1 >> d_2$, or, in another format, is marked as $\rho_{NN}(x) = d_1 / d_2$.

The R implementation

Look up the file of R codes, `ch_07_ classification _based.R`, from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ classification _based.R")
```

Monitoring the performance of the web server

Web server performance measurements are really important to the business and for operating system management. These measurements can be in the form of CPU usage, network bandwidth, storage, and so on.

The dataset comes from various sources such as benchmark data, logs, and so on. The types of outliers that appear during the monitoring of the web server are point outliers, contextual outliers, and collective outliers.

Detecting novelty in text, topic detection, and mining contextual outliers

If each of the data instances in the training dataset is related to a specific context attribute, then the dramatic deviation of the data point from the context will be termed as an outlier. There are many applications of this assumption.

The conditional anomaly detection (CAD) algorithm

The summarized pseudocodes of the CAD algorithm are as follows:

1: $Direct - CAD\ (D, K, S)\ \{$

2: $choose\ an\ initial\ set\ of\ values\ for < \mu_{V_j}, \sum V_j >, < \mu_{U_i}, \sum U_i >$
$, P(V_j|U_i), P(U_i), for\ all\ i, j;$

3: $while\ (the\ model\ continues\ improving, which\ mearured\ by\ improvement\ of\ \Lambda)\ \{$

4: $compute\ b_{kij}\ for\ all\ k, i\ and\ j\ ;$

5: $compute < \overline{\mu_{V_j}}, \overline{\sum V_j} >, < \overline{\mu_{U_i}}, \overline{\sum U_i} >, \overline{P(V_j|U_i)}, \overline{P(U_i)}\ for\ all\ i, j;$

6: $set < \mu_{V_j}, \sum V_j > = < \overline{\mu_{V_j}}, \overline{\sum V_j} >,\ \ < \mu_{U_i}, \sum U_i > = \ < \overline{\mu_{U_i}}, \overline{\sum U_i} >,\ \ P(V_j|U_i) = \overline{P\ (V_j|U_i)},$
$P(U_i)\ =\ P(U_i)$

7: $return\ CBLOF;$

8: $\}$

9: $\}$

The following are the summarized pseudocodes of the GMM-CAD-Full algorithm:

1: $GMM - CAD - Full\ ()\ \{$

2: $learn\ a\ set\ of\ n_U\ (d_U + d_V) - dimensional\ Gaussians\ over\ the\ dataset$
$\{(x_1, y_1), \ldots, (x_n, y_n)\}, the\ result\ set\ call\ as\ z$

3: $/*\ step\ 2\ ,\qquad determine\ U\ */$

4: $let\ \mu_{Z_i}\ refer\ the\ centroid\ of\ the\ i^{th}\ Gaussian\ in\ Z$

5: $let\ \sum Z_i\ refer\ to\ the\ covariance\ matrix\ of\ the\ i^{th}\ Gaussian\ in\ Z$

6: $for(i = 1\ to\ n_U)\ \{$

7: $P(U_I) = P(Z_I)$

```
    for(j = 1 to d_U) {
        μ_{V_i}[j] = μ_{Z_i}[j]
        for(k = 1 to d_U) {
            Σ V_i[j][k] = Σ Z_i[j][k]
        }
    }
}

/* step 3 ,    determine V */
for(i = 1 to n_v) {
    for(j = 1 to d_v) {
        μ_{V_i}[j] = μ_{Z_i}[j + d_U]
        for(k = 1 to d_v) {
            Σ V_i[j][k] = Σ Z_i[j + d_U][k + d_U]
        }
    }
}

/* step 4 */
run a second EM algorithm to learn the mapping function.
while (the model continues improving, which mearured by improvement of Λ) {
    compute b_kij for all k, i and j as in the previous sub section;
    compute P̄(V_j|U_i) as described in the previous subsection;
    set P(V_j|U_i) = P̄(V_j|U_i)
}
```

The summarized pseudocodes of the GMM-CAD-Split algorithm are as follows:

```
GMM-CAD-Split (){
    Learn U and V by performing two separate EM optimizations
    Run step (4) from the GMM-CAT-Full algorithm to learn the mapping function
}
```

The R implementation

Look up the file of R codes, `ch_07_ contextual _based.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ contextual _based.R")
```

Detecting novelty in text and topic detection

One application of outlier detection is in finding novel topics in a collection of documents or articles from newspapers. Major detection includes opinion detection. This is basically an outlier among a lot of opinions.

With the increase in social media, there are many events that happen every day. The earlier collection was that of only special events or ideas for related researchers or companies.

The characteristics related to increase in the collection are the various sources of data, documents in different formats, high-dimensional attributes, and sparse source data.

Collective outliers on spatial data

Given a dataset, if a collection of related data instances is anomalous with respect to the entire dataset, it is defined as a collective outlier.

The route outlier detection (ROD) algorithm

The summarized pseudocodes of the ROD algorithm are as follows. The input parameters include the multidimensional attribute space, attribute dataset (`D`), distance measure function (`F`), the depth of neighbor (`ND`), spatial graph (`G = (V, E)`), and confidence interval (`CI`):

```
for (i=1);i ≤ |RN| ; i++){
    O_i = Get_One_Object(i,D);
    NNS=Find_Neighbor_Nodes_set(O_i, ND, G)
    Accum_Dist=0;
    for(j=1; j ≤ |NSS|; j++){
        O_k = Get_One_Object(j.NNS);
        Accum_Dist += F(O_i, O_k, S)
    }
    AvgDist = Accum_Dist/ |NSS|;
    T_value = (AvgDist − μ_s) / σ_s
    if(Check_Normal_Table(T_value, CI) == True){
        Add_Element(Outlier_Set, i);
    }
}
return Outlier_Set
```

The R implementation

Look up the file of R codes, `ch_07_ rod.R`, from the bundle of R codes for the previously mentioned algorithm. The codes can be tested with the following command:

```
> source("ch_07_ rod.R")
```

Characteristics of collective outliers

Collective outliers denote a collection of data that is an abnormal contrast to the input dataset. As a major characteristic, only the collection of data appearing together will be collective outliers, but specific data itself in that collection does not appear together with other data in that collection of data, which is definitely not an outlier. Another characteristic of a collective outlier is that it can be a contextual outlier.

Collective outliers may be a sequence of data, spatial data, and so on.

Outlier detection in high-dimensional data

Outlier detection in high-dimensional data has some characteristics that make it different from other outlier detection problems.

The brute-force algorithm

The summarized pseudocodes of the brute-force algorithm are as follows:

Algorithm *BruteForce(Number: m, Dimensionality: k)*

$R_1 = Q_1 =$ Set of all $d \cdot \phi$ ranges;
for $i = 2$ **to** k **do**
  begin
  $R_i = R_{i-1} \oplus Q_1$;
  end;
Determine sparsity coefficients of all
  elements in R_k;
$\mathcal{F} =$ Set of m elements in R_k
    with most negative sparsity coefficients;
$\mathcal{O} =$ Set of points covered by $\mathcal{F}$;
return($\mathcal{F}$, $\mathcal{O}$);

The HilOut algorithm

The following are the summarized pseudocodes of the HilOut algorithm:

```
HilOut(DB, n, k, h) {
  Initialize(PF, DB)
  /* first phase */
  TOP ← Φ
  N* ← N; n* ← 0, ω* ← 0
  j ← 0

  while (j≤d && n* < n){
    Initialize(OUT)
    Initialize(WLB)
      Hilbert(v^(j))
      Scan(v^(j), kN/N*)
      TrueOutliers(OUT)

      TOP ← OUT ∪ WLB
     j ←j+1
  }
  /* second phase */
  if(n* < n)
      Scan(v^(d), N)
  return OUT;

}
```

The R implementation

Look up the R file, `hil_out.R`, from the bundle of R codes for the HilOut algorithm.

Look up the file of R codes, `ch_07_ hilout.R`, from the bundle of R codes for the previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_07_ hilout.R")
```

Time for action

Here are some practice questions for you so as to check your understanding of the concepts:

- What is an outlier?
- List as many types of outliers as possible and your categorization measures.
- List as many areas of application of outlier detection as possible.

Summary

In this chapter, we looked at:

- Statistical methods based on probabilistic distribution functions. The normal data points are those that are generated by the models. Otherwise, they are defined as outliers.
- Proximity-based methods.
- Density-based methods.
- Clustering-based methods.
- Classification-based methods.
- Mining contextual outliers.
- Collective outliers.
- Outlier detection in high-dimensional data.

The next chapter will cover the major topics related to outlier detection algorithms and examples for them, which are based on the previous chapters. All of this will be covered with a major difference in our viewpoint.

8

Mining Stream, Time-series, and Sequence Data

In this chapter, you will learn how to write mining codes for stream data, time-series data, and sequence data.

The characteristics of stream, time-series, and sequence data are unique, that is, large and endless. It is too large to get an exact result; this means an approximate result will be achieved. The classic data-mining algorithm should be extended, or a new algorithm needs to be designed for this type of the dataset.

In relation to the mining of stream, time-series, and sequence data, there are some topics we can't avoid. They are association, frequent pattern, classification and clustering algorithms, and so on. In the following sections, we will go through these major topics.

In this chapter, we will cover the following topics;

- The credit card transaction flow and STREAM algorithm
- Predicting future prices and time-series analysis
- Stock market data and time-series clustering and classification
- Web click streams and mining symbolic sequences
- Mining sequence patterns in transactional databases

The credit card transaction flow and STREAM algorithm

As we mentioned in the previous chapters, one kind of data source always requires a variety of predefined algorithms or a brand new algorithm to deal with. Streaming data behaves a bit different from a traditional dataset.

The streaming dataset comes from various sources in modern life, such as credit record transaction stream, web feeds, phone-call records, sensor data from a satellite or radar, network traffic data, a security event's stream, and a long running list of various data streams.

The targets to stream data processing are, and not limited to, summarization of the stream to specific extents.

With the characteristics of streaming data, the typical architecture to stream a management system is illustrated in the following diagram:

The STREAM algorithm is a classical algorithm used to cluster stream data. In the next section, the details are present and explained by R code.

The STREAM algorithm

The summarized pseudocode of the STREAM algorithm are as follows:

STREAM

For each chunk X_i in the stream

1. If a sample of size $\geq \frac{1}{\epsilon} \log \frac{k}{\delta}$ contains fewer than k distinct points then:
   - $X_i \longleftarrow$ weighted representation
2. Cluster X_i using LOCALSEARCH
3. $X' \longleftarrow ik$ centers obtained from chunks 1 through i iterations of the stream, where each center c obtained by clustering X_i is weighted by the number of points in X_i assigned to c.
4. Output the k centers obtained by clustering X_i using LOCALSEARCH

In the preceding algorithm, **LOCALSEARCH** is a revised k-median algorithm.

LOCALSEARCH (N, d ($\cdot$, $\cdot$), k, ϵ, ϵ',$\cdot\epsilon''$)

1. Read in n data points.
2. Set z_{min}=0
3. Set $z_{max}=\sum_{x\epsilon N} d(x, x_0)$, where x_0 is an arbitrary point in N
4. Set z to be $\frac{z_{max} + z_{min}}{2}$
5. Obtain an initial solution (I , a) using Algorithm InitialSolution (N , z).
6. Select Θ ($\frac{1}{p}$ $\log k$) random points to serve as feasible centers
7. While more or fewer that k centers and $z_{min} < (1-\epsilon'')$ z_{max}
   - Let (F, g) be the current solution
   - Run FL (N, d, ϵ, (F, g)) to obtain a new solution (F' , g')
   - If $|F'|$ is "about" k, run FL (N, d, ϵ, (F', g')) to obtain a new solution; reset (F', g') to be this new solution
   - If $|F'| < k$ then set z_{min} = z and z = $\frac{z_{max} + z_{min}}{2}$; else if $|F'| > k$ then set z_{max} = z and z = $\frac{z_{max} + z_{min}}{2}$
8. To Simulate a continuous space, move each cluster center to the center - of - mass for its cluster
9. Return our solution (F' , g')

InitialSolution (data set N, facility cost z)

1. Reorder data points randomly
2. Create a cluster center at the first point
3. For every point after the first,
   - Let d be the distance form the current data point to the nearest existing cluster center
   - With probability d / z create a new cluster center at the current data point; otherwise add the current point to the best current cluster

Algorithm FL(N, $d(\cdot,\cdot)$, z, ϵ, (I,a))

1. Begin with (I, a) as the current solution
2. Let C be the cost of the current solution on N. Consider the feasible centers in random order, and for each feasible center y, if $gain(y) > 0$, perform all advantageous closures and reassignments (as per *gain* description), to obtain a new solution (I', a') [a' should assign each point to its closest center in I']
3. Let C' be the cost of the new solution; if $C' \leq (1-\epsilon)C$, return to step 2

The single-pass-any-time clustering algorithm

Here is a clustering algorithm designed to deal with a high-frequency news stream:

Single pass anytime clustering algorithm.

Input:

Document vector stream $V = \{v_i | i=1,...,3\}$

Cluster candidate selection function $p(\cdot,\cdot)$.

Distance function $\delta(\cdot,\cdot)$ between vectors and clusters.

Distance threshold T.

Output:

Set of clusters $C = \{_{cj} \subset V \mid j=1,....,3\}$.

```
C := ∅
for all i = 1,...,3 do
    Ĉ := p(C,v_i) ⊆ C
    d̂ := min{δ(v_i, c)|c ∈ Ĉ}
    if d̂ < T then
        ĉ := c ∈ C δ(v_i)= d̂
        ĉ := ĉ ∪ {v_i}
    else
        C := C ∪ {{v_i}}
    end if
end for
```

The R implementation

Please take a look at the R codes file `ch_08_stream.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_08_stream.R")
```

The credit card transaction flow

Day by day, the growing e-commerce market is driving the growth of the usage of credit card, which, in turn, is bringing in a large number of transaction streams. Fraudulent usage of credit cards happens every day; we want the algorithm to detect this kind of transaction in a very short time compared to the big volume of transactions. Besides this, the requirement to find out the valuable customers by an analysis of the transaction streams is becoming more and more important. However, it is harder to get valid information in a very short time in response to the advertisement needs, such as recommend the necessary finance services or goods to these customers.

The credit card transaction flow is also the process to generate stream data, and the related stream-mining algorithm can be applied with high accuracy.

One application for the credit card transaction flow mining is the behavior analysis of the consumers. With each transaction record that is stored, various costs or purchases of a certain person can be tracked. With the mining of such transaction records, we can provide an analysis to the credit card holder to help them keep financial balance or other financial target. We can also provide this analysis to the card issuer that published the credit card to create a new business, such as funding or loaning, or to the retailers (the merchants) such as Walmart to help in the arrangement of appropriate goods.

Another application for the credit card transaction flow mining is fraud detection. It is the most obvious one among tons of applications. Using this solution, the card issuer or the bank can reduce the rate of successful fraud.

The dataset of the credit card transaction includes the owner of cards, place of consumption, date, cost, and so on.

Predicting future prices and time-series analysis

Auto Regressive Integrated Moving Average (**ARIMA**) is a classic algorithm to analyze time–series data. As the initial step, an ARIMA model will be chosen to model the time-series data. Assuming that p_t is a time-series variable such as a price, the formula is defined as follows, and it includes the main features of the variable:

$$\phi(B)\, pt = \theta(B)\, \varepsilon_t$$

$$\phi(B)\,pt = \left(1-\phi_1 B^1 - \phi_2 B^2\right)\left(1-\phi_{24}B^{24} - \phi_{48}B^{48}\right) \times\left(1-\phi_{168}B^{168}\right)(1-B)\left(1-B^{24}\right)$$

$$\begin{aligned}
&\left(1-\phi_1 B^1 - \phi_2 B^2 - \phi_3 B^3 - \phi_4 B^4 - \phi_5 B^5\right)\\
&\times\left(1-\phi_{23}B^{23} - \phi_{24}B^{24} - \phi_{47}B^{47} - \phi_{48}B^{48}\right.\\
&\left.-\phi_{72}B^{72} - \phi_{96}B^{96} - \phi_{120}B^{120} - \phi_{144}B^{144}\right)\\
&\times\left(1-\phi_{168}B^{168} - \phi_{336}B^{336} - \phi_{504}B^{504}\right)\log pt\\
&= c + \left(1-\theta_1 B^1 - \theta_2 B^2\right)\left(1-\theta_{24}B^{24}\right)\\
&\quad\times\left(1-\theta_{168}B^{168} - \theta_{336}B^{336} - \theta_{504}B^{504}\right)\varepsilon_t
\end{aligned}$$

$$\begin{aligned}
&\left(1-\phi_1 B^1 - \phi_2 B^2\right)\\
&\times\left(1-\phi_{23}B^{23} - \phi_{24}B^{24} - \phi_{47}B^{47} - \phi_{48}B^{48}\right.\\
&\quad\left.-\phi_{72}B^{72} - \phi_{96}B^{96} - \phi_{120}B^{120} - \phi_{144}B^{144}\right)\\
&\left(1-\phi_{167}B^{167} - \phi_{168}B^{168} - \phi_{169}B^{169} - \phi_{192}B^{192}\right)\\
&\times(1-B)\left(1-B^{24}\right)\left(1-B^{168}\right)\log pt\\
&\quad= c + \left(1-\theta_1 B^1 - \theta_2 B^2\right)\\
&\qquad\times\left(1-\theta_{24}B^{24} - \theta_{48}B^{48} - \theta_{72}B^{72} - \theta_{96}B^{96}\right)\\
&\qquad\times\left(1-\theta_{144}B^{144}\right)\\
&\qquad\times\left(1-\theta_{168}B^{168} - \theta_{336}B^{336} - \theta_{504}B^{504}\right)\varepsilon_t
\end{aligned}$$

The ARIMA algorithm

The summarized steps for the ARIMA algorithm is as follows:

1. A class of models is formulated assuming certain hypotheses.
2. A model is identified for the observed data.

3. The model parameters are estimated.
4. If the hypotheses of the model are validated, proceed with this step; otherwise, go to step 1 to refine the model. The model is now ready for forecasting.

Predicting future prices

Predicting future prices is one subproblem of predicting the future; this is just a sign of the difficulty of this issue. Another similar issue under this domain is estimating the future market demands.

The stock market is a good example of predicting the future price; here, the prices changes along with time. Predicting future prices helps estimate the equity returns in the future and also helps in deciding the timing for financial investors, such as when to buy/sell.

The price graph shows an oscillation. Many factors can affect this value, even the psycology of humans.

The key problem in this prediction is the huge data volume. As a direct result, the algorithms to be used for this topic need to be efficient. The other key problem is that the price might change dramatically in a short time. Also, the complexity of the market is definitely a big problem.

The data instances for the future price prediction include quantitative attributes such as technical factors, and they also include macroeconomics, microeconomics, political events, and investor expectations. Moreover, they include the domain expert's knowledge.

All the solutions based on the aggregate disperse information from the preselected factors.

Price is obviously affected by time, and is a time-series variable too. To forecast the future value, the time-series-analysis algorithm will apply. ARIMA can be used to forecast the prices of the next day.

One classical solution is to determine the trends of the market at a very early stage. Another solution is fuzzy solutions, which is also a good choice because of the huge volume of dataset and wide range of factors affecting the prices.

Stock market data and time-series clustering and classification

Time-series clustering has been proven to provide effective information for further research. In contrast to the classic clustering, the time-series dataset comprises data changed with time.

The hError algorithm

This algorithm is denoted as "**clustering seasonality patterns in the presence of errors**"; the summarized algorithm is listed in the following figure. The major characteristic of this algorithm is to introduce a specific distance function and a dissimilarity function.

Algorithm $hError(A, G)$
Input: $A_i = \{(x_{i1}, \sigma_{i1}), (x_{i2}, \sigma_{i2}), ..., (x_{iT}, \sigma_{iT})\}, i = 1, 2, ...n$
$G =$ number of clusters.
Output: $Cluster(i), i = 1, 2, ..., G.$
Start
  for $i = 1$ to n
    $Cluster(i) = \{i\}$
    $seas(i) = A_i$
  end
  $NumClust = n$
  while $NumClust > G$
    for $1 \leq i < j \leq NumClust$
      calculate $d_{ij} = dist(seas(i), seas(j))$ using equation (9)
    end
    $(I, J) = \arg\min_{1 \leq i < j \leq NumClust} d_{ij}$
    $Cluster(I) = Cluster(I) \cup Cluster(J)$
    $seas(I) = merge(seas(I), seas(J))$ using equations (10) and (11)
    $Cluster(J) = Cluster(NumClust)$
    $seas(J) = seas(NumClust)$
    $NumClust = NumClust - 1$
  end
  return $Cluster(i), i = 1, 2, ..., G$
end

Time-series classification with the 1NN classifier

The summarized pseudocode for the 1NN classifier algorithm is as follows:

Time Series Classification with 1NN Classifier

Input: Labelled time series data set $\mathbb{T}$, similarity measure operator $SimDist$, number of crosses k

Output: Average 1NN classification error ratio and standard deviation

```
Randomly divide 𝕋 into k stratified subsets 𝕋_1, ..., 𝕋_k
Initialize an array ratios[k]
for Each subset 𝕋_i of 𝕋 do
    if SimDist requires parameter tuning then
        Randomly split 𝕋_i into two equal size stratified subsets 𝕋_i1 and 𝕋_i2
        Use 𝕋_i1 for parameter tuning, by performing a leave-one-out classification with 1NN classifier
        Set the parameters to values that yields the minimum error ratio from the leave-one-out tuning process
    Use 𝕋_i as the training set, 𝕋 − 𝕋_i as the testing set
    ratio[i] ← the classification error ratio with 1NN classifier
return Average and standard deviation of ratios[k]
```

The R implementation

Please take a look at the R codes file `ch_08_herror.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_08_herror.R")
```

Stock market data

Stock market is a complex dynamic system, and many factors can affect the market. For example, breaking financial news is also a key factor for this topic.

The characteristics of stock market data are large volume (near infinite), real time, high dimensions, and high complexity. The various data streams from stock markets go along with many events and correlations.

Stock market sentiment analysis is one topic related to this domain. The stock market is too complex with many factors that can affect the market. People's opinion or sentiment is one of the major factors.

The real-time information needs from the stock market require the fast, efficient online-mining algorithms. The time-series data accounts for various data including the stock market data that updates along with time.

To predict the stock market, the past data is really important. Using the past return on certain stock, the future price of that stock can be predicted based on the price-data stream.

Web click streams and mining symbolic sequences

Web click streams data is large and continuously emerging, with hidden trends buried and to be discovered for various usages, such as recommendations. **TECNO-STREAMS** (**Tracking Evolving Clusters in NOisy Streams**) is a one-pass algorithm.

The TECNO-STREAMS algorithm

The whole algorithm is modeled on the following equations: the robust weight or activation function **(1)**, influence zone **(2)**, pure simulation **(3)**, optimal scale update **(4)**, incremental update of pure simulation and optimal update **(5)** and **(6)**, the simulation and scale values **(7)**, and finally, the D-W-B-cell update equations **(8)**.

$$w_{ij} = w_i\left(d_{ij}^2\right) = e - \left(\frac{d_{ij}^2}{2\sigma_{i,j}^2} + \frac{\left(J - j\right)}{\tau}\right)$$

$$IZ_i = \left\{x_j \in x_a \mid w_{ij} \geq w_{\min}\right\}$$

$$S_{ai,J} = \frac{\sum_{j}^{J} = 1 w_{ij}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{\sum_{j}^{J} = 1 w_{ij} d_{ij}^{2}}{2\sum_{j=1}^{J} w_{ij}}$$

$$S_{ai,J} = \frac{e^{-\frac{1}{\tau}} W_{i,J-1} + w_{iJ}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{e^{-\frac{1}{\tau}} \sigma_{i,J}^{2} W_{i,J-1} + w_{iJ} d_{i,J}^{2}}{2\left(e^{-\frac{1}{\tau}} W_{i,J-1} + w_{iJ} \right)}$$

$$S_{i} = S_{ai,J} + \alpha(t) \frac{\sum_{l=1}^{N_B^i} w_{il}}{\sigma_{i,J}^{2}} - \beta(t) \frac{\sum_{l=1}^{N_B^i} w_{il}}{\sigma_{i,J}^{2}}$$

$$\sigma_{i,J}^{2} = \frac{1}{2} \frac{D_{i,J}^{2} + \alpha(t) \sum_{l=1}^{N_B^i} w_{il} d_{l}^{2} - \beta(t) \sum_{l=1}^{N_B^i} w_{il} d_{l}^{2}}{Wi,J + \alpha(t) \sum_{l=1}^{N_B^i} w_{il} - \beta(t) \sum_{l=1}^{N_B^i} w_{il}}$$

The similarity measures applied in the learning phase are defined here:

$$S_{\cos ij} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sqrt{\sum_{k=1}^{n} x_{ik} \sum_{k=1}^{n} x_{ik} p_{jk}}}$$

$$S_{\cos ij} = \sqrt{\mathrm{Prec}_{ij}^{L} Covg_{ij}^{L}}$$

$$\mathrm{Prec}_{ij}^{L} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sum_{k=1}^{n} p_{jk}}$$

$$Covg_{ij}^{L} = \frac{\sum_{k=1}^{n} x_{ik} \times p_{jk}}{\sum_{k=1}^{n} x_{jk}}$$

$$S_{\min ij} = \min\left\{\mathrm{Prec}_{ij}^{L}, \mathrm{Covg}_{ij}^{L}\right\}$$

The similarity measures applied in the validation phase are defined here:

$$\mathrm{Prec}_{ij}^{v} = \frac{\sum_{k=1}^{n} pL_{ik} \times pGT_{jk}}{\sum_{k=1}^{n} pL_{jk}}$$

$$Covg_{ij}^{v} = \frac{\sum_{k=1}^{n} pL_{ik} \times pGT_{jk}}{\sum_{k=1}^{n} pGT_{jk}}$$

The summarized pseudocode for the TECNO-STREAMS algorithm is as follows:

TECNO-STREAMS Algorithm:
(optional steps are enclosed in [])

Fix the maximal population size, $N_{B_{max}}$*;*
Initialize D-W-B-cell population and $\sigma_i^2 = \sigma_{init}$ *using the first* $N_{B_{max}}$ *input data points;*
Compress immune network into K *subnets using 2 iterations of K Means;*
Repeat for each incoming input data point $\mathbf{x}_j$ {
  Present input data point to each subnet centroid, $\mathbf{C}_k, k = 1, \cdots, K$ *in network : Compute distance, activation weight,* w_{kj} *and update* σ_k^2 *incrementally using (6);*
  Determine the most activated subnet (the one with maximum w_{kj}*);*
  IF All B-cells in most activated subnet have $w_{ij} < w_{min}$ *(input data point does not sufficiently activate subnet) THEN*{
    Create by duplication a new D-W-B-cell = $\mathbf{x}_j$ *and* $\sigma_i^2 = \sigma_{init}$*;*
  }
  ELSE {
   Repeat for each D-W-B-cell$_i$ *in most activated subnet* {
    IF $w_{ij} > w_{min}$ *(input data point activates D-W-B-cell*$_i$*) THEN*
     Refresh age ($t = 0$*) for D-W-B-cell*$_i$*;*
    ELSE
     Increment age (t*) for D-W-B-cell*$_i$*;*
    Compute distance from input data point $\mathbf{x}_j$ *to D-W-B-cell*$_i$*;*
    Compute D-W-B-cell$_i$*'s stimulation level using (7);*
    Update D-W-B-cell$_i$*'s* σ_i^2 *using (8);*
   }
  }
  Clone and mutate D-W-B-cells;
  IF population size > $N_{B_{max}}$ *Then* {
   IF (Age of B-cell < t_{min}*) THEN*
    Temporarily scale D-W-B-cell's stimulation level to the network average stimulation;
   Sort D-W-B-cells in ascending order of their stimulation level;
   Kill worst excess (top ($N_B - N_{B_{max}}$*) according to previous sorting) D-W-B-cells;*
   [or move oldest/mature D-W-B-Cells to secondary (long term) storage];
  }
  Compress immune network periodically (after every T *input data points), into* K *subnets using 2 iterations of K Means with the previous centroids as initial centroids;*
}

The R implementation

Please take a look at the R codes file `ch_08_tecno_stream.R` from the bundle of R codes for previous algorithm. The codes can be tested with the following command:

```
> source("ch_08_tecno_stream.R")
```

Web click streams

The web click streams denote the user's behavior when visiting the site, especially for e-commerce sites and **CRM** (**Customer Relation Management**). The analysis of web click streams will improve the user experience of the customer and optimize the structure of the site to meet the customers' expectation and, finally, increase the income of the site.

In other aspects, web click streams mining can be used to detect DoS attacks, track the attackers, and prevent these on the Web in advance.

The dataset for web click stream is obviously the click records that get generated when the user visits various sites. The major characteristics of this dataset are that it is huge, and the size goes on increasing.

Mining sequence patterns in transactional databases

Mining sequence patterns can be thought of as association discovery over the temporal data or sequence dataset. Similarly, the classic pattern-mining algorithm should be extended or modified as per the sequence dataset's scenario.

The PrefixSpan algorithm

PrefixSpan is a frequent sequence-mining algorithm. The summarized pseudocode for the PrefixSpan algorithm is as follows:

Algorithm PREFIXSPAN

PREFIXSPAN $(\mathbf{D_r}, \mathbf{r}, minsup, \mathcal{F})$: $\mathbf{D_r} \leftarrow \mathbf{D}$, $\mathbf{r} \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$
foreach $s \in \Sigma$ *such that* $sup(s, \mathbf{D_r}) \geq minsup$ **do**
  $\mathbf{r}_s = \mathbf{r} + s$ `// extend r by symbol` s
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{(\mathbf{r}_s, sup(s, \mathbf{D_r}))\}$
  $\mathbf{D}_s \leftarrow \emptyset$ `// create projected data for symbol` s
  foreach $\mathbf{s}_i \in \mathbf{D_r}$ **do**
    $\mathbf{s}'_i \leftarrow$ projection of $\mathbf{s}_i$ w.r.t symbol s
    Remove any infrequent symbols from $\mathbf{s}'_i$
    Add $\mathbf{s}'_i$ to $\mathbf{D}_s$
  if $\mathbf{D}_s \neq \emptyset$ **then** PREFIXSPAN $(\mathbf{D}_s, \mathbf{r}_s, minsup, \mathcal{F})$

The R implementation

Please take a look at the R codes file `ch_08_prefix_span.R` from the bundle of R codes for the previous algorithm. The codes can be tested with the following command:

```
> source("ch_08_prefix_span.R")
```

Time for action

Here are some questions for you to know whether you have understood the concepts:

- What is a stream?
- What is a time series?
- What is a sequence?

Summary

In this chapter, we looked at stream mining, time-series analysis, mining symbolic sequences, and mining sequence patterns.

The next chapter will cover the major topics related to graph mining, algorithms, and some examples related to them.

9
Graph Mining and Network Analysis

In this chapter, you will learn the algorithms written in R for graph mining and network analysis.

In this chapter, we will cover the following topics:

- Graph mining
- Mining frequent subgraph patterns
- Social network mining
- Social influence mining

Graph mining

Grouping, messaging, dating, and many other means are the major forms of social communication or the classic social behavior in the social network. All these concepts are modeled with graphs; that is, nodes, edges, and other attributes. Graph mining is developed to mine this information, which is similar to other types of information, such as biological information, and so on.

Graph

Graph G contains nodes V and edges E and is represented with an equation, $G = (V, E)$. As per graph mining, there are some concepts that need to be clarified. There are two types of graphs: directed graphs, which have ordered pairs of vertices in the edge set, E, and undirected graphs.

Graph mining algorithms

Although the data instances under research are very different from the other data types that we saw earlier in this book, graph-mining algorithms still include frequent pattern (subgraph) mining, classification, and clustering.

In the next section, we will look at frequent subgraph patterns mining algorithm, links mining, and clustering.

Mining frequent subgraph patterns

The subgraph pattern or graph pattern is an important application of data mining; this is used for bioinformatics, social network analysis, and so on. Frequent subgraph patterns are patterns that occur frequently in a set of graphs or in a large graph.

The gPLS algorithm

gPLS

Input: Training examples $(G_1, y_1), (G_2, y_2), ..., (G_n, y_n)$
Output: Weight vectors w_i, $i = 1, ..., m$
1: $r_1 = y$, $X = \varnothing$;
2: **for** $i = 1, ..., m$ **do**
3: $P_i = \{p || \sum_{j=1}^{n} r_{ij} x_{jp}| \geq \epsilon\}$;
4: X_{P_i}: design matrix restricted to P_i;
5: $X \leftarrow X \cup X_{P_i}$;
6: $v_i = X^T r_i / \eta$;
7: $w_i = v_i - \sum_{j=1}^{i-1} (w_j^T X^T X v_i) w_j$;
8: $t_i = X w_i$;
9: $r_{i+1} = r_i - (y^T t_i) t_i$;

The GraphSig algorithm

GraphSig(D, min_sup, $maxPvalue$)

Input: Graph dataset D, support threshold min_sup, p-value threshold $maxPvalue$
Output: The set of all significant sub-feature vectors A

```
D' ← ∅;
A ← ∅;
for each g ∈ D do
    D' ← D' + RWR(g);
for each node label a in D do
    D'_a ← {v̲|v̲ ∈ D', label(v̲) = a};
    S ← FVMine(floor(D'_a), D'_a, 1);
    for each vector v̲ ∈ S do
        V ← {u|u is a node with label a, v̲ ⊆ vector(u)};
        E ← ∅;
        for each node u ∈ V do
            E ← E + CutGraph(u, radius);
        A ← A + Maximal_FSM(E, freq);
```

The gSpan algorithm

The summarized pseudocode for the gSpan algorithm is as follows:

Algorithm gSpan

```
// Initial Call:  C ← ∅
gSpan (C, D, minsup):
ℰ ← RightMostPath-Extensions(C, D)
foreach (t, sup(t)) ∈ ℰ do
    C' ← C ∪ t // extend the code with extended edge tuple t
    sup(C') ← sup(t) // record the support of new extension

    if sup(C') ≥ minsup and IsCanonical (C') then
        gSpan (C', D, minsup)
```

Rightmost path extensions and their supports

RIGHTMOSTPATH-EXTENSIONS $(C, \mathbf{D})$:
$R \leftarrow$ nodes on the rightmost path in C
$u_r \leftarrow$ rightmost child in C
$\mathcal{E} \leftarrow \emptyset$
foreach $G_i \in \mathbf{D}$, $i = 1, \cdots, n$ **do**
  if $C = \emptyset$ **then**

    foreach $\langle L(x), L(y), L(x,y) \rangle \in G_i$ **do**
      $f = \langle 0, 1, L(x), L(y), L(x,y) \rangle$
      Add tuple f to $\mathcal{E}$ along with graph id i

  else
    $\Phi_i = \text{SUBGRAPHISOMORPHISMS}(C, G_i)$
    foreach *isomorphism* $\phi \in \Phi_i$ **do**

      foreach $x \in N_{G_i}(\phi(u_r))$ *such that* $\exists v \leftarrow \phi^{-1}(x)$ **do**
        if $v \in R$ *and* $(u_r, v) \notin G(C)$ **then**
          $b = \langle u_r, v, L(u_r), L(v), L(u_r, v) \rangle$
          Add tuple b to $\mathcal{E}$ along with graph id i

      foreach $u \in R$ **do**
        foreach $x \in N_{G_i}(\phi(u))$ *and* $\nexists \phi^{-1}(x)$ **do**
          $f = \langle u, u_r + 1, L(\phi(u)), L(x), L(\phi(u), x) \rangle$
          Add tuple f to $\mathcal{E}$ along with graph id i
foreach *extension* $s \in \mathcal{E}$ **do**
  $sup(s) =$ number of distinct graph ids that support tuple s
return *set of pairs* $\langle s, sup(s) \rangle$ *for extensions* $s \in \mathcal{E}$, *in tuple sorted order*

The subgraph isomorphism enumeration algorithm

```
Enumerate subgraph isomorphisms
SubgraphIsomorphisms (C = {t_1, t_2, ··· , t_k}, G):
Φ ← {φ(0) → x | x ∈ G and L(x) = L(0)}
foreach t_i ∈ C, i = 1, ··· , k do
    ⟨u, v, L(u), L(v), L(u, v)⟩ ← t_i
    Φ' ← ∅
    foreach partial isomorphism φ ∈ Φ do
        if v > u then
            foreach x ∈ N_G(φ(u)) do
                if ∄φ^{-1}(x) and L(x) = L(v) and L(φ(u), x) = L(u, v) then
                    φ' ← φ ∪ {φ(v) → x}
                    Add φ' to Φ
        else
            if φ(v) ∈ N_{G_j}(φ(u)) then  Add φ to Φ'
    Φ ← Φ'
return Φ
```

The canonical checking algorithm

```
IsCanonical (C):
D_C ← {G(C)}
C* ← ∅ // initialize canonical DFScode
for i = 1 ··· k do
    ℰ = RightMostPath-Extensions(C*, D_C)
    (s_i, sup(s_i)) ← min{ℰ}
    if s_i < t_i then
        return false // C* is smaller, thus C is not canonical
    C* ← C* ∪ s_i
return true
```

The R implementation

Please take a look at the R codes file `ch_09_gspan.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_09_gspan.R")
```

Social network mining

Social network is based on human interactions, from the most classical definition. The data instances collected in the social network have graph-like and temporal characteristics. There are two major strategies for data mining tasks for social networks: one is linkage-based or structure-based, and the other is content-based. The data instances collected in the social network also have two kinds of data instances: static and dynamic or times-series data, such as the tweets on Twitter. Due to the characteristics of the data instance of graphs, there are vast versatile algorithms developed to solve the challenges.

Community detection and the shingling algorithm

Algorithm DenseSubgraph $\langle v, \Gamma(v) \rangle$

Choose c_1, s_1, c_2, s_2
Shingle2$(\langle v, \Gamma(v) \rangle, c_1, s_1, c_2, s_2)$
Let $S = \langle s, \Gamma(s) \rangle$ be first-level shingles
Let $\langle t, \Gamma(t) \rangle$ be second-level shingles
$\mathcal{C} = \mathbf{CC}(\langle t, \Gamma(t) \rangle)$
For $C \in \mathcal{C}$ do
    Output $\cup_{s \in C} \Gamma(s)$ as a dense subgraph

Algorithm CC $(t_1, \Gamma(t_1), \ldots, t_m, \Gamma(t_m))$

Let $V_S = \cup_{i=1}^{m} \Gamma(t_i)$
Let $\mathcal{C} = \{\{s\} \mid s \in V_S\}$
For $i = 1$ to m do
  Merge the sets $\{$ Find$(s) \mid s \in \Gamma(t_i)\}$ in $\mathcal{C}$
For $C \in \mathcal{C}$
  Output cluster $\{s \mid s \in C\}$

Algorithm Shingle2$(v_1, \ldots, v_n, s_1, c_1, s_2, c_2)$

For $i = 1$ to n do
  $S_1(v_i) =$ **Shingle**$(\Gamma(v_i), c_1, s_1)$
Let $S = \cup_{i=1}^{n} S_1(v_i)$
For $s \in S$ do
  Let $\Gamma(s) = \{v \mid S_1(v) \ni s\}$
  $S_2(s) =$ **Shingle**$(\Gamma(s), c_2, s_2)$
Let $T = \cup_{s \in S} S_2(s)$
For $t \in T$ do
  Let $\Gamma(t) = \{s \in S \mid S_2(s) \ni t\}$
  Output $\langle t, \Gamma(t) \rangle$

Algorithm Shingle$(a_1, \ldots, a_n, s, c)$

Let H be a hash function from strings to integers
Let p be a large random prime (say, 32 bits)
Let $a_1, b_1, \ldots, a_c, b_c$ be random integers in $[1 \ldots p]$
For $i = 1$ to n do $x_i = H(\text{“}a_i\text{”})$
For $j = 1$ to c do
  For $i = 1$ to n do $y_i = (a_j * x_i + b_j) \bmod p$
  Let $y'_1, \ldots, y'_s$ be s minimal elements of y
  Let $z_j = H(\text{“}y'_1 \circ \cdots \circ y'_s\text{”})$
Output $z_1, \ldots, z_c$

The node classification and iterative classification algorithms

$\underline{\text{ICA}(V, E, W, Y_l)}$
Compute Φ^1 from V, E, W, Y_l
Train classifier using Φ_l
for $t \leftarrow 1$ **to** τ **do**
  Apply classifier to Φ_u^t to compute Y_u^t
  Update Φ_u^t
$\tilde{Y} \leftarrow Y^\tau$
return $\tilde{Y}$

The second-order algorithm to reduce the number of iterations is as follows:

$\underline{\text{MAP}(\text{Key } v_i, \text{ Value } y_i^{t-1})}$
Data: P
foreach $v_j \in V | (i, j) \in E$ **do**
  `Emit`$(v_j, (y_i^{t-1}, p_{ij}))$

The R implementation

Please take a look at the R codes file `ch_09_shingling.R` from the bundle of R codes for previously mentioned algorithms. The codes can be tested with the following command:

```
> source("ch_09_shingling.R")
```

Time for action

Here are some practice questions for you to check whether you have understood the concepts:

- What is a graph?
- What graph opportunities are used?
- What is the PageRank algorithm, and what is its application in web search?

Summary

In this chapter, we looked at:

- Graph mining. We also saw that the characteristics of graph data can be divided into frequent pattern mining, classification, and clustering
- Mining frequent subgraph patterns is done to find the frequent patterns in a set of graphs or a single massive graph
- Social network analysis includes a wide range of web applications with broad definitions, such as Facebook, LinkedIn, Google+, StackOverflow, and so on

In the next chapter, we will focus on the major topics related to web mining and algorithms and look at some examples based on them.

10
Mining Text and Web Data

In this chapter, you will learn the algorithm written in R for text mining and web data mining.

For text mining, the semistructured and nonstructured documents are the main dataset. There are a few of major categories of text mining, such as clustering, document retrieval and representation, and anomaly detection. The application of text mining includes, but is not limited to, topic tracking, and text summarization and categorization.

Web content, structure, and usage mining is one application of web mining. Web mining is also used for user behavior modeling, personalized views and content annotation, and so on. In another aspect, web mining integrates the result information from the traditional data-mining technologies and the information from WWW.

In this chapter, we will cover the following topics:

- Text mining and the TM package
- Text summarization
- The question answering system
- Genre categorization of web pages
- Categorization of newspaper articles and newswires into topics
- Web usage mining with web logs

Text mining and TM packages

Along with the appearance of text mining, due to the characteristics of text or documents, the traditional data-mining algorithms need some minor adjustments or extensions. The classical text-mining process is as follows:

The popular text-clustering algorithms include the distance-based clustering algorithm, the hierarchical clustering algorithm, the partition-based clustering algorithm, and so on.

The popular text-classification algorithms include decision trees, pattern-based classification, SVM classification, Bayesian classification, and so on.

As a popular preprocessing step, here are the details of the word-extraction algorithm.

Text summarization

The target of text summarization is to generate a concise and coherent conclusion or summary of the major information of the input. Three major steps are performed in most of the summarization systems. They are as follows:

1. Build a temporary structure that contains the main portion of the key point of the input text.
2. Next, the sentences from the input are scored with the output of the first step.
3. Lastly, a final summary that represents the input documents is set up by several sentences.

One popular strategy is to remove the unimportant information, clauses, or sentences and, at the same time, build classifiers to make sure that the key information is not thrown away, which is, in another viewpoint, the relative importance of topics functioned here during the summarization process. The final result is represented in a coherent way.

The summarization is a dynamic nonstop process. First, we need to build summaries on the set of old documents' dataset, which means multidocuments' summarization. The second step is to summarize the new documents when we get the result of the first step.

Due to the difficulty in building a new sentence for the summaries, one solution is extractive summarization, that is, to extract the most relevant sentences from the document dataset or documents. With the growth of the size of the documents set, the time evolution of the topics is also an important issue for which statistical topic modeling such as time-series-based algorithms are proposed.

There are two popular solutions for intermediate representation: topic representation and indicator representation. Sentence scoring, the score of each sentence, is determined by a couple of important factors such as the combination of characters. Summary sentence selection is determined by the most important *N* sentences.

There are many good characteristics that describe the final summary: it is indicative or informative, extract or abstract, generic or query-oriented, consists of background or just the news, is monolingual or cross lingual, and consists of a single document or multiple documents.

The benefit of text summarization is to improve the efficiency of document processing during which the summarization of a certain document can help the reader decide whether to analyze the document for specific purposes. One example is to summarize the multilingual, large (nearing unlimited), dynamic dataset of documents from various sources, including the Web. Examples include summarization of medical articles, e-mail, the Web, speech, and so on.

Topic representation

Topic representation such as topic signature plays an important role in the document-summarization system. Various topic representations are provided, such as topic signature, enhanced topic signature, thematic signature, and so on.

The topic signature is defined as a set of related terms; topic is the target concept, and signature is a list of terms that is related to the topic with a specific weight. Each term can be the stemmed content words, bigram or trigram:

$$TS = \{topic, signature\} = \{topic, < (t_1, w_1), \ldots, (t_n, w_n) >\}$$

The topic term selection process is as follows. The input document sets are divided into two sets: relevant and nonrelevant texts for the topic. Two hypotheses are defined:

$$Hypothesis\ 1, (H_1), P(R|T_i) = p = P(R|\tilde{t_i})$$

$$Hypothesis\ 2, (H_2), P(R|T_i) = p_1 \neq p_2 = P(R|\tilde{t_i})$$

The first hypothesis denotes that the relevance of a document is independent of the term, whereas the second hypothesis indicates a strong relevance with the presence of those terms, given that $p_1 \gg p_2$ and the 2 x 2 contingency table:

	R	$\tilde{R}$
t_1	O_{11}	O_{12}
t_1	O_{21}	O_{22}

O_{11} represents the frequency of the term, t_1, occurring in R, and O_{12}, is the O_{21} frequency of the term occurring in $\tilde{R}$. O_{21} is the frequency of the term, $\tilde{t_i} \neq t_i$, occurring in R, and O_{22} is the frequency of the term, $\tilde{t_i} \neq t_i$, occurring in $\tilde{R}$.

The likelihood of both hypotheses is calculated as follows; here, *b* denotes the binomial distribution:

$$L(H_1) = b(O_{11}; O_{11} + O_{12}, p) * b(O_{21}; O_{21} + O_{22}, p)$$

$$L(H_2) = b(O_{11}; O_{11} + O_{12}, p_1) * b(O_{21}; O_{21} + O_{22}, p_2)$$

$$-2\log\lambda = \frac{L(H_1)}{L(H_2)}$$

The algorithm to create the topic signature for a given topic is illustrated as follows:

1. classify documents as relevant or nonrelevant according to the given topic
2. compute the -2logλ value using Equation 3 for each term in the document collection
3. rank terms according to their -2logλ value
4. select a confidence level from the χ^2 distribution table; determine the cutoff associated weight and the number of terms to be included in the signatures

The multidocument summarization algorithm

Here is the **Graph-Based Sub-topic Partition Algorithm** (**GSPSummary**) algorithm for multidocument summarization:

Input: A document collection D about the topic T
Output: An array of summary sentences S

```
repeat
InitGraphMatrix(&M,D);
ArrayRS;
i = GSPRankMethod(M, DistThre, ζ, RS);
ArrayNeighbours = NeighbourSearch(i, M, NeighbourThre);
iLen = LengthOfSentence(i);
if ((iLen + iSummaryLen) > SelectThre) then
    break;
end
InsertIntoSelectedArray(S,i);
iSummaryLen+ = iLen;
UpdateRemainGraph(RS);
until (RS.size() >= MINGRAPHSIZE);
```

The GSPRankMethod is defined as follows:

```
Input: An array S of n sentences, distance threshold ϵ, a size N feed matrix J
Output: The ID of the salient sentence of S
Array DistMatrix[n][n];
Array Degree[n];
maxDist = −INFINITE;
for i ← 0 to n do
    for j ← 0 to n do
        DistMatrix[i][j]=dist(S[i],S[j]);
        if DistMatrix[i][j] < ϵ then DistMatrix[i][j]=1;
        Degree[i]++;
        else DistMatrix[i][j]=0;
    end
end
Normalization of matrix DistMatrix[i][j];
L = PowerMethod(DistMatrix);
R = J · L;
return the ID with maximal score from R;
```

The Maximal Marginal Relevance algorithm

Maximal Marginal Relevance (**MMR**), which is comparatively suited for query-based and multidocument summarization, selects the most important sentence in each iteration of sentence selection. Each selected sentence has minimal relevance to the selected sentence set.

$$MMR \stackrel{\text{def}}{=} Arg \max_{D_i \in R \setminus S} \left[\lambda(Sim_1(D_i, Q) - (1-\lambda) \max_{D_j \in S} Sim_2(D_i, D_j)) \right]$$

The summarized algorithm of MMR is as follows:

MMR

Input: candidate set S and result set size k
Output: result set $R \subseteq S$, $|R| = k$

```
R ← ∅
s_s ← argmax_{s_i ∈ S}(mmr(s_i))
S ← S \ s_s
R ← s_s
while |R| < k do
    s_s ← argmax_{s_i ∈ S}(mmr(s_i))
    S ← S \ s_s
    R ← R ∪ s_s
```

The R implementation

Please take a look at the R codes file `ch_10_mmr.R` from the bundle of R codes for the preceding algorithms. The codes can be tested with the following command:

```
> source("ch_10_mmr.R")
```

The question answering system

The **question answering** (**QA**) system is a one hot topic related to IR, IE, NLP, data mining, and so on. A QA system mines a large set of texts to find a short phrase or sentence that answers a user's question with a certain precision. If the source is the Web or, even further, the whole information world, the challenge of the QA system increases dramatically.

There are basically three kinds of QA systems: slot filling, for which the format of the query and answer are similar; limited domain, for which the domains are limited with dictionaries and ontologies; and open domain, for which there is no limitation of domains.

One of the conceptual architectures among various solutions is illustrated in the following image. The input of the QA system is natural language questions; the output of the system, that is, the answer to the input question, is provided in natural language too. The system is composed of three major parts: the user interface, the processing part for questions, and the generation part for answers.

Assuming the question to be a coherent sentence, the questions from the user interface are processed, and the final right query is generated after a question-term identification, such as word segmentation and key word extraction and expansion. The ontology base here serves to support query expansion.

The answer is selected or generated from the FAQ base by the query that is provided in the earlier steps.

One popular question answering system is Yahoo! Answers, where a huge number of questions are asked and answered every day.

Genre categorization of web pages

Genre categorization can be used for large article corpora and web pages. A genre can be defined in terms of purpose and physical form. It denotes any widely accepted categories of texts defined by common communicative purpose or other functional traits, and the categories are extensible. The web genre can also be defined based on the facets, complexity of the language, subjectivity, and number of graphics. Genre categorization has many applications such as improving search efficiency and satisfying the users' information need.

For the WWW or web pages, the genre can be defined as the usability of the miner and feasibility with respect to the efficiency.

There are some major challenges for this web page genre categorization. One is the instability of the Web itself. The second is the complex and unpredictable properties of web pages. The third is how to judge the genre for a specific web page. There are more challenges, but they are not listed here, or they will appear in future applications. For certain web pages, they might have multiple genres or no genre for existing recognized genres libraries.

Due to the fast pace of the evolution of the Web, new genres are continuously introduced to the current genre classes and the current genre classes are continuously updated and upgraded.

Possible solutions include, but are not limited to, Naïve Bayes, k-Nearest Neighbor, SVM, and tree nodes as classification methods.

Categorizing newspaper articles and newswires into topics

Articles and newswires denote the huge periodical source of events of knowledge at different periods of time. The classification of text is the preprocessing step to store all these documents into a specific corpus. The categorization of text is the base of text processing.

We will now introduce an N-gram-based text-classification algorithm. From a longer string, an N-character slice is called N-gram. The key point of this algorithm is the calculation of the profiles of the N-gram frequencies.

Before the introduction of the algorithm, here are the necessary illustrations of a couple of concepts adopted in the algorithm:

The N-gram-based text categorization

The summarized pseudocodes for the N-gram-based text-categorization algorithm are as follows:

```
1: NGramClassifier (D, K, S) {
2:   set the initial value as 1/d for the weight of each training tuple;
3:   for (j ⟵ 1; j ≤ k; j++) {
4:     Create bootstrap sample, D_j, by sampling D with replacement
5:     Sample D with replacement according to the tuple weights to get D_j;
6:     Learning a model M_j with D_j;
7:     compute the error rate of M_j, i.e., error( D_j);
8:     if( error(M_j) > 0.5 ){
9:       go back to step 3 and try again;
10:    }

11:    for( each tuple in D_j that was corretly classified ){

12:      updates its weight value by multiplying with error(M_j)/(1−error(M_j));

13:    }
14:    normalize the weight of each tuple;
15:  }
16: }
```

The N-gram frequency-generation algorithm is as follows:

1: $NGramFreqGen(D, K, S)$ {
2: *Split the text into separate tokens consisting only letters and apostrophes.*
3: *Scan down each token, generating all possible* $n - gram$, *for* $n = 1$ *to* 5.
4: *Hash into a table to find the counter for the ngram, and increment it.*
5: *When the process done, export all* $n - gram$ *and corresponding counts;*
6: *Sorts the ocunts into reverse order by counts of occurences;*
7: *if(* $error(M_j) > 0.5$ *){*
8: *go back to step 3 and try again;*
9: }

10: *for(each tuple in* D_j *that was corretly classified){*
11: *updates its weight value by multiplying with* $\frac{error(M_j)}{1-error(M_j)}$;
12: }
13: *normalize the weight of each tuple;*
14: }
15: }

The R implementation

Please take a look at the R codes file `ch_10_ngram_classifier.R` from the bundle of R codes for the above algorithms. The codes can be tested with the following command:

```
> source("ch_10_ngram_classifier.R")
```

Web usage mining with web logs

Web usage mining denotes the discovery and analytics of patterns in web logs (such as system access logs) and transactions. The output is the relation of user interaction and resources on the Web. The user behavior can be identified based on this output. The web logs record the track of the web user's interaction with web servers, web proxy servers, and browsers.

The popular web usage mining process is illustrated in the images, and it includes three major steps: data collection and preprocessing, pattern discovery, and pattern analysis.

The preprocessing contains data cleaning, session identification, and data conversion. The pattern discovery includes path analysis, association rules, sequential patterns, and cluster and classification rules.

The FCA-based association rule mining algorithm

The summarized pseudocodes for the FCA-based association rule mining algorithm are as follows:

FCA-based Association Rule Mining

Input:

1. *WUL* – Web Usage Lattice, which is based on formal context $K = (G, M, I)$, G comprises all user access sessions in web logs, and the attribute set M includes all web pages. The relation I indicates which web pages are accessed by which access sessions.
2. $NL = \{N_1, N_2, \ldots, N_m\}$ – a set of concept nodes in *WUL*, where $N_i = \langle A_i, B_i, P_i \rangle$, $A_i \subseteq G$ is the extent of N_i , $B_i \subseteq M$ is the intent of N_i, $P_i = \{N_{i1}, N_{i2}, \ldots, N_{ip}\} \subseteq NL$ is the immediate parent nodes of N_i.
3. *MinSup* – minimum support threshold.
4. *MinConf* – minimum confidence threshold.

Output:

1. $ARS = \{AR_1, AR_2, \ldots, AR_n\}$ – a set of FCA-based association rules, where $AR_i = (X_i \Rightarrow Y_i$, *Support*, *Confidence*), $X_i, Y_i \subset M$, and $X_i \cap Y_i = \varnothing$.

Process:

1. Initialize $ARS = \varnothing$.
2. For each $N_i \in NL$, if $P_i \neq \varnothing$ and $Sup = |A_i|/|G| \geq MinSup$, do
3. If $|P_i| = 1$ and $B_{i1} \neq \varnothing$, do
4. Insert $((B_i - B_{i1}) \Rightarrow B_{i1}$, *Sup*, 100%) into *ARS* as a FCA-based exact rule.
5. For each $N_{ij} \in P_i$, if $B_{ij} \neq \varnothing$ and $Conf = |A_i|/|A_{ij}| \geq MinConf$, do
6. Insert $(B_{ij} \Rightarrow (B_i - B_{ij})$, *Sup*, *Conf*) into *ARS* as a FCA-based approximate rule.
7. Return *ARS*.

The R implementation

Please take a look at the R codes file `ch_10_fca.R` from the bundle of R codes for the above algorithms. The codes can be tested with the following command:

```
> source("ch_10_fca.R")
```

Time for action

Here are some questions for you to know whether you have understood the concepts:

1. What are the characteristics of text mining?
2. What is the difference between text mining and data mining?
3. What is the difference between web mining and data mining?

Summary

In this chapter, we looked at text mining, which is to find brand new or unknown information by extracting information from the documents dataset. We also looked at how text summarization presents the condensed result of the document set under research; in other words, takes a source, extracts contents, and presents the key contents in a condensed format that is sensitive to the final needs. Also, we looked at how genre classification discriminates between documents by form, style, and the targeted system or audience. We also covered the question answering system, topic detection, and web mining.

In this book, many useful data-mining algorithms are illustrated in the form of the R language, which has been there for years, even decades. The most popular algorithms are included with a detailed description. You can start working with the knowledge structure of the classical and living data-mining algorithms and solutions built with R.

Algorithms and Data Structures

Here is a list of algorithms related to association rules mining; it is only a small portion of the available algorithms, but it has proved to be effective:

Approach	**Dataset**	**Sequential pattern mining**	**Sequential rule mining**	**Frequent itemset mining**	**Association rule mining**
Apriori	Transaction			Yes	
AprioriTid	Transaction			Yes	
DHP (Direct Hashing and Pruning)	Transaction			Yes	
FDM (Fast Distributed Mining of association rules)	Transaction			Yes	
GSP (Generalized Sequential Patterns)	Sequence	Yes			
DIC	Transaction			Yes	
Pincer Search (the Pincer-search algorithm)	Transaction			Yes	
CARMA (Continuous Association Rule Mining Algorithm)	Transaction			Yes	

Approach	Dataset	Sequential pattern mining	Sequential rule mining	Frequent itemset mining	Association rule mining
CHARM (Closed Association Rule Mining)	Transaction			Yes (closed)	
Depth-project	Transaction			Yes (maximal)	
Eclat	Transaction			Yes	
SPAD	Sequence	Yes			
SPAM	Sequence	Yes			
Diffset	Transaction			Yes	
FP-growth	Transaction			Yes	FP-growth
DSM-FI (Data Stream Mining for Frequent Itemsets)	Transaction			Yes	
PRICES	Transaction			Yes	
PrefixSpan	Sequence	Yes			
Sporadic Rules	Transaction				Yes
IGB	Transaction				Yes
GenMax	Transaction			Yes (maximal)	
FPMax (Frequent Maximal Item Set)	Transaction			Yes	
FHARM (Fuzzy Healthy Association Rule Mining)	Transaction			Yes	
H-Mine	Transaction			Yes	
FHSAR	Transaction				Yes
Reverse Apriori	Transaction			Yes (maximal)	
DTFIM	Transaction			Yes	
GIT tree	Transaction			Yes	
Scaling Apriori	Transaction			Yes	
CMRules	Sequence		Yes		
Minimum effort	Transaction			Yes (maximal)	

Approach	**Dataset**	**Sequential pattern mining**	**Sequential rule mining**	**Frequent itemset mining**	**Association rule mining**
TopSeqRules	Sequence		Yes		
FPG ARM	Transaction			Yes	
TNR	Transaction				Yes
ClaSP	Sequence	Yes (closed)			

Index

Symbols

1NN classifier algorithm 232

A

AdaBoost algorithm 113115
affinity propagation (AP) clustering
about 156158
R implementation 158
spectral clustering algorithm 159
used, for unsupervised image categorization 159
agglomerative clustering
about 160
pseudocode 161
algorithm, for association rule generation
R implementation 56, 57
algorithms, association rules mining
listing 267
A-Priori 31
A-Priori algorithm
about 38
data structure 39
input data characteristics 39
join action 39
prune action 40
R implementation 41-44
variants 44
association rules
about 36, 37
generating, with algorithm 55, 56
associations 32
associative classification
about 128, 129
Classification Based on Association (CBA) 129
Classification Based on Multiple Association Rules (CMAR) 129
attribute
about 17
Auto Regressive Integrated Moving Average (ARIMA) algorithm
about 227-229
future prices, predicting 229

B

backpropagation algorithm. *See* **BP algorithm**
bagging algorithm
about 112, 113
input parameters 113
Balanced Iterative Reducing and Clustering using Hierarchies algorithm. *See* **BIRCH algorithm**
basket 33
Bayes classification
about 93
likelihood estimation 94, 95
prior probability estimation 94
pseudocode 95
R implementation 96
used, for Trojan traffic identification 93-98
Bayesian belief network algorithm. *See* **BBN algorithm**
Bayesian hierarchical clustering algorithm
about 164
pseudocode 164

BBN algorithm
about 118, 119
biological traits 120
R implementation 120
big data
about 2, 3
data types 2, 3
efficiency 3
scalability 3
binning 22
BIRCH algorithm
about 161, 162
CF-Tree insertion 161
CF-Tree rebuilding 161
pseudocode 162
Bonferroni correction 15
Bonferroni's Principle 15
boosting algorithm
about 113-115
BP algorithm
about 131-133
input parameters 133
parallel version, with MapReduce 135, 136
pseudocode 134
R implementation 135
brute-force algorithm 219

C

C4.5 algorithm
characteristics 83
parallel version, MapReduce 86
pseudocode 84
R implementation 85
used, for web spam detection 82-90
CART algorithm
about 90
characteristics 90
pseudocode 91, 92
R implementation 92
used, for web key resource page judgment 90-93
categorical attributes
about 19
nominal 19
ordinal 19
CF-Tree 161
chameleon algorithm
about 162
agglomerative hierarchical clustering 163
graph partitioning 162
sparsification 162
Charm algorithm
about 54
R implementation 55
CLARA algorithm
about 153, 154
pseudocode 154
R implementation 154
CLARANS algorithm
about 155
input parameters 155
pseudocode 156
R implementation 156
classification
about 68, 69
classification validation 68
training (supervised learning) 68
Classification and Regression Trees algorithm. *See* **CART algorithm**
classification-based methods
about 212
OCSVM (One Class SVM) algorithm 212
one-class nearest neighbor algorithm 213, 214
R implementation 214
web server performance, monitoring 214
Classification Based on Association (CBA)
about 129
pseudocode 129
Classification Based on Multiple Association Rules (CMAR) 129
classification, with frequent patterns
about 128
associative classification 128
discriminative frequent pattern-based classification 129
R implementation 130
sentential frequent itemsets 130
CLIQUE algorithm
about 181, 182
characteristics 182
pseudocode 182, 183
R implementation 183

web sentiment analysis 183
closed frequent itemsets
mining, with Charm algorithm 54
clustering
about 139
clustering algorithm design 140
cluster validation 140
feature selection 140
result interpretation 140
clustering algorithm
about 139
limitations 140, 141
usage 140
clustering-based methods
about 210
hierarchical clustering algorithm 210
k-means algorithm 210, 211
ODIN algorithm 211
R implementation 212
CLustering In QUEst algorithm. *See* **CLIQUE algorithm**
Clustering LARge Application algorithm. *See* **CLARA algorithm**
Clustering Large Applications based on RANdomized Search algorithm. *See* **CLARANS algorithm**
collective outliers
about 217
characteristics 219
route outlier detection (ROD) algorithm 218
Comprehensive R Archive Network (CRAN) 14
conditional anomaly detection (CAD) algorithm
about 215, 217
R implementation 217
conditional probability tables (CPT) 118
constraint-based frequent pattern mining 58, 59
contextual outliers
conditional anomaly detection (CAD) algorithm 215, 217
mining 214
continuous, numeric attributes 19
correlation rules 37, 38
credit card fraud detection 198
credit card transaction flow
mining 227
CRISP-DM
about 5
business understanding 6
data preparation 6
data understanding 6
deployment 7
evaluation 7
modeling 7
CRM (Customer Relation Management) 237
Cross-Industry Standard Process for Data Mining. *See* **CRISP-DM**
Cubic Clustering Criterion 211
CUR decomposition 25
customer purchase data analysis 191

D

DASL
about 4
URL 4
Data and Story Library. *See* **DASL**
data attributes
about 17, 18
categorical attributes 19
numeric attributes 18
data attributes, views
algebraic or geometric view 18
probability view 18
data classification
linearly separable 122
nonlinearly separable 122
data cleaning
about 21
junk 22
missing values, avoiding 21, 22
noisy data 22
outlier 22
data description
about 17-19
measures of central tendency 19
measures of data dispersion 19
data dimension reduction
about 23
CUR decomposition 25
eigenvalues 24

eigenvectors 24
PCA 24
SVD 24
data discretization
about 25, 26
by binning 26
by cluster analysis 26
by correlation analysis 27
by decision tree analysis 27
by histogram analysis 26
data integration
about 23
issues 23
data measuring 19, 20
data mining
about 4
feature extraction 5
process 5
statistics 15
summarization 5
Data Quality (DQ) 21
dataset
content-based features 90
link-based features 89
data smoothing
binning 22
classification 22
outlier 22
regression 22
data source
about 4
online resources 4
data transformation
about 25
aggregation 25
attribute construction 25
concept hierarchy generation, for nominal data 26
discretization 26
normalization 26
normalization methods 26
smoothing 25
DBSCAN algorithm
about 169, 170
characteristics 169
customer categorization analysis, of e-commerce 172
pseudocode 171
decision tree 70
decision tree induction
about 70-72
algorithm, pseudocode 74, 75
attribute selection measures 72
characteristics 70, 71
R implementation 75, 76
tree pruning 73
decision tree induction, attribute selection measures
Entropy 72
Gain 72
Gain Ratio 72
Gini Index 73
Information Gain 73
Split Info 73
DENCLUE algorithm
about 175, 176
density attractor 175
density function 175
gradient 175
influence function 175
pseudocode 177
R implementation 177
visitor analysis, in browser cache 177-179
DENsity-based CLUsEring algorithm. *See* **DENCLUE algorithm**
density-based cluster 170
density-based methods
about 205, 206, 207
High Contrast Subspace (HiCS) algorithm 208
intrusion detection 209
OPTICS-OF algorithm 207, 208
R implementation 209
Density Based Spatial Clustering of Applications with Noise algorithm. *See* **DBSCAN algorithm**
directed graphs 241
discrete, numeric attributes 19
discriminative frequent pattern-based classification
about 129
pseudocode 129
disjunctive normal form (DNF) 182
distance-based outlier detection

algorithm 202
divisive clustering 160
document retrieval
with SVM algorithm 128
document text
automatic abstraction, k-medoids algorithm used 152, 153
Dolphin algorithm 203

E

Eclat algorithm
about 44
R implementation 45, 46
e-commerce
customer categorization analysis 172
eigenvalues 24
eigenvectors 24
EM methods
about 111
AdaBoost algorithm 113-115
bagging algorithm 112, 113
boosting algorithm 113-115
parallel version, with MapReduce 117
Random forests algorithm 116
R implementation 116
structure 112
ensemble methods. *See* **EM methods**
Equivalence CLASS Transformation algorithm. *See* **Eclat algorithm**
escription length (DL) 105
Expectation Maximization (EM) algorithm
about 186
pseudocode 187
R implementation 187
user search intent, determining 187, 188

F

FCA-based association rule mining algorithm
R implementation 264
used, for web usage mining 264
feature extraction, examples
frequent itemsets 5
similar items 5
FindAllOutsD algorithm 201
FindAllOutsM algorithm 200
FP-Growth 31
FP-growth algorithm
about 46
data structure 47-50
input data characteristics 47-50
pseudo code 51
R implementation 51
frequent itemset 33, 34
Frequent Itemset Mining Dataset Repository
about 4
URL 4
frequent patterns
about 32
frequent itemset 32-34
frequent subsequence 32, 35
frequent substructures 32-36
frequent subgraph patterns mining algorithm
about 242
gPLS algorithm 242
GraphSig algorithm 243
gSpan algorithm 243
R implementation 246
frequent subsequence
about 35
examples 35
frequent substructures
about 35
examples 35, 36
future prices
predicting 229

G

generalized sequential patterns algorithm. *See* **GSP algorithm**
GenMax algorithm
about 52
R implementation 53
genre categorization
of web pages 258, 259
graph
about 241
directed graphs 241
undirected graphs 241
graph and network data

clustering 191
Graph-Based Sub-topic Partition Algorithm (GSPSummary) algorithm 255
graph mining
about 241
algorithms 242
GSP algorithm
features 60
R implementation 62
sequence dataset, mining 60, 61

H

hError algorithm
about 231
R implementation 232
hierarchical clustering
about 160
agglomerative clustering 160
Bayesian hierarchical clustering algorithm 164
BIRCH algorithm 161, 162
chameleon algorithm 162, 163
characteristics 160
divisive clustering 160
probabilistic hierarchical clustering algorithm 164
R implementation 165
used, for news categorization 165, 166
hierarchical clustering algorithm 210
High Contrast Subspace (HiCS) algorithm 208
high-dimensional data
clustering 188
high-performance algorithms 65
high-value credit card customers
classifying, ID3 algorithm used 76, 77, 82
HilOut algorithm
about 220
R implementation 220
horizontal format 39
hybrid association rules mining
about 58
constraint-based frequent pattern mining 58, 59
multilevel and multidimensional association rules mining 58, 59

I

ID3 algorithm
about 76
input parameters 77
output parameter 78
pseudocode 78
R implementation 79
used, for classifying high-value credit card customers 76-82
used, for web attack detection 80, 81
Information Retrieval. *See* **IR**
interval-scaled
dissimilarity 20
intrusion detection 209
Intrusion Detection System (IDS) 80
IR 11, 12
iterative classification algorithms 248
Iterative Dichotomiser 3 algorithm. *See* **ID3 algorithm**

K

k-itemset 33
k-means algorithm
about 142-211
guidelines 144
kernel k-means algorithm, pseudocode 145
k-modes algorithm 146
parallel version, with MapReduce 147
pseudocode 144
R implementation 146
search engine 142-149
shortages 143
k-medoids algorithm
about 150
case considerations 150, 151
PAM algorithm 152
R implementation 152
used, for automatic abstraction of document text 152, 153
k-Nearest Neighbors algorithm. *See* **kNN algorithm**
kNN algorithm
about 120, 121
pseudocode 121
R implementation 121

used, for protein classification 120

L

likelihood-based outlier detection algorithm
about 198
R implementation 198
Local Outlier Factor (LOF) 206
Local Reachability Density (LRD) 206

M

machine learning
statistics 15
machine learning (ML)
about 12, 16
architecture 17
batch versus online learning 17
feature selection 17
training and testing 17
training set, creating 17
machine learning (ML), classes
decision tree 16
instance-based learning 17
neural nets 17
perceptron 16
support-vector machines 17
MAFIA algorithm
about 188
customer purchase data analysis 191
pseudocode 188, 189
MapReduce
BP algorithm, parallel version 135, 136
C4.5 algorithm, parallel version 86
EM methods, parallel version 117
k-means algorithm, parallel version 147
SVM algorithm, parallel version 127, 128
market basket analysis
about 38
A-Priori algorithm 38
association rules, generating 55, 56
Charm algorithm 54
Eclat algorithm 44
FP-growth algorithm 46
GenMax algorithm 52
market basket model 38
market basket model 38
maximal frequent itemset (MFI)
about 52
mining, with GenMax algorithm 52
Maximal Marginal Relevance (MMR) algorithm
about 256
R implementation 257
Maximum Likelihood Estimation (MLE) 186
missing values
avoiding 21, 22
considerations 21
mobile fraud detection 204
multidocument summarization algorithm 255
multilevel and multidimensional association rules mining 58, 59

N

Naïve Bayes classification
characteristics 99
pseudocode 99
R implementation 100
used, for identifying spam e-mail 98-101
news categorization
with hierarchical clustering 165, 166
N-gram-based text-categorization algorithm
about 261
pseudocode 261, 262
R implementation 262
used, for categorizing newspaper articles 259, 260
used, for categorizing newswires 259, 260
NL algorithm 199
nominal attributes
dissimilarity 20
normalization methods, data transformation
min-max normalization 26
normalization by decimal scaling 26
z-score normalization 26
numeric attributes 18
numeric attributes, types
interval-scaled 19
ratio-scaled 19

O

OCSVM (One Class SVM) algorithm 212

ODIN algorithm 211
one-class nearest neighbor algorithm 213, 214
opinion mining 185
opinion-orientation algorithm 185
OPTICS algorithm
about 172, 173
core-distance of object 172, 173
pseudocode 174
reachability-distance of object 172, 173
R implementation 175
web pages, clustering 175
OPTICS-OF algorithm 207, 208
ordinal attributes
dissimilarity 20
outlier detection
about 195
brute-force algorithm 219
classification-based methods 212
clustering-based methods 210
density-based methods 205-207
HilOut algorithm 220
in high-dimensional data 219
methods 196
novelty, detecting in text 217
proximity-based methods 199
topic detection 217
with statistical method 196, 197

P

PAM algorithm 152
partition-based clustering
about 142
characteristics 142
Partitioning Around Medoids algorithm. *See* **PAM algorithm**
patterns
about 32
frequent patterns 32
PCA 24
PrefixSpan algorithm
about 238
R implementation 238
Principal-Component Analysis. *See* **PCA**
probabilistic hierarchical clustering algorithm 164
process, data mining
CRISP-DM 5, 6
SEMMA 5, 7, 8
proximity-based methods
about 199
activity monitoring 204
density-based outlier detection algorithm 199
distance-based outlier detection algorithm 199, 202
Dolphin algorithm 203
FindAllOutsD algorithm 201
FindAllOutsM algorithm 200
mobile fraud detection 204
NL algorithm 199
R implementation 204

Q

queries
boolean query 11
full document query 12
keyword query 11
natural language questions 12
phrase query 11
proximity query 12
question answering (QA) system 257, 258

R

R
about 14
advantage 14
disadvantage 14
statistics 15
visualization 28
Random forests algorithm 116
recommendation systems 181
Relative Closeness (RC), chameleon algorithm 163
Relative Interconnectivity (RI), chameleon algorithm 163
Repeated Incremental Pruning to Produce Error Reduction algorithm. *See* **RIPPER algorithm**
RHadoop 2
RIPPER algorithm
about 105

pseudocode 106
route outlier detection (ROD) algorithm
about 218
R implementation 219
rule-based classification
about 102, 104
decision tree, transforming into decision rules 103
player types, classifying in computer games 107, 108
R implementation 107
RIPPER algorithm 105, 106
sequential covering algorithm 104
rules
association rules 36, 37
correlation rules 36, 37, 38
generating, from sequential patterns 65

S

Sample, Explore, Modify, Model, Assess. *See* **SEMMA**
search engine
web page clustering 148, 149
SEMMA
about 5-8
assess 8
explore 7
model 8
modify 7
sample 7
sentential frequent itemsets
used, for text classification 130
sequence dataset
about 60
mining 59, 60
mining, with GSP algorithm 60, 61
sequence patterns
mining 237
PrefixSpan algorithm 238
sequential covering algorithm
about 104
pseudocode 104
Sequential Pattern Discovery using Equivalent classes algorithm. *See* **SPADE algorithm**
sequential patterns
rules, generating 65
shingling algorithm 246
single-pass-any-time clustering algorithm 226
singular-value decomposition. *See* **SVD**
social network
characteristics 8
collaboration networks 9
e-mail networks 9
example 10
mining 8
telephone networks 9
social networking service (SNS) 193
social network mining
about 246
community detection 246
iterative classification algorithms 248
node classification 248
R implementation 248
shingling algorithm 246
SPADE algorithm
about 63, 64
features 63
R implementation 64
spam e-mail
identifying, Naïve Bayes classification used 98-101
spectral clustering algorithm
about 159
pseudocode 159
R implementation 160
squared error-based clustering algorithm 143
STastistical Information Grid algorithm. *See* **STING algorithm**
statistical method
about 196, 197
credit card fraud detection 198
likelihood-based outlier detection algorithm 198
statistics
about 14
and R 15
data mining 15
limitations, on data mining 15
machine learning 15
STING algorithm

about 180
characteristics 180
pseudocode 180
recommendation systems 181
R implementation 181
stock market data 233
STREAM algorithm
about 224
credit card transaction flow 227
pseudocode 225
R implementation 226
stream data
mining 224
Structural Clustering Algorithm for Network (SCAN) algorithm
about 191
pseudocode 192
R implementation 192
social networking service (SNS) 193
summarization 5
Support Vector Machine algorithm. *See* **SVM algorithm**
SURFING algorithm
about 190
pseudocode 190
R implementation 191
SVD 24
SVM algorithm
about 121-123
parallel version, with MapReduce 127, 128
pseudocode 124-126
R implementation 126
used, for document retrieval 128
symbolic sequences
mining 233

T

Term Frequency-Inverse Document Frequency (TF-IDF)
about 148
text classification
with sentential frequent itemsets 130
text mining
about , 251, 11
for prediction 12
IR 11, 12
Text Retrieval Conference (TREC) 101
text summarization
about 252, 253
Maximal Marginal Relevance (MMR) algorithm 256
multidocument summarization algorithm 255, 256
topic representation 253-255
time-series data
clustering 230
clustering, with 1NN classifier algorithm 232
clustering, with hError algorithm 231
mining 227
stock market data 233
Time To Live (TTL) 98
topic detection 217
topic representation 253-255
topic signature 253
Tracking Evolving Clusters in NOisy Streams (TECNO-STREAMS) algorithm
about 233-236
R implementation 237
used, for mining web click streams 237
tree pruning
about 73
post-pruning 73
pre-pruning 73
Trojan horse 96
Trojan traffic identification
with Bayes classification 93-98

U

UCI Machine Learning Repository
about 4
URL 4
undirected graphs 241
unsupervised image categorization
with affinity propagation (AP) clustering 159
user search intent
determining 187, 188

V

vector-space model 148

vertical format 39
visitor analysis, in browser cache
bounce rate 179
browser type 179
connection speed 179
errors 179
hit 178
IP address 178
Java or Flash-enabled 179
keywords 179
new/return visitors 178
operating system version 179
page views 178
page views per visitor 178
referring pages/sites (URLs) 178
screen resolution 179
unique visitors 178
visit duration 179
visitor language 178
visitor location 178
visitor paths/navigation 179
visualization
about 27
with R 28
visualization, features
aesthetic 27
efficient 27
informative 27
novel 27

W

WAVE clustering algorithm
about 183
characteristics 184
opinion mining 185
pseudocode 184
R implementation 185
web attack
detecting, ID3 algorithm used 80, 81
DOS 81
probing 81
R2L 81
U2R 81
web click streams
mining 233, 237
web data mining
about 12, 13
web content mining 12
web structure mining 12
web usage mining 12
web data mining, tasks
information extraction (IE) 13
natural language processing (NLP) 13
question answering 13
resource discovery 13
web key resource page judgment
attributes 93
with CART algorithm 90-93
web logs
used, for web usage mining 262, 263
web mining 251
web page clustering 148, 149
web pages
clustering 175
genre categorization 258, 259
web sentiment analysist 183
web server
performance, monitoring 214
web spam
cloaking 89
content spam 88
detecting, C4.5 algorithm used 82-90
link spam 88
web usage mining
with, FCA-based association rule mining algorithm 264
with web logs 262, 263
WordNet
URL 4

Module 2

R Data Mining Blueprints

Learn about data mining with real-world datasets

Module 2: R Data Mining Blueprints

Chapter 1: Data Manipulation Using In-built R Data 7

What is data mining? 8

How is it related to data science, analytics, and statistical modeling? 11

Introduction to the R programming language 11

Getting started with R 12

Data types, vectors, arrays, and matrices 12

List management, factors, and sequences 15

Import and export of data types 16

Data type conversion 18

Sorting and merging dataframes 19

Indexing or subsetting dataframes 24

Date and time formatting 25

Creating new functions 26

User-defined functions 27

Built-in functions 27

Loop concepts – the for loop 28

Loop concepts – the repeat loop 28

Loop concepts – while conditions 29

Apply concepts 29

String manipulation 31

NA and missing value management 32

Missing value imputation techniques 32

Summary 33

Chapter 2: Exploratory Data Analysis with Automobile Data 34

Univariate data analysis 35

Bivariate analysis 41

Multivariate analysis 43

Understanding distributions and transformation 44

Normal probability distribution 44

Binomial probability distribution 45

Poisson probability distribution 46

Interpreting distributions 47

Interpreting continuous data 47

Variable binning or discretizing continuous data 50
Contingency tables, bivariate statistics, and checking for data normality 50
Hypothesis testing 56
Test of the population mean 56
One tail test of mean with known variance 56
One tail and two tail test of proportions 58
Two sample variance test 61
Non-parametric methods 64
Wilcoxon signed-rank test 64
Mann-Whitney-Wilcoxon test 65
Kruskal-Wallis test 65
Summary 66
Chapter 3: Visualize Diamond Dataset 67
Data visualization using ggplot2 71
Bar chart 83
Boxplot 84
Bubble chart 85
Donut chart 86
Geo mapping 87
Histogram 88
Line chart 90
Pie chart 90
Scatterplot 92
Stacked bar chart 97
Stem and leaf plot 98
Word cloud 99
Coxcomb plot 99
Using plotly 101
Bubble plot 102
Bar charts using plotly 103
Scatterplot using plotly 103
Boxplots using plotly 104
Polar charts using plotly 107
Polar scatterplot using plotly 107
Polar area chart 108
Creating geo mapping 109
Summary 110
Chapter 4: Regression with Automobile Data 111

Regression introduction 112
Formulation of regression problem 112
Case study 113
Linear regression 113
Stepwise regression method for variable selection 127
Logistic regression 129
Cubic regression 136
Penalized regression 137
Summary 141
Chapter 5: Market Basket Analysis with Groceries Data 142
Introduction to Market Basket Analysis 143
What is MBA? 143
Where to apply MBA? 145
Data requirement 145
Assumptions/prerequisites 147
Modeling techniques 147
Limitations 147
Practical project 148
Apriori algorithm 152
Eclat algorithm 156
Visualizing association rules 158
Implementation of arules 159
Summary 161
Chapter 6: Clustering with E-commerce Data 162
Understanding customer segmentation 163
Why understanding customer segmentation is important 163
How to perform customer segmentation? 163
Various clustering methods available 164
K-means clustering 166
Hierarchical clustering 173
Model-based clustering 179
Other cluster algorithms 180
Comparing clustering methods 184
References 184
Summary 184
Chapter 7: Building a Retail Recommendation Engine 185
What is recommendation? 186
Types of product recommendation 186

Techniques to perform recommendation 187
Assumptions 189
What method to apply when 189
Limitations of collaborative filtering 191
Practical project 192
Summary 201
Chapter 8: Dimensionality Reduction 202
Why dimensionality reduction? 203
Techniques available for dimensionality reduction 204
Which technique to apply where? 204
Principal component analysis 205
Practical project around dimensionality reduction 206
Attribute description 207
Parametric approach to dimension reduction 219
References 220
Summary 220
Chapter 9: Applying Neural Network to Healthcare Data 221
Introduction to neural networks 222
Understanding the math behind the neural network 224
Neural network implementation in R 225
Neural networks for prediction 229
Neural networks for classification 233
Neural networks for forecasting 235
Merits and demerits of neural networks 237
References 238
Summary 238
Index 239

1

Data Manipulation Using In-built R Data

The book *R Data Mining Blueprints* focuses mainly on learning methods and steps in performing data mining using the R programming language as a platform. Since R is an open source tool, learning data mining using R is very interesting for learners at all levels. The book is designed in such a way that the user can start from data management techniques, exploratory data analysis, data visualization, and modeling up to creating advanced predictive modeling such as recommendation engines, neural network models, and so on. This chapter gives an overview of the concept of data mining, its various facets with data science, analytics, statistical modeling, and visualization. This chapter gives a glimpse of programming basics using R, how to read and write data, programming notations, and syntax understanding with the help of a real-world case study. This chapter includes R scripts for practice to get hands-on experience of the concepts, terminologies, and underlying reasons for performing certain tasks. The chapter is designed in such a way that any reader with little programming knowledge should be able to execute R commands to perform various data mining tasks.

In this chapter, we will discuss in brief the meaning of data mining and its relations with other domains such as data science, analytics, and statistical modeling; apart from this, we will start the data management topics using R so that you can achieve the following objectives:

- Understanding various data types used in R, including vector and its operations
- Indexing of data frames and factors sequences
- Sorting and merging dataframes and data type conversion
- String manipulation and date object formatting
- Handling missing values and NAs and missing value imputation techniques
- Flow control, looping constructs, and the use of apply functions

What is data mining?

Data mining can be defined as the process of deciphering meaningful insights from existing databases and analyzing results for consumption by business users. Analyzing data from various sources and summarizing it into meaningful information and insights is that part of statistical knowledge discovery that helps not only business users but also multiple communities such as statistical analysts, consultants, and data scientists. Most of the time, the knowledge discovery process from databases is unexpected and the results can be interpreted in many ways.

The growing number of devices, tablets, smartphones, computers, sensors, and various other digital devices helps in generating and collecting data at a much faster rate than ever before. With the ability of modern-day computers, the increased data can be preprocessed and modeled to answer various questions related to any business decision-making process. Data mining can also be defined as a knowledge-intensive search across discrete databases and information repositories using statistical methodologies, machine learning techniques, visualization, and pattern recognition technologies.

The growth of structured and unstructured data, such as the existence of bar codes in all products in a retail store, attachment of RFID-based tags on all assets in a manufacturing plant, Twitter feeds, Facebook posts, integrated sensors across a city to monitor the changing weather conditions, video analysis, video recommendation based on viewership statistics, and so on creates a conducive ecosystem for various tools, technologies, and methodologies to splurge. Data mining techniques applied to the variety of data discussed previously not only provide meaningful information about the data structure but also recommend possible future actions to be taken by businesses.

Figure 1: Data Mining – a multi-disciplinary subject

The process of data mining involves various steps:

1. Extract the required data from databases and data warehouses.
2. Perform a sanity check on the data to remove redundant characters and irrelevant information.
3. At times, it is important to combine information from various other disjoint databases. Hence, look for common attributes to combine databases.
4. Apply data transformation techniques. Sometimes, it is required to include a few attributes and features in a model.
5. Pattern recognition among the input features, where any of the pattern recognition methods can be applied.

6. Knowledge representation. This includes representation of knowledge mined from the databases in a visual form to various business stakeholders.

Figure 2: A typical data mining process flow

Having discussed the process flow of data mining and the core components, it is also important to look at a few challenges that one may encounter in data mining, such as computational efficiency, unstructured databases and their confluence with structured databases, high-dimensional data visualization, and so on. These issues can be resolved using innovative approaches. In this book, we are going to touch upon a few solutions while performing practical activities on our projects.

How is it related to data science, analytics, and statistical modeling?

Data science is a broader topic under which the data mining concept resides. Going by the aforementioned definition of data mining, it is a process of identifying patterns hidden in data and some interesting correlations that can provide useful insights. Data mining is a subset in data science projects that involves techniques such as pattern recognition, feature selection, clustering, supervised classification, and so on. Analytics and statistical modeling involve a wide range of predictive models-classification-based models to be applied on datasets to solve real-world business problems. There is a clear overlap between the three terminologies – data science, analytics, statistical modeling, and data mining. The three terminologies should not be looked at in isolation. Depending upon the project requirements and the kind of business problem, the overlap position might change, but at a broad level, all the concepts are well associated. The process of data mining also includes statistical and machine learning-based methods to extract data and automate rules and also represent data using good visualizations.

Introduction to the R programming language

In this chapter, we are going to start with basic programming using R for data management and data manipulation; we are also going to cover a few programming tips. R can be downloaded from`https://cran.r-project.org/`. Based on the operating system, anyone can download and install R binary files on their systems. The R programming, language which is an extension of the S language, is a statistical computing platform. It provides advanced predictive modeling capability, machine learning algorithms implementation capability, and better graphical visualization. R has various other plugins such as R.Net, rJava, SparkR, and RHadoop, which increases the usability of R in big data scenarios. The user can integrate R scripts with other programming platforms. Detailed information about R can be accessed using the following link:

`https://cran.r-project.org/`.

```
R version 3.2.1 (2015-06-18) -- "World-Famous Astronaut"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Getting started with R

The opening message when starting R would be as shown in the preceding screenshot. Everything entered in the R console is an object, many objects created in an active R session have different attributes associated with them, and one common attribute associated with an object is called its class. There are two popular approaches to object-oriented programming in R, which are S3 classes and S4 classes. The basic difference between the S3 and S4 approaches is that the former is a more flexible approach; however, the latter is a more structured approach to object-oriented programming. Both S3 and S4 approaches recognize any symbol, character, and number as an object in R sessions and provide functionality where the object can be used for further computations.

Data types, vectors, arrays, and matrices

There are two broad sets of data types: atomic vectors and compound vectors. There are basically five data types in R programming under the atomic vector category: numeric or numbers, characters or strings, factors, logical, and complex. And there are four compound data types: data frame, lists, array, and matrix. The primary data object in R is a vector; even when we assign a single-digit number to any alphabet, it is a single element vector in R. All data objects contain a mode and a length. The mode determines the kind of data stored in the object, and the length determines the number of elements contained in that object. The `c()` function in R implies concatenating of various elements in a vector.

Let's take a look at various examples showing different data types in R:

```
> x1<-c(2.5,1.4,6.3,4.6,9.0)
> class(x1)
[1] "numeric"
> mode(x1)
[1] "numeric"
> length(x1)
[1] 5
```

In the preceding script, the `x1` vector is a numeric vector and the number of elements is `5`. Both `class()` and `mode()` return the same results, hence both determine the type of vector:

```
> x2<-c(TRUE,FALSE,TRUE,FALSE,FALSE)
> class(x2)
[1] "logical"
> mode(x2)
[1] "logical"
> length(x2)
[1] 5
```

The `x2` vector is a logical vector having five elements. The logical vector elements or values can be written as either `T/F` or `TRUE/FALSE`.

```
    > x3<-
c("DataMining","Statistics","Analytics","Projects","MachineLearning")
    > class(x3)
    [1] "character"
    > length(x3)
    [1] 5
```

Object `x3` represents a character vector of length 25. All elements of the vector can be mentioned within double quote (`" "`) or single quote (`' '`).

The factor is another form of data where various categories listed in the vector are known as levels, in the preceding example; vector `a` is a character vector with two levels or categories, which are repeated with some frequency. The `as.factor()` command is used to convert a character vector into a factor data type. After applying that, it indicates there are five levels such as `Analytics`, `DataMining`, `MachineLearning`, `Projects`, and `Statistics`. The `table()` command indicates the frequency table computed from the factor variable:

```
> x<-data.frame(x1,x2,x3)
> class(x)
[1] "data.frame"
> print(x)
  x1    x2              x3
1 12  TRUE       Analytics
```

```
2 13 FALSE      DataMining
3 24  TRUE MachineLearning
4 54 FALSE        Projects
5 29  TRUE      Statistics
```

Dataframes are another popular form of data type in the R programming language that include all different data types. A dataframe is a list that contains multiple vectors of the same length and different types of data. If you simply import a dataset from a spreadsheet, the data type by default becomes dataframe. Later on, the data type for individual variables can be changed. So, dataframe can be defined as a matrix that contains columns of different data types. In the preceding script, the dataframe `x` contains three different data types: numeric, logical, and character. Most real-world datasets contain different data types; for example, in a retail store, information about customers is stored in a database. This includes customer ID, purchase date, amount purchased, whether part of any loyalty program or not, and so on.

One important point about vectors: all elements of a vector should be of the same type. If not, R will forcibly convert that by coercion. For example, in a numeric vector, if one element contains a character value, the vector type will change from numeric to character. The script is given as follows:

```
> x1<-c(2.5,1.4,6.3,4.6,9.0)
> class(x1)
[1] "numeric"
> x1<-c(2.5,1.4,6.3,4.6,9.0,"cat")
> class(x1)
[1] "character"
```

R is case sensitive, so `"cat"` is different from `"Cat"`. Hence, please be careful while assigning object names to the vectors. At times, it would be difficult to remember the object names:

```
> ls()
 [1] "a"              "centers"        "df"             "distances"
 [5] "dt2"            "i"              "indexes"        "km"
 [9] "kmd"            "kmeans.results" "log_model"      "mtcars"
[13] "outliers"       "pred"           "predict.kmeans" "probs"
[17] "Smarket"        "start"          "sumsq"          "t"
[21] "test"           "Titanic"        "train"          "x"
[25] "x1"             "x2"             "x3"             "x4"
[29] "x5"             "y"              "z"
```

To know what all objects are active in the current R session, the `ls()` command can be used; the `list` command will print all the active objects in the current session. Let's take a look at what a list is, how to retrieve the elements of a list, and how the list function can be used.

List management, factors, and sequences

A list is an ordered collection of objects that can contain arbitrary objects. Elements of a list can be accessed using the double square bracket. Those collections of objects are not necessarily of the same type. They can be of different types:

```
> mylist<-list(custid=112233, custname="John R", mobile="989-101-1011",
+ email="JohnR@gmail.com")
> mylist
$custid
[1] 112233
$custname
[1] "John R"
$mobile
[1] "989-101-1011"
$email
[1] "JohnR@gmail.com"
```

In the preceding example, the customer ID and mobile number are of numeric data type; however, the customer name and e-mail ID are of character data type. There are basically four elements in the preceding list. To extract elements from a list, we use double square brackets, and if we need to extract only a sublist from the list, we can use a single square bracket:

```
> mylist[[2]]
[1] "John R"
> mylist[2]
$custname
[1] "John R"
```

The next thing related to lists is how to combine more than one list. Lists can be combined using the `cbind()` function, that is, the column bind function:

```
    > mylist1<-list(custid=112233, custname="John R",
mobile="989-101-1011",
    + email="JohnR@gmail.com")
    > mylist2<-list(custid=443322, custname="Frank S",
mobile="781-101-6211",
    + email="SFranks@hotmail.com")
    > mylist<-cbind(mylist1,mylist2)
```

```
> mylist
         mylist1           mylist2
custid   112233            443322
custname "John R"          "Frank S"
mobile   "989-101-1011"    "781-101-6211"
email    "JohnR@gmail.com" "SFranks@hotmail.com"
```

Factors can be defined as various levels occurring in a categorical or nominal variable with a certain frequency. In other words, levels that are repetitive in a categorical variable are known as factors. In the following sample script, a character vector "domains" contains many levels; using the `factor` command, the frequency for each level can be estimated.

Sequences are repeated number of iterations, either numerical values or categorical or nominal values that can be part of a dataset. Numeric sequences can be created using a colon operator. To generate sequences using factor variables, the `gl()` function can be used. This is a very useful function while computing quantiles and graphical functions. Also, there are various other possible scenarios where you can use the function:

```
> seq(from=1,to=5,length=4)
[1] 1.000000 2.333333 3.666667 5.000000
> seq(length=10,from=-2,by=.2)
 [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
> rep(15,10)
 [1] 15 15 15 15 15 15 15 15 15 15
> gl(2,5,labels=c('Buy','DontBuy'))
 [1] Buy     Buy     Buy     Buy     Buy     DontBuy DontBuy DontBuy
DontBuy
[10] DontBuy
Levels: Buy DontBuy
```

The first line of code generates the sequence in ascending order, the second line creates a reverse sequence, and the last line creates a sequence for the factor data type.

Import and export of data types

If the Windows directory path is set, to import a file into the R system, it is not required to write the complete path where the file resides. If the Windows directory path is set to some other location in your system and still you want to access the file, then the complete path needs to be given to read the file:

```
> getwd()
[1] "C:/Users/Documents"
> setwd("C:/Users/Documents")
```

Any file in the documents folder can be read without mentioning the detailed path. Hence it is always suggested to change the Windows directory to the folder where the file resides.

There are different file formats; among them, CSV or text format is the best for the R programming platform. However, we can import from other file formats:

```
> dt<-read.csv("E:/Datasets/hs0.csv")
> names(dt)
 [1] "X0"      "X70"     "X4"      "X1"      "X1.1"    "general" "X57"
 [8] "X52"     "X41"     "X47"     "X57.1"
```

If you are using the `read.csv` command, there is no need to write the header `True` and separator as comma, but if you are using the `read.table` command, it is mandatory to use. Otherwise, it will read the first variable from the dataset:

```
> data<- read.table("E:/Datasets/hs0.csv",header=T,sep=",")
> names(data)
 [1] "X0"      "X70"     "X4"      "X1"      "X1.1"    "general" "X57"
 [8] "X52"     "X41"     "X47"     "X57.1"
```

While mentioning paths to extract the files, you can use either `/` or `\\`; both ways will work. In real-life projects, typically data is stored in Excel format. How to read data from Excel format is a challenge. It is not always convenient to store data in CSV format and then import. The following script shows how we can import Excel files in R. Two additional libraries are required to import an RDBMS file such as Excel. Those libraries are mentioned in the script and the sample data snippets are given as well:

```
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars
> library(xlsxjars)
> dat<-read.xlsx("E:/Datasets/hs0.xls","hs0")
> head(dat)
  gender  id race ses schtyp  prgtype read write math science socst
1      0  70    4   1      1  general   57    52   41      47    57
2      1 121    4   2      1   vocati   68    59   53      63    61
3      0  86    4   3      1  general   44    33   54      58    31
4      0 141    4   3      1   vocati   63    44   47      53    56
5      0 172    4   2      1 academic   47    52   57      53    61
6      0 113    4   2      1 academic   44    52   51      63    61
```

Importing data from SPSS files is explained as follows. Legacy enterprise-based software systems generate data in either SPSS format or SAS format. The syntax for importing data from SPSS and SAS files needs additional packages or libraries. To import SPSS files, the `Hmisc` package is used, and to import SAS files, the `sas7bdat` library is used:

```
> library(Hmisc)
> mydata <- spss.get("E:/Datasets/wage.sav", use.value.labels=TRUE)
> head(mydata)
   HRS  RATE ERSP ERNO NEIN ASSET  AGE   DEP RACE SCHOOL
1 2157 2.905 1121  291  380  7250 38.5 2.340 32.1   10.5
2 2174 2.970 1128  301  398  7744 39.3 2.335 31.2   10.5
3 2062 2.350 1214  326  185  3068 40.1 2.851   NA    8.9
4 2111 2.511 1203   49  117  1632 22.4 1.159 27.5   11.5
5 2134 2.791 1013  594  730 12710 57.7 1.229 32.5    8.8
6 2185 3.040 1135  287  382  7706 38.6 2.602 31.4   10.7
> library(sas7bdat)
> mydata <- read.sas7bdat("E:/Datasets/sales.sas7bdat")
> head(mydata)
  YEAR NET_SALES PROFIT
1 1990       900    123
2 1991       800    400
3 1992       700    300
4 1993       455     56
5 1994       799    299
6 1995       666    199
```

Exporting a dataset from R to any external location can be done by changing the read command to the write command and changing the directory path where you want to store the file.

Data type conversion

There are various types of data such as numeric, factor, character, logical, and so on. Changing one data type to another if the formatting is not done properly is not difficult at all using R. Before changing the variable type, it is essential to look at the data type it currently is. To do that, the following command can be used:

```
> is.numeric(x1)
[1] TRUE
> is.character(x3)
[1] TRUE
> is.vector(x1)
[1] TRUE
> is.matrix(x)
[1] FALSE
> is.data.frame(x)
[1] TRUE
```

When we are checking a numeric variable as numeric or not, the resultant output will display `TRUE` or `FALSE`. The same holds for other data types as well. If any data type is not right, that can be changed by the following script:

```
> as.numeric(x1)
[1] 2.5 1.4 6.3 4.6 9.0
> as.vector(x2)
[1]  TRUE FALSE  TRUE FALSE FALSE
> as.matrix(x)
     x1    x2      x3                x4  x5
[1,] "2.5" " TRUE" "DataMining"      "1" "1+ 0i"
[2,] "1.4" "FALSE" "Statistics"      "2" "6+ 5i"
[3,] "6.3" " TRUE" "Analytics"       "3" "2+ 2i"
[4,] "4.6" "FALSE" "Projects"        "4" "4+ 1i"
[5,] "9.0" "FALSE" "MachineLearning" "5" "6+55i"
> as.data.frame(x)
   x1    x2              x3 x4    x5
1 2.5  TRUE      DataMining  1 1+ 0i
2 1.4 FALSE      Statistics  2 6+ 5i
3 6.3  TRUE       Analytics  3 2+ 2i
4 4.6 FALSE        Projects  4 4+ 1i
5 9.0 FALSE MachineLearning  5 6+55i
> as.character(x2)
[1] "TRUE"  "FALSE" "TRUE"  "FALSE" "FALSE"
```

When using `as.character()`, even a logical vector is changed from logical to character vector. For a numeric variable, nothing is changed as the `x1` variable was already in numeric form. A logical vector can also be changed from logical to factor using this command:

```
> as.factor(x2)
[1] TRUE  FALSE TRUE  FALSE FALSE
Levels: FALSE TRUE
```

Sorting and merging dataframes

Sorting and merging are two importantconcepts in data management. The object can be a single vector or it can be a data frame or matrix. To sort a vector in R, the `sort()` command is used. An decreasing order option can be used to change the order to ascending or descending. For a data frame such as `ArtPiece.csv`, the order command is used to sort the data, where ascending or descending order can be set for multiple variables. Descending order can be executed by putting a negative sign in front of a variable name. Let's use the dataset to explain the concept of sorting in R as shown in the following script:

```
> # Sorting and Merging Data
> ArtPiece<-read.csv("ArtPiece.csv")
> names(ArtPiece)
 [1] "Cid"                          "Critic.Ratings"
"Acq.Cost"
 [4] "Art.Category"                 "Art.Piece.Size"
"Border.of.art.piece"
 [7] "Art.Type"                     "Prominent.Color"
"CurrentAuctionAveragePrice"
[10] "Brush"                        "Brush.Size"
"Brush.Finesse"
[13] "Art.Nationality"              "Top.3.artists"
"CollectorsAverageprice"
[16] "Min.Guarantee.Cost"
> attach(ArtPiece))
```

In the `ArtPiece` dataset, there are 16 variables: 10 numeric variables, and 6 categorical variables. Using the `names` command, all the names of the variables in the dataset can be printed. The `attach` function helps in keeping all the variable names in the current session of R, so that every time the user will not have to type the dataset name before the variable name in the code:

```
> sort(Critic.Ratings)
 [1] 4.9921 5.0227 5.2106 5.2774 5.4586 5.5711 5.6300 5.7723 5.9789
5.9858 6.5078 6.5328
[13] 6.5393 6.5403 6.5617 6.5663 6.5805 6.5925 6.6536 6.8990 6.9367
7.1254 7.2132 7.2191
[25] 7.3291 7.3807 7.4722 7.5156 7.5419 7.6173 7.6304 7.6586 7.7694
7.8241 7.8434 7.9315
[37] 7.9576 8.0064 8.0080 8.0736 8.0949 8.1054 8.2944 8.4498 8.4872
8.6889 8.8958 8.9046
[49] 9.3593 9.8130
```

By default, the sorting is done by ascending order. To sort the vector based on descending order, it is required to put a negative sing before the name of the variable. The `Critic.Ratings` variable can also be sorted based on descending order, which is shown as follows. To sort in descending order, `decreasing` is true and it can be set within the command:

```
> sort(Critic.Ratings, decreasing = T)
 [1] 9.8130 9.3593 8.9046 8.8958 8.6889 8.4872 8.4498 8.2944 8.1054
8.0949 8.0736 8.0080
[13] 8.0064 7.9576 7.9315 7.8434 7.8241 7.7694 7.6586 7.6304 7.6173
7.5419 7.5156 7.4722
[25] 7.3807 7.3291 7.2191 7.2132 7.1254 6.9367 6.8990 6.6536 6.5925
6.5805 6.5663 6.5617
[37] 6.5403 6.5393  6.5328 6.5078 5.9858 5.9789 5.7723 5.6300 5.5711
```

```
5.4586 5.2774 5.2106
    [49] 5.0227 4.9921
```

Instead of sorting a single numeric vector, most of the times, it is required to sort a dataset based on some input variables or attributes present in the dataframe. Sorting a single variable is quite different from sorting a dataframe. The following script shows how a dataframe can be sorted using the `order` function:

```
> i2<-ArtPiece[order(Critic.Ratings,Acq.Cost),1:5]
> head(i2)
   Cid Critic.Ratings Acq.Cost          Art.Category Art.Piece.Size
9    9         4.9921    39200             Vintage I  26in. X 18in.
50  50         5.0227    52500        Portrait Art I  26in. X 24in.
26  26         5.2106    31500           Dark Art II    1in. X 7in.
45  45         5.2774    79345             Gothic II   9in. X 29in.
21  21         5.4586    33600  Abstract Art Type II  29in. X 29in.
38  38         5.5711    35700 Abstract Art Type III   9in. X 12in.
```

The preceding code shows critic ratings and acquisition cost sorted in ascending order. The `order` command is used instead of the `sort` command. The `head` command prints the first six observations by default from the sorted dataset. The number `1:5` in the second argument after the `order` command implies that we want to print the first six observations and five variables from the `ArtPiece` dataset. If it is required to print 10 observations from the beginning, `head(i2, 10)` can be executed. The dataset does not have any missing values; however, the existence of NA or missing values cannot be ruled out from any practical dataset. Hence, with the presence of NA or missing values, sorting a data frame can be tricky. So by including some arbitrary NA values in the dataset, the order command produced the following result:

```
> i2<-ArtPiece[order(Border.of.art.piece, na.last = F),2:6]
> head(i2)
   Critic.Ratings Acq.Cost         Art.Category Art.Piece.Size
Border.of.art.piece
    18         7.5156    34300          Vintage III   29in. X 6in.
    43         6.8990    59500 Abstract Art Type II  23in. X 21in.
    1          8.9046    49700  Abstract Art Type I  17in. X 27in.
Border 1
    12         7.5419    37100        Silhoutte III   28in. X 9in.
Border 10
    14         7.1254    54600           Vintage II   9in. X 12in.
Border 11
    16         7.2132    23100           Dark Art I  10in. X 22in.
Border 11
```

The `NA.LAST` command is used to separate the missing values and NAs from the dataset. They can be either placed at the bottom of the dataset using `NA.LAST` is TRUE or at the

beginning of the data set as `NA.LAST` is FALSE. By keeping NAs separate from the order function, data sanity can be maintained.

The `merge` function helps in combining two data frames. To combine two data frames, at least one column name should be identical. The two data frames can also be joined by using a function called column bind. To display the difference between the column bind and `merge` functions, we have taken `audit.CSV` dataset. There are two small datasets, A and B, prepared out of the `audit` dataset:

```
> A<-audit[,c(1,2,3,7,9)]
> names(A)
[1] "ID"         "Age"        "Employment" "Income"     "Deductions"
> B<-audit[,c(1,3,4,5,6)]
> names(B)
[1] "ID"         "Employment" "Education"  "Marital"    "Occupation"
```

Two columns, `ID` and `Employment`, are common in both datasets `A` and `B`, which can be used as a primary key for merging two data frames. Using the `merge` command, the common columns are taken once in the `result` dataset from the `merge` function. The merged data frame contains all the rows that have complete entries in both the data frames:

```
    > head(merge(A,B),3)
           ID Employment Age   Income Deductions Education   Marital
Occupation
    1 1004641    Private  38  81838.0          0   College Unmarried
Service
    2 1010229    Private  35  72099.0          0 Associate    Absent
Transport
    3 1024587    Private  32 154676.7          0    HSgrad  Divorced
Clerical
```

The `merge` function allows four different ways of combining data: natural join, full outer join, left outer join, and right outer join. Apart from these joins, two data frames can also be merged using any specific column or multiple columns. Natural join helps to keep only rows that match from the data frames, which can be specified by the `all=F` argument:

```
    > head(merge(A,B, all=F),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

Full outer join allows us to keep all rows from both data frames, and this can be specified by the `all=T` command. This performs the complete merge and fills the columns with NA values where there is no matching data in both the data frames. The default setting of the `merge` function drops all unmatched cases from both the data frames; to keep all the cases in new data frame, we need to specify `all=T`:

```
    > head(merge(A,B, all=T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1004641    Private  38  81838.0          0      <NA>    <NA>
<NA>
    2 1010229    Private  35  72099.0          0      <NA>    <NA>
<NA>
    3 1024587    Private  32 154676.7          0      <NA>    <NA>
<NA>
```

Left outer join allows us to include all the rows of data frame one (`A`) and only those from data frame two (`B`) that match; to perform this, we need to specify `all.x=T`:

```
    > head(merge(A,B, all.x = T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1004641    Private  38  81838.0          0      <NA>    <NA>
<NA>
    2 1010229    Private  35  72099.0          0      <NA>    <NA>
<NA>
    3 1024587    Private  32 154676.7          0      <NA>    <NA>
<NA>
```

Right outer join allows us to include all the rows of data frame `B` and only those from `A` that match; specify `all.y=T`:

```
    > head(merge(A,B, all.y = T),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

In data frames `A` and `B`, two columns are common; they are `ID` and `Employment`. Using the `merge` command, if we select by one common variable, the other common variable would appear on the result data frame. If we select multiple data frames as the criteria to merge, then all duplicate columns from the resulting data frame will disappear:

```
    > head(merge(A,B,by="ID"),3)
           ID Age Employment.x   Income Deductions Employment.y Education
Marital Occupation
    1 1044221  60      Private  7568.23          0      Private   College
Married  Executive
    2 1047095  74      Private 33144.40          0      Private    HSgrad
Married    Service
    3 1047698  43      Private 43391.17          0      Private  Bachelor
Married  Executive
    > head(merge(A,B,by=c("ID","Employment")),3)
           ID Employment Age   Income Deductions Education Marital
Occupation
    1 1044221    Private  60  7568.23          0   College Married
Executive
    2 1047095    Private  74 33144.40          0    HSgrad Married
Service
    3 1047698    Private  43 43391.17          0  Bachelor Married
Executive
```

The `merge` function works when two data frames contain at least one common column, if both the data frames contain disjoint columns or there is no common column between the two data frames, then to combine both data frames, the column bind function can be used. The column bind function prints all the columns in data frame `A` and data frame `B` and puts them side by side:

```
> A<-audit[,c(2,7,9)]
> names(A)
[1] "Age"        "Income"     "Deductions"
> B<-audit[,c(4,5,6)]
> names(B)
[1] "Education"  "Marital"    "Occupation"
> head(cbind(A,B),3)
  Age   Income Deductions Education   Marital Occupation
1  38  81838.0          0   College Unmarried    Service
2  35  72099.0          0 Associate    Absent  Transport
3  32 154676.7          0    HSgrad  Divorced   Clerical
```

Indexing or subsetting dataframes

While working on a client dataset with a large number of observations, it is required to subset the data based on some selection criteria and with or without replacement-based sampling. Indexing is the process of extracting the subset of data from the dataframe based on some logical conditions. The `subset` function helps in extracting elements from the data frame like indexing:

```
> newdata <- audit[ which(audit$Gender=="Female" & audit$Age > 65), ]
> rownames(newdata)
 [1] "49"   "537"  "552"  "561"  "586"  "590"  "899"  "1200" "1598"
"1719"
```

The preceding code explains: select those observations from the `audit` dataset where the gender is female and the age is more than 65 years. Which command is used to select that subset of data `audit` based on the preceding two criteria? There are 10 observations satisfying the preceding condition; the row numbers of the data frame are printed previously. A similar result can be obtained by using the `subset` function as well. Instead of the `which` function, the `subset` function should be used, as the latter is more efficient in passing multiple conditions. Let's take a look at the way the `subset` function is used:

```
> newdata <- subset(audit, Gender=="Female" & Age > 65,
select=Employment:Income)
> rownames(newdata)
 [1] "49"   "537"  "552"  "561"  "586"  "590"  "899"  "1200" "1598"
"1719"
```

The additional argument in the `subset` function makes the function more efficient as it provides the additional benefit of selecting specific columns from the dataframe where the logical condition is satisfied.

Date and time formatting

The date functions return a `Date` class that represents the number of days since January 1, 1970. The `as.numeric()` function can be used to create a numeric variable with the number of days since 1/1/1970. The return value of `as.Date()` is a `Date` class object:

```
> Sys.time()
[1] "2015-11-10 00:43:22 IST"
> dt<-as.Date(Sys.time())
> class(dt)
[1] "Date"
```

The system time function captures the date and time with the time zone. When we convert the system time using the `as.Date` function and stored as a new object in R, we find that the class of that object is `Date`. The `weekdays` function returns the name of the day such as `"Monday"` or `"Wednesday"`. The `months` function returns the name of the month from the `date` variable. The `quarters` function returns the name of the quarter for the date object and year value also can be extracted using the `substr()` command:

```
> weekdays(as.Date(Sys.time()))
```

```
[1] "Monday"
> months(as.Date(Sys.time()))
[1] "November"
> quarters(as.Date(Sys.time()))
[1] "Q4"
> substr(as.POSIXct(as.Date(Sys.time())),1,4)
[1] "2015"
```

If the date variable given in the dataset is not in proper format for further computations, it can be formatted using the `format` function:

```
> format(Sys.time(),format = "%m %d %y")
[1] "11 10 15"
```

There are various options that can be passed to the format argument based on the user requirement:

Option	What it does
`#%d`	Means day as a number from (`0-31`) `01-31`
`#%a`	Means abbreviated weekday as Mon
`#%`	A means unabbreviated weekday, Monday
`#%m`	Month (`00-12`)
`#%b`	Abbreviated month
`#%B`	Unabbreviated month January
`#%y`	Two-digit year (`13`)
`#%Y`	Four-digit year (`2013`)

Table 1: Formatting date options

Practical datasets contain date fields such as the transaction date in retail, visit date in healthcare, and processing date in BFSI; and any time series data contains at least one time element. To include the date variable in any statistical model, data transformation is required, such as calculating the vintage of a customer in a retail scenario. The data transformation can be done using the aforementioned options.

Creating new functions

There are two different types of functions in R, user-defined functions and built-in functions.

User-defined functions

A user-defined function provides customization and flexibility to users to write their functions to perform computations. It has a general notation shown as follows:

```
newFunc <- function(x){define function}
> int<-seq(1:20)
> int
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> myfunc<-function(x){x*x}
> myfunc(int)
 [1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256
289 324
 361 400
```

In the preceding script, we are creating a small sequence of numbers from 1 to 20 and a user-defined function to calculate the square of each integer. Using the new function, we can calculate the square of any number. So the user can define and create his or her own custom function.

Built-in functions

Built-in functions, such as mean, median, standard deviation, and so on, provide the user the ability to compute basic statistics using R. There are many built-in functions; the following table displays a few important built-in functions:

Function	**Description**
`abs(x)`	Absolute value
`sqrt(x)`	Square root
`ceiling(x)`	Rounding up the number
`floor(x)`	Rounding down the number
`trunc(x)`	`trunc(5.99)` is `5`
`round(x, digits=n)`	`round(3.475, digits=2)` is `3.48`
`signif(x, digits=n)`	`signif(3.475, digits=2)` is `3.5`

`cos(x),sin(x),tan(x)`	Also `acos(x),cosh(x),acosh(x)`, and so on
`log(x)`	Natural logarithm
`log10(x)`	Common logarithm
`exp(x)`	`e^x`

Table 2: Some built-in functions

Loop concepts – the for loop

The `for` loop is the most popular looping construct in R. Using a `for` loop, a similar task can be performed many times iteratively, let's look at a sample example where the `for` loop concept is applied. In the following code, a series of numbers from 10 to 25 is created. The null vector `v` is acting like a storage unit. If the condition mentioned in the following code is not met, the loop is never executed:

```
x<-100:200
y <- NULL # NULL vector as placeholder
for(i in seq(along=x)) {
   if(x[i] < 150) {
   y <- c(y, x[i] - 50)
   } else {
   y <- c(y, x[i] + 50)
   }
   }
print(y)
```

Loop concepts – the repeat loop

The `repeat` loop is used to iterate a certain calculation over a vector or dataframe. There is no provision to check the condition to exit the loop; generally a `break` statement is used to exit the loop. If you fail to provide any break condition within the `repeat` loop, you will end up running the `repeat` loop infinitely. Let's look at the code showing how to write a `repeat` loop. The `break` condition used in the following code is if `x > 2.6`:

```
x <- 100
repeat {
  print(x)
  x = sqrt(x)+10
  if (x > 2.6){
    break
```

```
    }
  }
```

Loop concepts – while conditions

The structure of the `while` loop in R is simple; it starts with a desired result that the user wants to see from the experiment. As the condition is entered in the beginning, the body of the loop will start iteration and go on as long the condition is being met. The skeleton structure of a `while` loop consists of a constraint condition to start with; here is an example:

```
x <- 10
while (x < 60) {
  print(x)
  x = x+10
}
```

If we compare different types of loops in the R programming language, `for` loop and `while` loop are very frequently used; `repeat` loop is not used that frequently because of the time it takes to complete the run. If we compare the loops with the apply group of functions, the latter set of functions is quite effective in handling different tasks in R. Let's look at the apply group of functions.

Apply concepts

The `apply` function uses an array, a matrix, or a dataframe as an input and returns the result in an array format. The calculation or operation is defined by the user's custom function or using any built-in functions. The margin argument is used to specify which margin we want to apply to the function and which margin we wish to keep. If the array we are using is a matrix, then we can specify the margin to be either `1` (apply the function to the rows) or `2` (apply the function to the columns). The function can be any function such as `mean`, `median`, `standard deviation`, `variance`, and so on that is built in or user defined. Here we are going to use iris dataset to perform the task:

```
> apply(ArtPiece[,2:3],2,mean)
Critic.Ratings       Acq.Cost
      7.200416   44440.900000
> apply(ArtPiece[,2:3],1,mean)
 [1] 24854.45 26604.68 17153.69 14353.28 14003.47 19604.05 14703.27
15753.29 19602.50
[10] 26954.24 19254.00 18553.77 18903.97 27303.56 24153.74 11553.61
23804.04 17153.76
```

```
    [19] 19953.30 24854.22 16802.73 20303.33 14354.91 26952.99 24503.28
15752.61 28004.45
    [28] 30803.81 29403.27 19604.00 29053.88 17152.81 33253.91 24502.89
37453.92 12604.15
    [37] 21353.82 17852.79 28703.83 29753.25 23453.27 18204.34 29753.45
27654.05 39675.14
    [46] 24853.61 16102.99 13653.98 14353.66 26252.51
```

The `lapply` function is useful when dealing with (applying any function) dataframes. In R, the dataframe is considered a list and the variables in the dataframe are the elements of the list. We can, therefore, apply a function to all the variables in a dataframe using the `lapply` function:

```
> lapply(ArtPiece[,2:3],mean)
$Critic.Ratings
[1] 7.200416
$Acq.Cost
[1] 44440.9
```

The `sapply` function applies to elements in a list and returns the results in a vector, matrix, or list. When the argument is `simplify=F`, then the `sapply` function returns the results in a list just like the `lapply` function. However, when the argument is `simplify=T`, which is the default argument, the `sapply` function returns the results in a simplified form if at all possible:

```
> sapply(ArtPiece[,2:3],mean)
Critic.Ratings       Acq.Cost
      7.200416   44440.900000
```

When we want to apply a function to subsets of a vector and the subsets are defined by some other vector, usually a factor. The output from `tapply` is a matrix/array, where an element in the matrix/array is the value of `f` at a grouping `g` of the vector, and `g` gets pushed to the row/col names:

```
> head(tapply(Critic.Ratings,Acq.Cost,summary),3)
$`23100`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.213   7.213   7.213   7.213   7.213   7.213
$`25200`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  8.294   8.294   8.294   8.294   8.294   8.294
$`27300`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.958   7.958   7.958   7.958   7.958   7.958
```

There are other functions in the `apply` family of functions, such as:

- `eapply`: Apply a function over values in an environment
- `mapply`: Apply a function to multiple list or vector arguments
- `rapply`: Recursively apply a function to a list

String manipulation

String manipulation or character manipulation is an important aspect of any data management system. In a typical real-world dataset, names of customers for example are written in different ways, such as J H Smith, John h Smith, John h smith, and so on. Upon verifying, it is observed that all three names belong to the same person. In typical data management, it is important to standardize the text columns or variables in a dataset because R is case sensitive and it reads any discrepancy as a new data point. There can be many other variables such as the name/model of a vehicle, product description, and so on. Let's look how the text can be standardized using some functions:

```
    > x<-"data Mining is not a difficult subject, anyone can master the
subject"
    > class(x)
    [1] "character"
    > substr(x, 1, 12)
    [1] "data Mining "
```

The object `X` in the preceding script is a string or character object. The `substr` command is used to pull a `sub` string from the string with the position defined in the function. If certain patterns or texts need to be altered or changed, then the `sub` command can be used. There are four important arguments that the user needs to pass: the string in which a pattern needs to be searched, the pattern, the modified pattern that needs to be replaced, and whether case sensitivity is acceptable or not. Let's look at a sample script:

```
    > sub("data mining", "The Data Mining", x, ignore.case =T, fixed=FALSE)
    [1] "The Data Mining is not a difficult subject, anyone can master the
subject"
    > strsplit(x, "")
    [[1]]
     [1] "d" "a" "t" "a" " " "M" "i" "n" "i" "n" "g" " " "i" "s" " " "n"
"o" "t" " " "a" " "
    [22] "d" "i" "f" "f" "i" "c" "u" "l" "t" " " "s" "u" "b" "j" "e" "c"
"t" "," " " "a" "n"
    [43] "y" "o" "n" "e" " " "c" "a" "n" " " "m" "a" "s" "t" "e" "r" " "
"t" "h" "e" " " "s"
    [64] "u" "b" "j" "e" "c" "t"
```

The `strsplit` function helps in expanding the letters from a string. The `sub` command is used to alter a pattern that is not right in the string. The `ignore.Case` option provides the user the chance to keep the case sensitivity on or off while searching for the pattern in the defined string.

NA and missing value management

Missing value treatment is an important task in standard data mining literature. In the R programming language, missing values are represented as `NA`. NAs are not string or numeric values; they are considered as an indicator for missing values. After importing a dataset into the R programming platform, it is important to check whether, for any variable, missing values exist or not; to check that, the `is.na()` command is used. Please see the example given here:

```
> x<-c(12,13,14,21,23,24,NA,25,NA,0,NA)
> is.na(x)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> mean(x,na.rm=TRUE)
[1] 16.5
> mean(x)
[1] NA
```

Object `x` is a numeric vector that contains some NA values, to verify that `is.na()` can be used, wherever it is satisfied the result would be `TRUE`. If we compute anything with the presence of `NA`, we end up getting an error or no result. Either we can replace the data set by altering the NA values, or we can remove those NA values while performing any computation. As in the preceding script, it is `na.rm=T` that is used to remove the NAs from the mean computation for object `x`.

Missing value imputation techniques

To delete missing values from the dataset, `na.omit()` can be used. It removes the entire row even if the data is missing for a single variable. There are various missing value imputation methods:

- **Mean imputation**: The missing values in a data vector is replaced by the mean or median value of that vector, excluding the NA

- **Local average method**: Taking the local average for the missing value, by taking into account 3 or 5 periods moving average, that is for a 3 period the average of missing data prior value and posterior value can decide what the missing value should be
- **Keeping that separate**: Sometimes the imputation is simply not possible, and may be the client would be interested in keeping the missing values separate to understand the missing behavior separately
- Model-based: There are some model-based missing value imputation techniques such as the regression-based missing value prediction method
- **Clustering**: Similar to regression-based prediction to the missing value imputation, k-means clustering can be used to impute the missing values from the dataset

Summary

In the light of the preceding discussion, it can be summarized that data manipulation and data management is an important step in performing data mining on various live projects. Since R provides a better statistical programming platform and visualization, the use of R to explain various data mining concepts to the readers makes sense. In this chapter, we looked at an introduction to data mining and R with concepts, a few programming basics, R data types, and so on. We also covered importing and exporting of various external file formats using R, sorting and merging concepts, and missing data management techniques.

In the next chapter, we are going to learn more about performing exploratory data analysis using R and how to understand univariate, bivariate and multivariate datasets. First of all, we are going to understand the concepts, practical interpretation, and then R implementation to gain knowledge on exploratory data analysis.

2

Exploratory Data Analysis with Automobile Data

Exploratory data analysis is an integral part of data mining. It involves numerical as well as graphical representation of variables in a dataset for easy understanding and quick conclusion about a dataset. It is important to get an understanding about the dataset, type of variables considered for analysis, association between various variables, and so on. Creating cross tabulations to understand the relationship between categorical variables and performing classical statistical tests on the data to verify various different hypotheses about the data can be tested out.

You will now get an understanding about the following things:

- How to use basic statistics to know properties of a single and multiple variables
- How to calculate correlation and association between two or more variables
- Performing multivariate data analysis
- Statistical properties of various probability functions for any dataset
- Applying statistical tests on data to conclude hypotheses
- Comparing two or more samples

Univariate data analysis

To generate univariate statistics about a dataset, we have to follow two approaches, one for continuous variables and the other for discrete or categorical variables. Univariate statistics for continuous variables includes numerical measures such as averages (mean), variance, standard deviation, quantiles, median quartiles, and so on. The mean represents each and every point in the dataset; the variance indicates the fluctuation/deviation of the individual data points from the mean, which is the center of the distribution. Quantiles are also known as percentiles, which divide the distribution into 100 equal parts. The 10th percentile value is equivalent to 1 decile, the 25th percentile value is equivalent to the 1st quartile, and 75th percentile value is equivalent to the 3rd quartile.

There are other statistical measures of central tendency for understanding the univariate characteristics of a dataset. Median and mode are referred to as positional values, but still mode can be looked at for a continuous variable to check whether it is a bimodal series. In case of bimodal series, it is difficult to compute the center of the distribution. For ordinal data or rank data calculation of mean, representation is a good idea; it is always suggested to represent univariate statistics using median or mode. Comparison of mean, median, and mode values along with skewness, kurtosis, and standard deviation gives a clear picture about the shape of the data distribution. All of these measures of central tendency and measures of dispersion together can be calculated using a single command and also using different individual commands, which are given as follows.

Here, we are going to use two datasets, `diamonds.csv` and `Cars93.csv`. Both belong to two libraries that are inbuilt in the R software for practical demo purposes.

Let's use a few commands in R to understand the data better:

```
> names(Cars93)
[1] "Manufacturer" "Model" "Type" "Min.Price"
[5] "Price" "Max.Price" "MPG.city" "MPG.highway"
[9] "AirBags" "DriveTrain" "Cylinders" "EngineSize"
[13] "Horsepower" "RPM" "Rev.per.mile" "Man.trans.avail"
[17] "Fuel.tank.capacity" "Passengers" "Length" "Wheelbase"
[21] "Width" "Turn.circle" "Rear.seat.room" "Luggage.room"
[25] "Weight" "Origin" "Make"
```

The `Cars93.csv` dataset contains the previously mentioned variable names and it has 27 variables and 93 observations. The variable type can be printed using the `str()` function:

```
> str(Cars93)
'data.frame': 93 obs. of 27 variables:
```

```
$ Manufacturer : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5
...
$ Model : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74
73 35 ...
$ Type : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
$ Min.Price : num 12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
$ Price : num 15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
$ Max.Price : num 18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
$ MPG.city : int 25 18 20 19 22 22 19 16 19 16 ...
$ MPG.highway : int 31 25 26 26 30 31 28 25 27 25 ...
$ AirBags : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2
...
$ DriveTrain : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
$ Cylinders : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5
...
$ EngineSize : num 1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
$ Horsepower : int 140 200 172 172 208 110 170 180 170 200 ...
$ RPM : int 6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
$ Rev.per.mile : int 2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
$ Man.trans.avail : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
$ Fuel.tank.capacity: num 13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
$ Passengers : int 5 5 5 6 4 6 6 6 5 6 ...
$ Length : int 177 195 180 193 186 189 200 216 198 206 ...
$ Wheelbase : int 102 115 102 106 109 105 111 116 108 114 ...
$ Width : int 68 71 67 70 69 69 74 78 73 73 ...
$ Turn.circle : int 37 38 37 37 39 41 42 45 41 43 ...
$ Rear.seat.room : num 26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
$ Luggage.room : int 11 15 14 17 13 16 17 21 14 18 ...
$ Weight : int 2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
$ Origin : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
$ Make : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

Calculating the univariate statistics for a few continuous (`Price`, `MPG.city`, and `MPG.highway`) and discrete variables (`Type`, `AirBags`, and manual transmission available) can be displayed here. You can practice the rest of the variable to get a complete understanding of the dataset.

The `max()` command estimates the maximum value for a variable. `min()` computes the minimum value. `sum()` calculates the total of all the values. The `mean()` function calculates the arithmetic average of the values, the `median()` function calculates the median value, and `range()` calculates the vector of `min()` and `max()`. `var()` computes the sample variance and `cor()` correlation between two vectors. `rank()` calculates the vector of the ranks of the values in a vector. The `quantile()` function computes a vector containing the minimum, lower quartile, median, upper quartile, and maximum of a vector.

Using the `summary()` function for univariate:

```
> summary(Cars93$Price)
Min. 1st Qu. Median Mean 3rd Qu. Max.
7.40 12.20 17.70 19.51 23.30 61.90
> summary(Cars93$MPG.city)
Min. 1st Qu. Median Mean 3rd Qu. Max.
15.00 18.00 21.00 22.37 25.00 46.00
> summary(Cars93$MPG.highway)
Min. 1st Qu. Median Mean 3rd Qu. Max.
20.00 26.00 28.00 29.09 31.00 50.00
> summary(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
> summary(Cars93$AirBags)
Driver & Passenger Driver only None
16 43 34
> summary(Cars93$Man.trans.avail)
No Yes
32 61
```

Now let's look at the results of the summary function on the data frame. For a continuous variable, the numerical measures of central tendency are computed, and, for a categorical variable, the class frequencies are computed:

```
> summary(Cars93)
Manufacturer Model Type Min.Price
Chevrolet: 8 100 : 1 Compact:16 Min. : 6.70
Ford : 8 190E : 1 Large :11 1st Qu.:10.80
Dodge : 6 240 : 1 Midsize:22 Median :14.70
Mazda : 5 300E : 1 Small :21 Mean :17.13
Pontiac : 5 323 : 1 Sporty :14 3rd Qu.:20.30
Buick : 4 535i : 1 Van : 9 Max. :45.40
(Other) :57 (Other):87
Price Max.Price MPG.city MPG.highway
Min. : 7.40 Min. : 7.9 Min. :15.00 Min. :20.00
1st Qu.:12.20 1st Qu.:14.7 1st Qu.:18.00 1st Qu.:26.00
Median :17.70 Median :19.6 Median :21.00 Median :28.00
Mean :19.51 Mean :21.9 Mean :22.37 Mean :29.09
3rd Qu.:23.30 3rd Qu.:25.3 3rd Qu.:25.00 3rd Qu.:31.00
Max. :61.90 Max. :80.0 Max. :46.00 Max. :50.00
AirBags DriveTrain Cylinders EngineSize
Driver & Passenger:16 4WD :10 3 : 3 Min. :1.000
Driver only :43 Front:67 4 :49 1st Qu.:1.800
None :34 Rear :16 5 : 2 Median :2.400
6 :31 Mean :2.668
8 : 7 3rd Qu.:3.300
rotary: 1 Max. :5.700
```

```
Horsepower RPM Rev.per.mile Man.trans.avail
Min. : 55.0 Min. :3800 Min. :1320 No :32
1st Qu.:103.0 1st Qu.:4800 1st Qu.:1985 Yes:61
Median :140.0 Median :5200 Median :2340
Mean :143.8 Mean :5281 Mean :2332
3rd Qu.:170.0 3rd Qu.:5750 3rd Qu.:2565
Max. :300.0 Max. :6500 Max. :3755
Fuel.tank.capacity Passengers Length Wheelbase
Min. : 9.20 Min. :2.000 Min. :141.0 Min. : 90.0
1st Qu.:14.50 1st Qu.:4.000 1st Qu.:174.0 1st Qu.: 98.0
Median :16.40 Median :5.000 Median :183.0 Median :103.0
Mean :16.66 Mean :5.086 Mean :183.2 Mean :103.9
3rd Qu.:18.80 3rd Qu.:6.000 3rd Qu.:192.0 3rd Qu.:110.0
Max. :27.00 Max. :8.000 Max. :219.0 Max. :119.0
Width Turn.circle Rear.seat.room Luggage.room
Min. :60.00 Min. :32.00 Min. :19.00 Min. : 6.00
1st Qu.:67.00 1st Qu.:37.00 1st Qu.:26.00 1st Qu.:12.00
Median :69.00 Median :39.00 Median :27.50 Median :14.00
Mean :69.38 Mean :38.96 Mean :27.83 Mean :13.89
3rd Qu.:72.00 3rd Qu.:41.00 3rd Qu.:30.00 3rd Qu.:15.00
Max. :78.00 Max. :45.00 Max. :36.00 Max. :22.00
NA's :2 NA's :11
Weight Origin Make
Min. :1695 USA :48 Acura Integra: 1
1st Qu.:2620 non-USA:45 Acura Legend : 1
Median :3040 Audi 100 : 1
Mean :3073 Audi 90 : 1
3rd Qu.:3525 BMW 535i : 1
Max. :4105 Buick Century: 1
(Other) :87
```

The summary command for continuous variables such as RPM, horsepower, and so on shows the minimum, 1st quartile, mean, median, 3rd quartile, and maximum values. The univariate statistics for the categorical variable, which are car type, airbags, manual transmission availability, and so on, are represented as frequency tables. The class that has the highest frequency is considered to be the modal class.

Similar summary statistics can be generated using a few more functions such as `fivenum()` and `describe()`, which provides more information than the summary function:

```
> fivenum(Cars93$Price)
[1] 7.4 12.2 17.7 23.3 61.9
> fivenum(Cars93$MPG.city)
[1] 15 18 21 25 46
> fivenum(Cars93$MPG.highway)
[1] 20 26 28 31 50
```

The `describe()` function from the library (`Hmisc`) can be used to get a better understanding of the data description:

```
> library(Hmisc)
> describe(Cars93)
Cars93
27 Variables 93 Observations
Price
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 81 1 19.51 8.52 9.84 12.20 17.70 23.30 33.62
.95
36.74
lowest : 7.4 8.0 8.3 8.4 8.6, highest: 37.7 38.0 40.1 47.9 61.9
MPG.city
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 21 0.99 22.37 16.6 17.0 18.0 21.0 25.0 29.0
.95
31.4
lowest : 15 16 17 18 19, highest: 32 33 39 42 46
---------------------------------------------------------------------------
--------------
MPG.highway
n missing unique Info Mean .05 .10 .25 .50 .75 .90
93 0 22 0.99 29.09 22.0 23.2 26.0 28.0 31.0 36.0
.95
37.4
lowest : 20 21 22 23 24, highest: 38 41 43 46 50
---------------------------------------------------------------------------
--------------
AirBags
n missing unique
93 0 3
Driver & Passenger (16, 17%), Driver only (43, 46%)
None (34, 37%)
---------------------------------------------------------------------------
--------------
Man.trans.avail
n missing unique
93 0 2
No (32, 34%), Yes (61, 66%)
```

The univariate summary statistics can also be calculated using the `apply()` function. The univariate statistics gives an impression about the shape of the distribution:

```
> n_cars93<-Cars93[,c(5,7,8)]
> c_cars93<-Cars93[,c(3,9,16)]
> apply(n_cars93,2,mean)
Price MPG.city MPG.highway
19.50968 22.36559 29.08602
```

To understand the shape of a distribution for a variable, we can use skewness and box plot. We can also use the `skewness` function, which is available in `library(e1071)`:

```
> library(e1071)
> apply(n_cars93,2,skewness)
Price MPG.city MPG.highway
1.483982 1.649843 1.190507
```

We can create a custom function for measuring skewness and use that along with the `apply()` function:

```
> skewness<-function(x){
+ m3<-sum((x-mean(x))^3)/length(x)
+ s3<-sqrt(var(x))^3
+ m3/s3 }
> apply(n_cars93,2,skewness)
Price MPG.city MPG.highway
1.483982 1.649843 1.190507
```

Skewness is known as a measure of symmetrical distribution where the value of skewness indicates whether a distribution is positively skewed or negatively. When the value of skewness approaches 0 or is close to zero, it indicates that the distribution is symmetric and the mean, median, and mode are exactly the same. When the value of skewness is less than 0, it indicates the value of the mean is less than the mode, and this is because of availability of extreme values on the negative side of the normal distribution. When the value of skewness is greater than 0, it indicates that the value of the mean is greater than the mode, and this is because of availability of extreme values on the right-hand side of the normal distribution. Since identification of outliers and their removal is very important, the measure of skewness helps in that direction, but that is not the only way to figure out outliers. There are other methods such as boxplot, and other custom outlier detection formulas. If we look at the previous three variables, it implies that there are indications that outlier values may exist on the positive side of the normal curve for price, `MPG.city`, and `MPG.highway` variables because the skewness value is greater than 0. To verify the existence of outliers, we can take a boxplot and print the outliers.

Bivariate analysis

The relationship or association between two variables is known as bivariate analysis. There are three possible ways of looking at the relationship:

- Numeric-to-numeric relationship
- Numeric-to-categorical relationship
- Categorical-to-categorical relationship

To know the bivariate relationship between two numeric variables, typically a scatter plot is used if the two variables happen to be continuous, and a bar plot is used if one variable is categorical and the other is continuous:

```
> library(ggplot2)
> library(gridExtra)
> ggplot(Cars93,
aes(Cars93$Price,Cars93$MPG.city))+geom_point(aes(colour=(Cars93$Type)))+ge
om_smooth()
```

Figure 1: Showing the relationship between price and mileage within a city for different car types

Similarly, the relationship between price and highway mileage can be represented using a scatter plot as well:

```
> library(ggplot2)
> library(gridExtra)
> ggplot(Cars93,
aes(Cars93$Price,Cars93$MPG.highway))+geom_point(aes(colour=(Cars93$Type)))
+geom_smooth()
```

Figure 2: Relationship between price and mileage on highways

The numeric-categorical and two categorical relationships are explained in `Chapter 3`, *Visualize Diamond Dataset*, in detail.

Multivariate analysis

The multivariate relationship is a statistical way of looking at multiple dependent and independent variables and their relationships. In this chapter, we will briefly talk about multivariate relationships between more than two variables, but we will discuss the details of multivariate analysis in our subsequent chapters. Multivariate relationships between various variables can be known by using the correlation method as well as cross tabulation:

```
> pairs(n_cars93,main="Correlation Plot", col="blue")
```

Understanding distributions and transformation

Understanding probability distributions is important in order to have a clear idea about the assumptions of any statistical hypothesis test. For example, in linear regression analysis, the basic assumption is that the error distribution should be normally distributed and the variables' relationship should be linear. Hence, before moving to the stage of model formation, it is important to look at the shape of the distribution and types of transformations that one may look into to make the things right. This is done so that any further statistical techniques can be applied on the variables.

Normal probability distribution

The concept of normal distribution is based on **Central Limit Theorem** (**CLT**), which implies that the population of all possible samples of size n drawn from a population with mean μ and variance $\sigma 2$ approximates a normal distribution with mean μ and $\sigma 2 \backslash n$ when n increases towards infinity. Checking the normality of variables is important to remove outliers so that the prediction process does not get influenced. Presence of outliers not only deviates the predicted values but would also destabilize the predictive model. The following sample code and example show how to check normality graphically and interpret the same.

To test out the normal distribution, we can use the mean, median, and mode for some of the variables:

```
> mean(Cars93$Price)
[1] 19.50968
> median(Cars93$Price)
[1] 17.7
> sd(Cars93$Price)
[1] 9.65943
> var(Cars93$Price)
[1] 93.30458
> skewness(Cars93$Price)
[1] 1.483982
ggplot(data=Cars93, aes(Cars93$Price)) + geom_density(fill="blue")
```

From the preceding image, we can conclude that the `price` variable is positively skewed because of the presence of some outlier values on the right-hand side of the distribution. The mean of the `price` variable is inflated and greater than the mode because the mean is subject to extreme fluctuations.

Now let's try to understand a case where normal distribution can be used to answer any hypothesis.

Suppose the variable mileage per gallon on a highway is normally distributed with a mean of `29.08` and a standard deviation of `5.33`. What is the probability that a new car would provide a mileage of `35`?

```
> pnorm(35,mean(Cars93$MPG.highway),sd(Cars93$MPG.highway),lower.tail = F)
[1] 0.1336708
```

Hence the required probability that a new car would provide a mileage of `35` is 13.36%, since the expected mean is higher than the actual mean; the lower tail is equivalent to false.

Binomial probability distribution

Binomial distribution is known as discrete probability distribution. It describes the outcome of an experiment. Each trial is assumed to have only two outcomes: either success or failure, either yes or no. For instance, in the `Cars93` dataset variable, whether manual transmission is available or not is represented as *yes* or *no*.

Let's take an example to explain where binomial distribution can be used. The probability of a defective car given a specific component is not functioning is 0.1%. You have 93 cars manufactured. What is the probability that at least 1 defective car can be detected from the current lot of `93`:

```
> pbinom(1,93,prob = 0.1)
[1] 0.0006293772
```

So the required probability that a defective car can get identified in a lot of 93 cars is `0.0006`, which is very less, given the condition that the probability of a defective part is `0.10`.

Poisson probability distribution

Poisson distribution is for count data where, given the data and information about an event, you can predict the probability of any number occurring within that limit using the Poisson probability distribution.

Let's take an example. Suppose 200 customers on an average visit a particular e-commerce website portal every minute. Then find the probability of having 250 customers visit the same website in a minute:

```
> ppois(250,200,lower.tail = F)
[1] 0.0002846214
```

Hence, the required probability is `0.0002`, which is very rare. Apart from the aforementioned common probability distributions, which are used very frequently, there are many other distributions that can be used in rare situations.

Interpreting distributions

Calculation of probability distributions and fitting data points specific to various types of distribution and subsequent interpretation help in forming a hypothesis. That hypothesis can be used to estimate the event probability given a set of parameters. Let's take a look at the interpretation of different types of distributions.

Interpreting continuous data

The maximum likelihood estimation of the distributional parameters from any variable in a dataset can be known by fitting a distribution. The density function is available for distributions such as “beta”, “cauchy”, “chi-squared”, “exponential”, “f”, “gamma”, “geometric”, “log-normal”, “lognormal”, “logistic”, “negative binomial”, “normal”, “Poisson”, “t”, and “weibull”. These are recognized, case being ignored. For continuous data, we will be using normal and t distributions:

```
> x<-fitdistr(Cars93$MPG.highway,densfun = "t")
> x$estimate
m s df
28.430527 3.937731 4.237910
> x$sd
m s df
0.5015060 0.5070997 1.9072796
> x$vcov
m s df
m 0.25150831 0.06220734 0.2607635
s 0.06220734 0.25715007 0.6460305
df 0.26076350 0.64603055 3.6377154
> x$loglik
[1] -282.4481
> x$n
[1] 93
```

In the preceding code, we have taken the `MPG.highway` variable from the `Cars93` dataset. By fitting a t distribution to the variable, we obtain the parameter estimates, the estimated standard errors, the estimated variance co-variance matrix, the log likelihood value, as well as the total count. Similar activity can be performed by fitting a normal distribution to the continuous variable:

```
> x<-fitdistr(Cars93$MPG.highway,densfun = "normal")
> x$estimate
mean sd
29.086022 5.302983
> x$sd
```

```
mean sd
0.5498938 0.3888336
> x$vcov
mean sd
mean 0.3023831 0.0000000
sd 0.0000000 0.1511916
> x$loglik
[1] -287.1104
> x$n
[1] 93
```

Now we are going to see how to graphically represent the variable normality:

```
> qqnorm(Cars93$MPG.highway)
> qqline(Cars93$MPG.highway)
```

The deviation of data points represented as circles are distanced from the straight line, *t*.

Interpreting discrete data: as all the categories in it:

```
> table(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
```

```
> freq<-table(Cars93$Type)
> rel.freq<-freq/nrow(Cars93)*100
> options(digits = 2)
> rel.freq
Compact Large Midsize Small Sporty Van
17.2 11.8 23.7 22.6 15.1 9.7
> cbind(freq,rel.freq)
freq rel.freq
Compact 16 17.2
Large 11 11.8
Midsize 22 23.7
Small 21 22.6
Sporty 14 15.1
Van 9 9.7
```

To visualize this graphically, we need to represent it through a bar plot:

```
> barplot(freq, main = "Distribution of Categorical Variable")
```

Variable binning or discretizing continuous data

The continuous variable is the most appropriate step that one needs to take before including the variable in the model. This can be explained by taking one example fuel tank capacity of a car from the `Cars93` dataset. Based on the fuel tank capacity, we can create a categorical variable with high, medium and low, lower medium:

```
> range(Cars93$Fuel.tank.capacity)
[1] 9.2 27.0
> cat
[1] 9.2 13.2 17.2 21.2 25.2
> options(digits = 2)
> t<-cut(Cars93$Fuel.tank.capacity,cat)
> as.data.frame(cbind(table(t)))
V1
(9.2,13.2] 19
(13.2,17.2] 33
(17.2,21.2] 36
(21.2,25.2] 3
```

The range of fuel tank capacity is identified as `9.2` and `27.0`. Then, logically the class difference of 4 is used to arrive at classes. Those classes define how each value from the variable is assigned to each group. The final outcome table indicates that there are 4 groups; the top fuel tank capacity is available on 4 cars only.

Variable binning or discretization not only helps in decision tree construction but is also useful in the case of logistic regression mode and any other form of machine-learning-based models.

Contingency tables, bivariate statistics, and checking for data normality

Contingency tables are frequency tables represented by two or more categorical variables along with the proportion of each class represented as a group. Frequency table is used to represent one categorical variable; however, contingency table is used to represent two categorical variables.

Let's see an example to understand contingency tables, bivariate statistics, and data normality using the `Cars93` dataset:

```
> table(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
> table(Cars93$AirBags)
Driver & Passenger Driver only None
16 43 34
```

The individual frequency table for two categorical variables `AirBags` and `Type` of the car is represented previously:

```
> contTable<-table(Cars93$Type,Cars93$AirBags)
> contTable
Driver & Passenger Driver only None
Compact 2 9 5
Large 4 7 0
Midsize 7 11 4
Small 0 5 16
Sporty 3 8 3
Van 0 3 6
```

The `conTable` object holds the cross tabulation of two variables. The proportion of each cell in percentage is reflected in the following table. If we need to compute the row percentages or column percentages, then it is required to specify the values in the argument:

```
> prop.table(contTable)
Driver & Passenger Driver only None
Compact 0.022 0.097 0.054
Large 0.043 0.075 0.000
Midsize 0.075 0.118 0.043
Small 0.000 0.054 0.172
Sporty 0.032 0.086 0.032
Van 0.000 0.032 0.065
```

For row percentages, the value needs to be 1, and for column percentages, the value needs to be entered as 2 in the preceding command:

```
> prop.table(contTable,1)
Driver & Passenger Driver only None
Compact 0.12 0.56 0.31
Large 0.36 0.64 0.00
Midsize 0.32 0.50 0.18
Small 0.00 0.24 0.76
Sporty 0.21 0.57 0.21
Van 0.00 0.33 0.67
> prop.table(contTable,2)
```

```
Driver & Passenger Driver only None
Compact 0.125 0.209 0.147
Large 0.250 0.163 0.000
Midsize 0.438 0.256 0.118
Small 0.000 0.116 0.471
Sporty 0.188 0.186 0.088
Van 0.000 0.070 0.176
```

The summary of the contingency table performs a chi-square test of independence between the two categorical variables:

```
> summary(contTable)
Number of cases in table: 93
Number of factors: 2
Test for independence of all factors:
Chisq = 33, df = 10, p-value = 3e-04
Chi-squared approximation may be incorrect
```

The chi-square test of independence for all factors is represented previously. The message that the chi-squared approximation may be incorrect is due to the presence of null or less than 5 values in the cells of the contingency table. As in the preceding case, two random variables, car type and airbags, can be independent if the probability distribution of one variable does not impact the probability distribution of the other variable. The null hypothesis for the chi-square test of independence is that two variables are independent of each other. Since the p-value from the test is less than 0.05, at 5% level of significance we can reject the null hypothesis that the two variables are independent. Hence, the conclusion is that car type and airbags are not independent of each other; they are quite related or dependent.

Instead of two variables, what if we add one more dimension to the contingency table? Let's take `Origin`, and then the table would look as follows:

```
> contTable<-table(Cars93$Type,Cars93$AirBags,Cars93$Origin)
> contTable
, , = USA
Driver & Passenger Driver only None
Compact 1 2 4
Large 4 7 0
Midsize 2 5 3
Small 0 2 5
Sporty 2 5 1
Van 0 2 3
, , = non-USA
Driver & Passenger Driver only None
Compact 1 7 1
Large 0 0 0
```

```
Midsize 5 6 1
Small 0 3 11
Sporty 1 3 2
Van 0 1 3
```

The `summary` command for the test of independence of all factors can be used to test out the null hypothesis:

```
> summary(contTable)
Number of cases in table: 93
Number of factors: 3
Test for independence of all factors:
Chisq = 65, df = 27, p-value = 5e-05
Chi-squared approximation may be incorrect
```

Apart from the graphical methods discussed previously, there are some numerical statistical tests that can be used to know whether a variable is normally distributed or not. There is a library called `norm.test` for performing data normality tests, a list of functions that help in assessing the data normality from this library are listed as follows:

`ajb.norm.test`	**Adjusted Jarque-Bera test for normality**
`frosini.norm.test`	Frosini test for normality
`geary.norm.test`	Geary test for normality
`hegazy1.norm.test`	Hegazy-Green test for normality
`hegazy2.norm.test`	Hegazy-Green test for normality
`jb.norm.test`	Jarque-Bera test for normality
`kurtosis.norm.test`	Kurtosis test for normality
`skewness.norm.test`	Skewness test for normality
`spiegelhalter.norm.test`	Spiegelhalter test for normality
`wb.norm.test`	Weisberg-Bingham test for normality

`ad.test`	**Anderson-Darling test for normality**
`cvm.test`	Cramér-von Mises test for normality
`lillie.test`	Lilliefors (Kolmogorov-Smirnov) test for normality
`pearson.test`	Pearson chi-square test for normality
`sf.test`	Shapiro-Francia test for normality

Let's apply the normality test on the `Price` variable from the `Cars93` dataset:

```
> library(nortest)
> ad.test(Cars93$Price) # Anderson-Darling test
Anderson-Darling normality test
data: Cars93$Price
A = 3, p-value = 9e-07
> cvm.test(Cars93$Price) # Cramer-von Mises test
Cramer-von Mises normality test
data: Cars93$Price
W = 0.5, p-value = 6e-06
> lillie.test(Cars93$Price) # Lilliefors (KS) test
Lilliefors (Kolmogorov-Smirnov) normality test
data: Cars93$Price
D = 0.2, p-value = 1e-05
> pearson.test(Cars93$Price) # Pearson chi-square
Pearson chi-square normality test
data: Cars93$Price
P = 30, p-value = 3e-04
> sf.test(Cars93$Price) # Shapiro-Francia test
Shapiro-Francia normality test
data: Cars93$Price
```

From the previously mentioned tests, it is evident that the `Price` variable is not normally distributed as the p-values from all the statistical tests are less than `0.05`. If we add more dimensions to the bi-variate relationship, it becomes multivariate analysis. Let's try to understand the relationship between horsepower and length of a car from the `Cars93` dataset:

```
> library(corrplot)
> o<-cor(Cars93[,c("Horsepower","Length")])
> corrplot(o,method = "circle",main="Correlation Plot")
```

When we include more variables, it becomes a multivariate relationship. Let's try to plot a multivariate relationship between various variables from the `Cars93` dataset:

```
> library(corrplot)
> t<-
cor(Cars93[,c("Price","MPG.city","RPM","Rev.per.mile","Width","Weight","Hor
sepower","Length")])
> corrplot(t,method = "ellipse")
```

There are various methods that can be passed as an argument to the correlation plot. They are `"circle"`, `"square"`, `"ellipse"`, `"number"`, `"shade"`, `"color"`, and `"pie"`.

Hypothesis testing

The null hypothesis states that nothing has happened, the means are constant, and so on. However, the alternative hypothesis states that something different has happened and the means are different about a population. There are certain steps in performing a hypothesis test:

1. **State the null hypothesis**: A statement about the population is assumed; for example, the average mileage of cars within a city is 40.
2. **State the alternative hypothesis**: If the null hypothesis turns out to be false, then what other possibility is there? For example, if the mileage within the city is not 40, then is it greater than 40 or less than 40? If it is not equal to 40, then it is a non-directional alternative hypothesis.
3. **Calculate the sample test statistic**: The test statistic could be t-test, f-test, z-test, and so on. Select the appropriate test statistic based on the data availability and the hypothesis declared previously.
4. **Decide the confidence limit**: There are three different confidence limits: 90%, 95% and 99% depending on the degree of accuracy related to a specific business problem. It is up to the researcher/analyst to choose the level of confidence interval.
5. **Define the alpha value**: If the confidence level selected is 95%, the alpha value is going to be 5%. Hence deciding the alpha value would help in calculating the p-value for the test.
6. **Decision**: If the p-value selected is less than the alpha level, then there is evidence that the null hypothesis can be rejected; if it is not, then we are going to accept the null hypothesis.

Test of the population mean

Using the hypothesis testing procedure, let's take an example from the `Cars93` dataset to test out the population mean.

One tail test of mean with known variance

Suppose the researcher claims that the average mileage given by all the cars collected in the sample is more than 35. In the sample of 93 cars, it is observed that the mean mileage of all cars is 29. Should you accept or reject the researcher's claim?

The following script explains how you are going to conclude this:

```
Null Hypothesis: mean = 35
Alternative hypothesis= mean > 35> mu<-mean(Cars93$MPG.highway)
> mu
[1] 29
> sigma<-sd(Cars93$MPG.highway)
> sigma
[1] 5.3
> n<-length(Cars93$MPG.highway)
> n
[1] 93
> xbar= 35
> z<-(xbar-mu)/(sigma/sqrt(n))
> z
[1] 11
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha)
> z1
[1] 1.6
> ifelse(z > z1,"Reject the Null Hypothesis","Accept the Null Hypothesis")
Null Hypothesis: mean = 35
Alternative hypothesis= mean < 35
Two tail test of mean, with known variance:> mu<-mean(Cars93$MPG.highway)
> mu
[1] 29.09
> sigma<-sd(Cars93$MPG.highway)
> sigma
[1] 5.332
> n<-length(Cars93$MPG.highway)
> n
[1] 93
> xbar= 35
> z<-(xbar-mu)/(sigma/sqrt(n))
> z
[1] 10.7
>
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm((1-alpha)/2))
Error: unexpected ')' in "z1 = qnorm((1-alpha)/2))"
> c(-z1,z1)
[1] -1.96 1.96
>
>
> ifelse(z > z1 | z < -z1,"Reject the Null Hypothesis","Accept the Null
Hypothesis")
```

The way we analyzed the one-tail and two-tail test of population mean for the sample data in the case of known variance.

One tail and two tail test of proportions

Using the `Cars93` dataset, suppose 40% of the USA-manufactured cars have an RPM of more than 5000. From the sample data, we found that 17 out of 57 cars have RPM above 5000. What do you interpret from the context?

```
> mileage<-subset(Cars93,Cars93$RPM > 5000)
> table(mileage$Origin)
USA non-USA
17 40
> p1<-17/57
> p0<- 0.4
> n <- length(mileage)
> z <- (p1-p0)/sqrt(p0*(1-p0)/n)
> z
[1] -1.079
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha)
> z1
[1] 1.645
> ifelse(z > z1,"Reject the Null Hypothesis","Accept the Null Hypothesis")
[1] "Accept the Null Hypothesis"
```

If the alternative hypothesis is not directional, it is a case of two-tailed test of proportions; nothing from the preceding calculation would change except the critical value calculation. The detailed script is given as follows:

```
> mileage<-subset(Cars93,Cars93$RPM > 5000)
> table(mileage$Origin)
USA non-USA
17 40
> p1<-17/57
> p0<- 0.4
> n <- length(mileage)
> z <- (p1-p0)/sqrt(p0*(1-p0)/n)
> z
[1] -1.079
> #computing the critical value at 5% alpha level
> alpha = .05
> z1 = qnorm(1-alpha/2)
> c(-z1,z1)
[1] -1.96 1.96
```

```
> ifelse(z > z1 | z < -z1,"Reject the Null Hypothesis","Accept the Null
Hypothesis")
[1] "Accept the Null Hypothesis"
```

- **Two sample paired test for continuous data**: The null hypothesis that is being tested in the two-sample paired test would be that there is no impact of a procedure on the subjects, the treatment has no effect on the subjects, and so on. The alternative hypothesis would be there is a statistically significant impact of a procedure, effectiveness of a treatment, or drug on the subjects.

  Though we don't have such a variable in the `Cars93` dataset, we can still assume the paired relationship between minimum prices and maximum prices for different brands of cars.

- **Null hypothesis for the two sample t-test**: There is no difference in the mean prices.
- **Alternative hypothesis**: There is a difference in the mean prices:

```
> t.test(Cars93$Min.Price, Cars93$Max.Price, paired = T)
Paired t-test
data: Cars93$Min.Price and Cars93$Max.Price
t = -9.6, df = 92, p-value = 2e-15
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-5.765 -3.781
sample estimates:
mean of the differences
-4.773
```

  The p-value is less than 0.05. Hence, it can be concluded that the difference in mean minimum price and maximum price is statistically significant at alpha 95% confidence level.

- **Two sample unpaired test for continuous data**: From the `Cars93` dataset, the mileage on the highway and within the city is assumed to be different. If the difference is statistically significant, can be tested by using the independent samples t-test for comparison of means.
- **Null hypothesis**: There is no difference in the MPG on highway and MPG within city.
- **Alternative hypothesis**: There is a difference in the MPG on highway and MPG within city:

```
Welch Two Sample t-test
data: Cars93$MPG.city and Cars93$MPG.highway
```

```
t = -8.4, df = 180, p-value = 1e-14
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-8.305 -5.136
sample estimates:
mean of x mean of y
22.37 29.09
```

From the two samples t-test, when the two samples are independent, the p-value is less than 0.05; hence, we can reject the null hypothesis that there is no difference in the mean mileage on highway and within city. There is a statistically significant difference in the mean mileage within city and on highway. This can be represented in a slightly different way, by putting a null hypothesis is the mean mileage difference within city is different for manual versus automatic cars:

```
, data=Cars93)
Welch Two Sample t-test
data: Cars93$MPG.city by Cars93$Man.trans.avail
t = -6, df = 84, p-value = 4e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-6.949 -3.504
sample estimates:
mean in group No mean in group Yes
18.94 24.16
```

So, the conclusion from the preceding test is that there is a statistically significant difference in the mean mileage between automatic and manual transmission vehicle types; this is because the p-value is less than 0.05.

Before applying t-test, it is important to check the data normality; normality of a variable can be assessed using the Shapiro test function:

```
> shapiro.test(Cars93$MPG.city)
Shapiro-Wilk normality test
data: Cars93$MPG.city
W = 0.86, p-value = 6e-08
)
> qqline(Cars93$MPG.city)
```

Looking at the QQ plot for the lineage per gallon within city and the histogram, it can be concluded that the variable is not normally distributed. Since the mileage variable is not normally distributed, it is required to apply non-parametric methods such as Wilcoxon signed rank test or Kolmogorov-Smirnov test.

Two sample variance test

To compare the variances of two samples, F-test is used as a statistic:

```
> var.test(Cars93$MPG.highway~Cars93$Man.trans.avail, data=Cars93)
F test to compare two variances
data: Cars93$MPG.highway by Cars93$Man.trans.avail
F = 0.24, num df = 31, denom df = 60, p-value = 5e-05
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
0.1330 0.4617
sample estimates:
ratio of variances
0.2402
```

Since the p-value is less than 0.05, we can reject the null hypothesis that there is no difference in the variance of mileage on a highway for manual and automatic cars. This implies there is a statistically significant difference in the variance of two samples at 95% confidence level.

The variances of the two groups can also be tested using Bartlett test:

```
> bartlett.test(Cars93$MPG.highway~Cars93$Man.trans.avail, data=Cars93)
Bartlett test of homogeneity of variances
data: Cars93$MPG.highway by Cars93$Man.trans.avail
Bartlett's K-squared = 17, df = 1, p-value = 4e-05
```

From the preceding test, it can also be concluded that the null hypothesis of equal variances can be rejected at alpha 0.05 level, and it can be proven that there is a statistically significant difference in the variance of the two samples.

One-way ANOVA:, one-way ANOVA can be used. The variable considered is RPM, and the grouping variable considered is `Cylinders`.

Null hypothesis: There is no difference in the means of RPM across different cylinder types.

Alternative hypothesis: There is a difference in the mean RPM for at least one cylinder type:

```
> aov(Cars93$RPM~Cars93$Cylinders)
Call:
aov(formula = Cars93$RPM ~ Cars93$Cylinders)
Terms:
Cars93$Cylinders Residuals
Sum of Squares 6763791 25996370
Deg. of Freedom 5 87
Residual standard error: 546.6
Estimated effects may be unbalanced
```

```
> summary(aov(Cars93$RPM~Cars93$Cylinders))
Df Sum Sq Mean Sq F value Pr(>F)
Cars93$Cylinders 5 6763791 1352758 4.53 0.001 **
Residuals 87 25996370 298809
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the preceding ANOVA, the p-value is less than 0.05; hence, the null hypothesis can be rejected. That means at least for one cylinder type, the mean RPM is statistically significantly different. To identify which cylinder type is different, a post hoc test can be performed on the results of the ANOVA model:

```
> TukeyHSD(aov(Cars93$RPM~Cars93$Cylinders))
Tukey multiple comparisons of means
95% family-wise confidence level
Fit: aov(formula = Cars93$RPM ~ Cars93$Cylinders)
$`Cars93$Cylinders`
diff lwr upr p adj
4-3 -321.8 -1269.23 625.69 0.9201
5-3 -416.7 -1870.88 1037.54 0.9601
6-3 -744.1 -1707.28 219.11 0.2256
8-3 -895.2 -1994.52 204.04 0.1772
rotary-3 733.3 -1106.11 2572.78 0.8535
5-4 -94.9 -1244.08 1054.29 0.9999
6-4 -422.3 -787.90 -56.74 0.0140
8-4 -573.5 -1217.14 70.20 0.1091
rotary-4 1055.1 -554.08 2664.28 0.4027
6-5 -327.4 -1489.61 834.77 0.9629
8-5 -478.6 -1755.82 798.67 0.8834
rotary-5 1150.0 -801.03 3101.03 0.5240
8-6 -151.2 -817.77 515.47 0.9857
rotary-6 1477.4 -141.08 3095.92 0.0941
rotary-8 1628.6 -74.42 3331.57 0.0692
```

Wherever the p-adjusted value is less than 0.05, the mean difference in RPM is statistically significantly different from other groups.

Two way ANOVA with post hoc tests: The factors considered are origin and airbags. The hypothesis that needs to be tested is: is there any impact of both the categorical variables on the RPM variable?

```
> aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags)
Call:
aov(formula = Cars93$RPM ~ Cars93$Origin + Cars93$AirBags)
Terms:
Cars93$Origin Cars93$AirBags Residuals
Sum of Squares 8343880 330799 24085482
```

```
Deg. of Freedom 1 2 89
Residual standard error: 520.2
Estimated effects may be unbalanced
> summary(aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags))
Df Sum Sq Mean Sq F value Pr(>F)
Cars93$Origin 1 8343880 8343880 30.83 2.9e-07 ***
Cars93$AirBags 2 330799 165400 0.61 0.54
Residuals 89 24085482 270623
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> TukeyHSD(aov(Cars93$RPM~Cars93$Origin + Cars93$AirBags))
Tukey multiple comparisons of means
95% family-wise confidence level
Fit: aov(formula = Cars93$RPM ~ Cars93$Origin + Cars93$AirBags)
$`Cars93$Origin`
diff lwr upr p adj
non-USA-USA 599.4 384.9 813.9 0
$`Cars93$AirBags`
diff lwr upr p adj
Driver only-Driver & Passenger -135.74 -498.8 227.4 0.6474
None-Driver & Passenger -25.68 -401.6 350.2 0.9855
None-Driver only 110.06 -174.5 394.6 0.6280
```

Non-parametric methods

When a training dataset does not conform to any specific probability distribution because of non-adherence to the assumptions of that specific probability distribution, the only option left to analyze the data is via non-parametric methods. Non-parametric methods do not follow any assumption regarding the probability distribution. Using non-parametric methods, one can draw inferences and perform hypothesis testing without adhering to any assumptions. Now let's look at a set of on-parametric tests that can be used when a dataset does not conform to the assumptions of any specific probability distribution.

Wilcoxon signed-rank test

If the assumption of normality is violated, then it is required to apply non-parametric methods in order to answer a question such as: is there any difference in the mean mileage within the city between automatic and manual transmission type cars?

```
> wilcox.test(Cars93$MPG.city~Cars93$Man.trans.avail, correct = F)
Wilcoxon rank sum test
data: Cars93$MPG.city by Cars93$Man.trans.avail
W = 380, p-value = 1e-06
```

```
alternative hypothesis: true location shift is not equal to 0
```

The argument paired can be used if the two samples happen to be matching pairs and the samples do not follow the assumptions of normality:

```
> wilcox.test(Cars93$MPG.city, Cars93$MPG.highway, paired = T)
Wilcoxon signed rank test with continuity correction
data: Cars93$MPG.city and Cars93$MPG.highway
V = 0, p-value <2e-16
alternative hypothesis: true location shift is not equal to 0
```

Mann-Whitney-Wilcoxon test

If two samples are not matched, are independent, and do not follow a normal distribution, then it is required to use Mann-Whitney-Wilcoxon test to test the hypothesis that the mean difference in the two samples are statistically significantly different from each other:

```
> wilcox.test(Cars93$MPG.city~Cars93$Man.trans.avail, data=Cars93)
Wilcoxon rank sum test with continuity correction
data: Cars93$MPG.city by Cars93$Man.trans.avail
W = 380, p-value = 1e-06
alternative hypothesis: true location shift is not equal to 0
```

Kruskal-Wallis test

To compare means of more than two groups, that is, the non-parametric side of ANOVA analysis, we can use the Kruskal-Wallis test. It is also known as a distribution-free statistical test:

```
> kruskal.test(Cars93$MPG.city~Cars93$Cylinders, data= Cars93)
Kruskal-Wallis rank sum test
data: Cars93$MPG.city by Cars93$Cylinders
Kruskal-Wallis chi-squared = 68, df = 5, p-value = 3e-13
```

Summary

Exploratory data analysis is an important activity in almost all types of data mining projects. Understanding the distribution, shape of the distribution, and vital parameters of the distribution, is very important. Preliminary hypothesis testing can be done to know the data better. Not only the distribution and its properties but also the relationship between various variables is important. Hence in this chapter, we looked at bivariate and multivariate relationships between various variables and how to interpret the relationship. Classical statistical tests, such as t-test, F-test, z-test and other non-parametric tests are important to test out the hypothesis. The hypothesis testing itself is also important to draw conclusions and insights from the dataset.

In this chapter we discussed various statistical tests, their syntax, interpretations, and situations where we can apply those tests. After performing exploratory data analysis, in the next chapter we are going to look at various data visualization methods to get a 360-degree view of data. Sometimes, a visual story acts as the simplest possible representation of data. In the next chapter, we are going to use some inbuilt datasets with different libraries to create intuitive visualizations.

3
Visualize Diamond Dataset

Every data mining project is incomplete without proper data visualization. While looking at numbers and statistics it may tell a similar story for the variables we are looking at by different cuts, however, when we visually look at the relationship between variables and factors it shows a different story altogether. Hence data visualization tells you a message, that numbers and statistics fail to do that. From a data mining perspective, data visualization has many advantages, which can be summarized in three important points:

- Data visualization establishes a robust communication between the data and the consumer of the data
- It imprints a long lasting impact as people may fail to remember numbers but they do remember charts and shapes
- When data scales up to higher dimension, representation in numbers does not make sense, but visually it does

In this chapter, the reader will get to know the basics of data visualization along with how to create advanced data visualization using existing libraries in R programming language. Typically, data visualization approach can be addressed in two different ways:

- What do you want to display to the audience? Is it comparison, relationship, or any other functionality?
- How do you want to display the insights? Which one is the most intuitive way of chart or graph to display the insights?

Based on the preceding two points, let's have a look at the data visualization rules and theories behind each visualization, and then we are going to look at the practical aspect of implementing the graphs and charts using R Script.

From a functional point of view, the following are the graphs and charts which a data scientist would like the audience to look at to infer the information:

- **Comparisons between variables**: Basically, when it is needed to represent two or more categories within a variable, then the following charts are used:
    - Bar chart
    - Box plot
    - Bubble chart
    - Histogram
    - Line graph
    - Stacked bar chart
    - Radar chart
    - Pie chart
- **Testing/viewing proportions**: It is used when there is a need to display the proportion of contribution by one category to the overall level:
    - Bubble chart
    - Bubble map
    - Stacked bar chart
    - Word cloud
- **Relationship between variables**: Association between two or more variables can be shown using the following charts:
    - Scatterplot
    - Bar chart
    - Radar chart
    - Line graph
    - Tree diagram
- **Variable hierarchy**: When it is required to display the order in the variables, such as a sequence of variables, then the following charts are used:
    - Tree diagram
    - Tree map
- **Data with locations**: When a dataset contains the geographic location of different cities, countries, and states names, or longitudes and latitudes, then the following charts can be used to display visualization:
    - Bubble map
    - Geo mapping
    - Dot map
    - Flow map

- **Contribution analysis or part-to-whole**: When it is required to display constituents of a variable and contribution of each categorical level towards the overall variable, then the following charts are used:
    - Pie chart
    - Stacked bar chart
    - Donut chart
- **Statistical distribution**: In order to understand the variation in a variable across different dimensions, represented by another categorical variable, the following charts are used:
    - Box plot
    - Bubble chart
    - Histogram
    - Stem and leaf plot
- **Unseen patterns**: For pattern recognition and relative importance of data points on different dimensions of a variable, the following charts are used:
    - Bar chart
    - Box plot
    - Bubble chart
    - Scatterplot
    - Spiral plot
    - Line chart
- **Spread of values or range**: The following charts only give the spread of the data points across different bounds:
    - Span chart
    - Box plot
    - Histogram
- **Textual data representation**: This is a very interesting way of representing the textual data:
    - Word cloud

Keeping in mind the preceding functionalities that people use in displaying insights to the readers, we can see that one graph is referred by much functionality. In other words, one graph can be used in multiple functions to show the insights. These graphs and charts can be displayed by using various open source R packages, such as `ggplot2`, `ggvis`, `rCharts`, `plotly`, and `googleVis` by taking one open source dataset.

In the light of the previously mentioned ten points, the data visualization rules can be created to select the best representation depending on what you want to represent:

- Relationship between two variables can be represented using a scatterplot
- Relationship between more than two variables can be represented using a bubble chart
- For understanding the distribution of a single variable with few sample size, histogram is used
- For understanding the distribution of a single variable with large sample size, density plot is used
- Distribution of two variables can be represented using scatterplot
- For representation of distribution between 3 variables, 3D scatterplot is used
- Any variable with a time stamp, such as day, week, month, year, and so on, can be represented using line chart
- Time should always be on the horizontal axis and the metric to be measured should always be on the vertical axis
- Each graph should have a name and label so that the user does not have to go back to the data to understand what it is

In this chapter, we will primarily focus on the `ggplot2` library and `plotly` library. Of course we will cover a few more interesting libraries to create data visualization. The graphics packages in R can be organized as per the following sequences:

- Plotting
- Graphic applications (such as effect ordering, large datasets and trees, and graphs)
- Graphics systems
- Devices
- Colors
- Interactive graphics
- Development

A detailed explanation of the various libraries supporting the previous functionalities can be found in the link `https://cran.r-project.org/web/views/Graphics.html`. For good visual object, we need more data points so that the density of the graphs can be more. In this context, we will be using two datasets, `diamonds.csv` and `cars93.csv`, to show data visualization.

Data visualization using ggplot2

There are two approaches to go ahead with data visualization, horizontally and vertically. Horizontal drill down means creating different charts and graphs using `ggplot2` and vertical drill down implies creating one graph and adding different components to the graph. First, we will understand how to add and modify different components to a graph, and then we will move horizontally to create different types of charts.

Let's look at the dataset and libraries required to create data visualization:

```
> #getting the library
> library(ggplot2);head(diamonds);names(diamonds)
X carat cut color clarity depth table price x y z
1 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43
2 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31
3 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31
4 4 0.29 Premium I VS2 62.4 58 334 4.20 4.23 2.63
5 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75
6 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48
[1] "X" "carat" "cut" "color" "clarity" "depth" "table" "price" "x"
[10] "y" "z"
```

The `ggplot2` library is also known as the grammar of graphics for data visualization. The process to start the graph requires a dataset and two variables, and then different components of a graph can be added to the base graph by using the `+` sign. Let's get into creating a nice visualization using the `diamonds.csv` dataset:

```
> #starting a basic ggplot plot object
> gg<-ggplot(diamonds,aes(price,carat))+geom_point(color="brown4")
> gg
```

In the preceding script, `diamonds` is the dataset, and `carat` and `price` are the two variables. Using `ggplot` function, the base graph is created and adding point to the `ggplot`, the object is stored in object `gg`.

Now we are going to add various components of a graph to make the `ggplot` graph more interesting:

After creating the base plot, it is required to add title and label to the graph. This can be done using two functions, `ggtitle` or `labs`. Then, let's add a theme to the plot for customizing text element:

```
> #adding a title or label to the graph
> gg<-gg+ggtitle("Diamond Carat & Price")
> gg
> gg<-gg+labs("Diamond Carat & Price")
> gg
> #adding theme to the plot
> gg<-gg+theme(plot.title= element_text(size = 20, face = "bold"))
> gg
```

Currently, the graph looks a little congested. To make the graph more intuitive, we need to add labels to the *x* axis and *y* axis, removing ticks and text from any axis to make the graph more clear. Rotating text in any axis is required when the row name or the column name contains text or a large number which is difficult to read in full length:

```
> #adding labels to the graph
> gg<-gg+labs(x="Price in Dollar", y="Carat", )
> gg
> #removing text and ticks from an axis
> gg<-gg+theme(axis.ticks.y=element_blank(),axis.text.y=element_blank())
> gg
> gg<-gg + theme(axis.text.x=element_text(angle=50, size=10, vjust=0.5))
> gg
> gg<-gg + theme(axis.text.x=element_text(color = "chocolate", vjust=0.45),
+ axis.text.y=element_text(color = "brown1", vjust=0.45))
> gg
```

In order to focus on any specific portion of the plot, the *x* axis limit and *y* axis limit can be changed as follows. It also shows the number of rows removed while executing the limit on both the axes:

```
> #setting limits to both axis
> gg<-gg + ylim(0,0.8)+xlim(250,1500)
> gg
Warning message:
Removed 33937 rows containing missing values (geom_point).
```

If both *x* axis and *y* axis represent continuous data, any third variable as a factor can be introduced to the `ggplot` object to set legends and look at the data how it is distributed across the `factor` variable:

```
> #how to set legends in a graph
> gg<-ggplot(diamonds,aes(price,carat,color=factor(cut)))+geom_point()
> gg
> gg<-ggplot(diamonds,aes(price,carat,color=factor(color)))+geom_point()
> gg
```

```
> gg<-ggplot(diamonds,aes(price,carat,color=factor(clarity)))+geom_point()
> gg
> gg<-gg+theme(legend.title=element_blank())
> gg
```

```
> gg<-gg+theme(legend.title = element_text(colour="darkblue", size=16,
+ face="bold"))+scale_color_discrete(name="By Different Grids of Clarity")
> #changing the backgroup boxes in legend
> gg<-gg+theme(legend.key=element_rect(fill='dodgerblue1'))
> gg
> #changing the size of the symbols used in legend
> gg<-gg+guides(colour = guide_legend(override.aes = list(size=4)))
> gg
> #changing the size of the symbols used in legend
> gg<-gg+guides(colour = guide_legend(override.aes = list(size=4)))
> gg
```

In addition to the previous visualization, it is required to connect the scatterplots and change the background. Add lines to the scatterplot in order to understand the sequence *r* pattern that exists between the variables which are related:

```
> #adding line to the data points
> gg<-gg+geom_line(color="darkcyan")
> gg
> #changing the background of an image
> gg<-gg+theme(panel.background = element_rect(fill = 'chocolate3'))
> gg
> #changing plot background
> gg<-gg+theme(plot.background = element_rect(fill = 'skyblue'))
> gg
```

Another important aspect of data visualization is how to display multi-dimensional cuts in a plot. For example, in the `diamonds` datasets, we are currently looking at the relationship between the price of the diamond and the carat it contains. There are three more variables: `cut`, `color`, and `clarity`. It makes sense to understand if the relationship is consistent across those three variables. That means, understanding if we can plot the relationship between the carat and the price of the diamond by different cut, different color, and different clarity categories. Let's look at the distribution of the three categorical variables:

```
> table(diamonds$cut);table(diamonds$clarity);table(diamonds$color)
Fair Good Very Good Premium Ideal
1610 4906 12082 13791 21551
I1 SI2 SI1 VS2 VS1 VVS2 VVS1 IF
741 9194 13065 12258 8171 5066 3655 1790
D E F G H I J
6775 9797 9542 11292 8304 5422 2808
> #adding a multi-variable cut to the graph
> gg<-gg+facet_wrap(~cut, nrow=4)
> gg
```

In the preceding graph, the `cut` variable is used to show the relationship between the carat and the price. The number of rows selected is four, to represent the graphs in a clear manner. If we add one more variable to the `cut` variable, that is `clarity`, the graph would be more intuitive and insightful:

```
> #adding two variables as cut to display the relationship
> gg<-gg+facet_wrap(~cut+clarity, nrow=4)
> gg
```

While creating the graphs using a multi-dimensional `cut` variable, it is not necessary that all the graphs would be on the same scale. In automatic mode, the scales become a standard for all the plots, hence, sometimes certain plots get compressed. Thus, it is required to make the graphs scale free in order to rearrange the scales based on observed values:

```
> #scale free graphs in multi-panels
> gg<-gg+facet_wrap(~color, ncol=2, scales="free")
> gg
```

Using the `facet_grid()` option, we can display the bi-variate relationship between two categorical variables using the `ggplot2` library:

```
> #bi-variate plotting using ggplot2
> gg<-gg+facet_grid(color~cut)
> gg
```

There are certain external graphical themes which can be imported to the `ggplot2` function for visualization, such as `library(ggthemes)`. Tableau, which is a tool known for data visualization, its color, and themes, can also be used along with ggplots:

```
> #changing discrete category colors
> ggplot(diamonds, aes(price, carat, color=factor(cut)))+
+ geom_point() +
+ scale_color_brewer(palette="Set1")
> #Using tableau colors
> library(ggthemes)
> ggplot(diamonds, aes(price, carat, color=factor(cut)))+
+ geom_point() +
+ scale_color_tableau()
```

Plots created can be slightly modified using the `color` gradient and plotting a distribution on the graph itself:

```
> #using color gradient
> ggplot(diamonds, aes(price, carat))+
+ geom_point() +
+ scale_color_gradient(low = "blue", high = "red")
> #plotting a distribution on a graph
> mid<-mean(diamonds$price)
> ggplot(diamonds, aes(price, carat, color=depth))+geom_point()+
+ scale_color_gradient2(midpoint=mid,
+ low="blue", mid="white", high="red" )
```

Having discussed in depth about the components in a graph building process, now let's try to understand how to create different charts and graphs using `ggplot2`. `qplot()` is a basic plotting function in `ggplot2`, which is a wrapper for creating different types of plots. There are two options for a user, either go plotting with the `qplot()` or `ggplot()` function. To create different graphs, we are going to use the `Cars93.csv` dataset.

Bar chart

Bar charts are preferred as a method of visualization for the categorical variables, also used to represent the count or percentage of each group. The horizontal axis represents the categories and the vertical axis either represents the count or the percentage:

```
> #creating bar chart
> barplot <- ggplot(Cars93,aes(Type))+
+ geom_bar(width = 0.5,fill="royalblue4",color="red")+
+ ggtitle("Vehicle Count by Category")
> barplot
```

Boxplot

It is not only easy to interpret boxplots using the `ggplot` package, but also easy to customize the plot. One can easily recognize the outliers imposed on each of the corresponding boxplots:

```
> #creating boxplot
> boxplot <- ggplot(Cars93,aes(Type,Price))+
+ geom_boxplot(width = 0.5,fill="firebrick",color="cadetblue2",
+ outlier.colour = "purple",outlier.shape = 2)+
+ ggtitle("Boxplot of Price by Car Type")
> boxplot
```

Bubble chart

The bubble chart belongs to the family of the scatterplot. It is preferred when it is required to represent three quantitative variables. Two quantitative variables are represented on two axis and one quantitative variable is used to represent the size of each bubble in a bubble chart:

```
> #creatting Bubble chart
> bubble<-ggplot(Cars93, aes(x=EngineSize, y=MPG.city)) +
+ geom_point(aes(size=Price,color="red")) +
+ scale_size_continuous(range=c(2,15)) +
+ theme(legend.position = "bottom")
> bubble
```

Donut chart

Donut chart is used in place of pie chart when the number of categories exceeds five:

```
> #creating Donut charts
> ggplot(Cars93) + geom_rect(aes(fill=Cylinders, ymax=Max.Price,
+ ymin=Min.Price, xmax=4, xmin=3)) +
+ coord_polar(theta="y") + xlim(c(0, 4))
```

Geo mapping

Any dataset that has a city name and state name or a country name can be plotted on a geographical map using google visualization library in R. Using another open source dataset, `state.x77` inbuilt in R, we can show how a geo map looks. The google visualization library, using Google maps API, tries to plot the geographic locations on the plot along with the enterprise data. It publishes the output in a browser which can be stored back as an image to use it further:

```
> library(googleVis)
> head(state.x77)
Population Income Illiteracy Life Exp Murder HS Grad Frost
Alabama 3615 3624 2.1 69.05 15.1 41.3 20
Alaska 365 6315 1.5 69.31 11.3 66.7 152
Arizona 2212 4530 1.8 70.55 7.8 58.1 15
Arkansas 2110 3378 1.9 70.66 10.1 39.9 65
California 21198 5114 1.1 71.71 10.3 62.6 20
```

```
Colorado 2541 4884 0.7 72.06 6.8 63.9 166
Area
Alabama 50708
Alaska 566432
Arizona 113417
Arkansas 51945
California 156361
Colorado 103766
> states <- data.frame(state.name, state.x77)
> gmap <- gvisGeoMap(states, "state.name", "Area",
+ options=list(region="US", dataMode="regions",
+ width=900, height=600))
> plot(gmap)
```

Histogram

This is probably the easiest plot that every data mining professional must be doing. The following code explains how a histogram can be created using the `ggplot` library:

```
> #creating histograms
> histog <- ggplot(Cars93,aes(RPM))+
+ geom_histogram(width = 0.5,fill="firebrick",color="cadetblue2",
+ bins = 20)+
+ ggtitle("Histogram")
> histog
```

Line chart

Line chart is not a preferred chart while showing raw data. However, it is important while showing some variations across different categories relating to some metric. Though it is not a preferred chart, but it depends on the practitioner how he/she wants to display and tell a story to the reader:

```
> #creating line charts
> linechart <- ggplot(Cars93,aes(RPM,Price))+
+ geom_line(color="cadetblue4")+
+ ggtitle("Line Charts")
>
> linechart
```

Pie chart

A pie chart is a representation of categorical variables when the label for each categorical variable is less than 10. If it exceeds 10, then it is suggested to look at a histogram or barplot for comparison. Using the `ggplot` library, the pie chart can be created. The script is as follows:

```
> #creating pie charts
> pp <- ggplot(Cars93, aes(x = factor(1), fill = factor(Type))) +
+ geom_bar(width = 1)
> pp + coord_polar(theta = "y")
```

```
> # 3D Pie Chart from data frame
> library(plotrix)
> t <- table(Cars93$Type);par(mfrow=c(1,2))
> pct <- paste(names(t), "\n", t, sep="")
> pie(t, labels = pct, main="Pie Chart of Type of cars")
> pie3D(t,labels=pct,main="Pie Chart of Type of cars")
```

Scatterplot

Scatterplot is a very important plot to understand the bivariate relationship that exists in data. It also shows the pattern in which the data is stored over a period of time. It is also important to show the data in a proper way while showing it in scatterplots. The following example shows how a bivariate relationship can be displayed along with some third dimension dictating the visualization in a bivariate relationship. The third dimension could be a continuous variable or a categorical variable. Using the `gridExtra()` library, additional graphing window can be created where two or more plots can be represented side by side, with some relationship:

```
> library(gridExtra)
> sp <- ggplot(Cars93,aes(Horsepower,MPG.highway))+
+ geom_point(color="dodgerblue",size=5)+ggtitle("Basic Scatterplot")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp
> #adding a cantinuous variable Length to scale thee scatterplot points
> sp2<-sp+geom_point(aes(color=Length), size=5)+
+ ggtitle("Scatterplot: Adding Length Variable")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp2
>
> grid.arrange(sp,sp2,nrow=1)
```

In the second graph in the preceding plot, the length variable, which is continuous, is dictating the relationship between the horsepower and the highway mileage per gallon. The light blue colored dots indicate lengthy cars while the darker dots indicate smaller cars. Instead of a continuous variable, if we use a factor variable to scale the relationship between the two variables, we will be able to see a plot like given in first graph in the following plot:

```
> #adding a factor variable Origin to scale the scatterplot points
> sp3<-sp+geom_point(aes(color=factor(Origin)),size=5)+
+ ggtitle("Scatterplot: Adding Origin Variable")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp3
> #adding custom color to the scatterplot
> sp4<-sp+geom_point(aes(color=factor(Origin)),size=5)+
+ scale_color_manual(values = c("red","blue"))+
+ ggtitle("Scatterplot: Adding Custom Color")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp4
> grid.arrange(sp3,sp4,nrow=1)
```

To display the cause and effect relationship, one needs to display a trendline or a regression line on a scatterplot. Using the `ggplot2` library, different regression lines can be plotted, such as linear, non linear, generalized linear, and so on. When the number of observations in a dataset is less than 1000, the loess regression method is applied by default. However, when it is more than 1000, the generalized additive model is applied. The trend lines are displayed next. The first plot indicates a line graph connecting all the points, and the second plot indicates the robust linear model:

```
> sp5<-sp+geom_point(color="blue",size=5)+geom_line()+
+ ggtitle("Scatterplot: Adding Lines")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
> sp5
> #adding regression lines to the scatterplot
> sp6<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "lm",se =T)+
+ geom_smooth(method = "rlm",se =T)+
+ ggtitle("Adding Regression Lines")+
+ theme(plot.title= element_text(size = 12, face = "bold"))
```

```
> sp6
> grid.arrange(sp5,sp6,nrow=1)
```

Adding the generalized regression model and loess as a non-linear regression model, we can modify the scatterplots as follows:

```
> sp7<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "auto",se =T)+
+ geom_smooth(method = "glm",se =T)+
+ ggtitle("Adding Regression Lines")+
+ theme(plot.title= element_text(size = 20, face = "bold"))
> sp7
> #adding regression lines to the scatterplot
> sp8<-sp+geom_point(color="firebrick",size=5)+
+ geom_smooth(method = "gam",se =T)+
+ ggtitle("Adding Regression Lines")+
+ geom_smooth(method = "loess",se =T)+
+ theme(plot.title= element_text(size = 20, face = "bold"))
> sp8
> grid.arrange(sp7,sp8,nrow=1)
```

3D scatterplot is another addition to the list of scatterplot functions we are looking at. The 3D scatterplot library enables the users to look at the plot and rotate it, to view the data points from different angles. Once executed, the following script would open up a new rgl device window. Just rotate the graph and you would be able to see the data points from different angles:

```
> library(scatterplot3d);library(Rcmdr)
> scatter3d(MPG.highway~Length+Width|Origin, data=Cars93,
fit="linear",residuals=TRUE, parallel=FALSE, bg="black", axis.scales=TRUE,
grid=TRUE, ellipsoid=FALSE)
```

Stacked bar chart

Stacked bar charts are just another variant of bar charts, where more than two variables can be plotted with different combinations of color. The following example codes show some variants of stacked bar charts:

```
> qplot(factor(Type), data=Cars93, geom="bar", fill=factor(Origin))
>
> #or
>
> ggplot(Cars93, aes(Type, fill=Origin)) + geom_bar()
```

Stem and leaf plot

A stem and leaf plot is a textual representation of a quantitative variable that segments the values to their most significant numeric digits. For example, the stem and leaf plot for the mileage within the `city` variable from the `Cars93.csv` dataset is represented as follows:

```
> stem(Cars93$MPG.city)
The decimal point is 1 digit(s) to the right of the |
1 | 556667777777788888888888999999999
2 | 00000000111111222222233333333344444
2 | 5555556688999999
3 | 01123
3 | 9
4 | 2
4 | 6
```

To interpret the results of a stem and leaf plot: if we need to know how many observations are there which are greater than 30, the answer is 8, the digit on the left of pipe indicates items and the numbers on the right indicate units, hence the respective numbers are 30, 31, 31, 32, 33, 39, 42, 46.

Word cloud

Word cloud is a data visualization method which is preferred when it is required to represent a textual data. For example, the representation of a bunch of text files with few words having frequent appearances across those set of documents would summarize the topic of discussion. Hence, word cloud representation is a visual summary of the textual unstructured data. This is mostly used to represent social media posts, such as Twitter tweets, Facebook posts, and so on. There are various pre-processing tasks before arriving to create a word cloud, the final output from a text mining exercise would be a data frame with words and their respective frequencies:

```
#Word cloud representation
library(wordcloud)
words<-c("data","data mining","analytics","statistics","graphs",
"visualization","predictive analytics","modeling","data science",
"R","Python","Shiny","ggplot2","data analytics")
freq<-c(123,234,213,423,142,145,156,176,214,218,213,234,256,324)
d<-data.frame(words,freq)
set.seed(1234)
wordcloud(words = d$words, freq = d$freq, min.freq = 1,c(8,.3),
max.words=200, random.order=F, rot.per=0.35,
colors=brewer.pal(7, "Dark2"))
```

Coxcomb plot

The coxcomb chart, which is also known as polar chart or rose chart, is a combination of pie chart and bar chart. The area of each section is adjusted based on the values of that segment by changing the radius. Anyone can understand the insights represented using coxcomb chart and does not require any technical knowledge:

```
> #coxcomb chart = bar chart + pie chart
> cox<- ggplot(Cars93, aes(x = factor(Type))) +
+ geom_bar(width = 1, colour = "goldenrod1",fill="darkviolet")
> cox + coord_polar()
```

A new variant of coxcomb plot by changing the coordinate polar measure, which is theta:

```
> #coxcomb chart = bar chart + pie chart
> cox<- ggplot(Cars93, aes(x = factor(Type))) +
+ geom_bar(width = 1, colour = "goldenrod1",fill="darkred")
> cox + coord_polar()
> #a second variant of coxcomb plot
> cox + coord_polar(theta = "y")
```

Using plotly

So far, we have looked at various scenarios of creating plots using the `ggplot2` library. In order to take the plotting to a new level, there are many libraries which can be referred to. Out of them, one library is `plotly`, which is designed as an interactive browser-based charting library built on the JavaScript library. Let's look at a few examples on how it works.

Bubble plot

Bubble plot is a nice visualization in which the size of the bubble indicates the weight of each variable as it is present in the dataset. Let's look at the following plot:

```
> #Bubble plot using plotly
> plot_ly(Cars93, x = Length, y = Width, text = paste("Type: ", Type),
+ mode = "markers", color = Length, size = Length)
```

The combination of `ggplot2` with the `plotly` library makes for good visualization, as the features of both the libraries are embedded with the `ggplotly` library:

```
> #GGPLOTLY: ggplot plus plotly
> p <- ggplot(data = Cars93, aes(x = Horsepower, y = Price)) +
+ geom_point(aes(text = paste("Type:", Type)), size = 2,
color="darkorchid4") +
+ geom_smooth(aes(colour = Origin, fill = Origin)) + facet_wrap(~ Origin)
>
> (gg <- ggplotly(p))
```

Bar charts using plotly

Bar charts using `plotly` look smarter than regular bar charts available in R, as a base functionality, let's have a look at the graph as follows:

```
> p <- plot_ly(
+ x = Type,
+ y = Price,
+ name = "Price by Type",
+ type = "bar")
> p
```

Scatterplot using plotly

Representation of two continuous variables can be shown using a scatterplot. Let's look at the data represented next:

```
> # Simple scatterplot
> library(plotly)
> plot_ly(data = Cars93, x = Horsepower, y = MPG.highway, mode = "markers")
> #Scatter Plot with Qualitative Colorscale
> plot_ly(data = Cars93, x = Horsepower, y = MPG.city, mode = "markers",
+ color = Type)
```

Boxplots using plotly

Following are examples of creating some interesting boxplots using the **plotly** library:

```
> #Box Plots
> library(plotly)
> ### basic boxplot
> plot_ly(y = MPG.highway, type = "box") %>%
+ add_trace(y = MPG.highway)
```

```
> ### adding jittered points
> plot_ly(y = MPG.highway, type = "box", boxpoints = "all", jitter = 0.3,
+ pointpos = -1.8)
```

```
> ### several box plots
> plot_ly(Cars93, y = MPG.highway, color = Type, type = "box")
```

```
> ### grouped box plots
> plot_ly(Cars93, x = Type, y = MPG.city, color = AirBags, type = "box")
%>%
+ layout(boxmode = "group")
```

Polar charts using plotly

The polar chart visualization using `plotly` is more interesting to look at. This is because when you hover your cursor over the chart, the data values become visible and differences in pattern can be recognized. Let's look at the example code as given next:

```
> #Polar Charts in R
> library(plotly)
pc <- plot_ly(Cars93, r = Price, t = RPM, color = AirBags,
mode = "lines",colors='Set1')
layout(pc, title = "Cars Price by RPM", orientation = -90,
font='bold')
```

Polar scatterplot using plotly

Using the `plotly` library, there is one additional chart type which a user can create. It's called polar scatterplot. Instead of a two dimensional scatterplot, the points are plotted in a circular fashion:

```
> #Polar Scatter Chart
pc <- plot_ly(Cars93, r = Price, t = Horsepower, color = Type,opacity =
0.7,
mode = "markers",colors = 'Dark2')
layout(pc, title = "Price of Cars by Horsepower",plot_bgcolor =
toRGB("coral"),
font='bold')
```

The graph polar scatterplot shows the price of cars by horsepower and the colors indicate the type of cars, the rings indicate proximity or closeness, one midsize car is very distinctly different from other variants as observed from the graph.

Polar area chart

The polar area chart, also known as the radar chart or coxcomb chart as renamed in some other packages, is shown next using a sample code. It shows the relationship between the of cars and horsepower by car type:

```
> #Polar Area Chart
pc <- plot_ly(Cars93, r = Price, t = Horsepower, color = Type, type =
"area")
layout(pc,title = "Price of Cars by Horsepower",orientation = 270,
plot_bgcolor = toRGB("tan"),font='bold')
```

Creating geo mapping

Geo mapping is a type of chart which is used by data mining experts when the dataset contains location information. The geo mapping plots are supported by the `ggmap` library. The location information can be accessed in three different ways:

- By the name of the place, location name, and address
- By the latitude and longitude of the place
- By exact location, lower left longitude, lower left latitude, upper right longitude, and upper left latitude

Once the map location is identified, then by using the `ggmap` function the location can be identified on a map:

```
>library(ggmap)
>gc <- geocode("statue of liberty", source = "google")
>googMap <- get_googlemap(center = as.numeric(gc))
>(bb <- attr(googMap, "bb"))
>bb2bbox(bb)
>gc<-get_map(location = c(lon = gc$lon, lat = gc$lat))
>ggmap(gc)
```

Summary

In this chapter, we looked at the various types of charts. We briefly discussed the syntax to create those charts and we also discussed briefly where to use which type of chart. To create a visual display to explain the insights and message to the audience is a skill. It takes time and becomes perfect by experience. In this chapter, we only looked at the most important data visualization methods used in data mining domain. However, there is overabundance of graphs and charts to be selected to create innovative presentations. So the key take away from this chapter is that the reader now knows the data visualization rules, the types of charts and graphs used in data mining to show the relationship between various variables, and understands the distribution of various variables. At the same time, the reader has got hands on experience by doing side by side practice on two important data visualization libraries: `ggplot2` and `plotly`. In the next chapter, we are going to learn about application of various regression techniques, interpretation of regression results, and visualization of regression results to understand the relationship between various variables.

4

Regression with Automobile Data

One of the primary objectives of data mining projects is to understand the relationship between various variables and establish a cause and effect relationship between the variable of interest and other explanatory variables. In data mining projects, performing predictive analytics not only entails insights on hidden messages in datasets but also helps in making future decisions which might impact the business outcomes. In this chapter, the reader will get to know the basics of predictive analytics using regression methods, including various linear and nonlinear regression methods using R programming. The reader will be able to understand the theoretical background as well as get practical hands-on experience with all the regression methods using R programming language.

In this chapter, we will cover the following topics:

- Regression formulation
- Linear regression
- Logical regression
- Cubic regression
- Stepwise regression
- Penalized regression

Regression introduction

Regression methods help in predicting the future outcomes of a target variable. As an example, here are a few business cases:

- In the sales and marketing domain, how can a business achieve a significant improvement in sales? Can we successfully predict using the if a necessary amendment is made to the drivers of sales?
- In the retail domain, can we predict the number of visitors to a website, so that necessary tech support can be aligned to help operate the site better?
- How can a retail/e-commerce owner predict the number of footfall to their store in a month/week/year?
- In the banking domain, how can a bank predict the number of people applying for home loans, car loans, and personal loans, so that they can maintain their liquid capital to support the demand?
- In the banking domain, prediction of default probability can be estimated using regression methods so that a bank can decide whether to approve a loan/credit card to a customer.
- In the automobile manufacturing domain, the sales unit of vehicles is indirectly proportional to the price of the vehicles and the price of a vehicle is decided by many factors such as usages of different metals/components, and various vehicle features such as RPM, mileage, length, width, and others. How can a manufacturer predict the unit of sales?

There are different methods to perform regression, including both linear as well as non-linear. Regression-based predictive analytics is being used in different industries in a different way. Regression methods support the prediction of a continuous variable, prediction of the probability of a success or failure of a variable, prediction of events based on features, and so on.

Formulation of regression problem

The formulation of a regression problem is very essential in creating a good regression-based predictive model. A typical approach in building a good predictive model follows a process of converting a business problem to a statistical problem, then converting the statistical problem to a statistical solution, and finally converting the statistical solution to a business solution. The following steps are required to build a good predictive model using regression based methods:

- A clear understanding about the background or context is very much required. Sometimes a good predictive model does not make sense for a business from a context point of view. However, a contextually relevant predictive model may not be a good predictive model.
- A clear understanding about the objective is needed: what are you predicting and why are you predicting? Domain expertise is required. Most of the times, non-relevant features get added to a model, having no practical sense, because they show a mere correlation.
- A clear understanding about correlation and regression is important. It's a common phenomenon that people misconstrue a mere correlation or association as regression. "All regression may show causal relationship, but not the contrary".
- Conversion of a business problem to a statistical problem should be done carefully, so that assumptions and business understanding can be taken care of in the model.

Initial exploratory data analysis reveals the relationship between variables so that the variables can be included in a predictive model. The exploratory data analysis involves univariate, bivariate, and multivariate data analysis. Missing value imputation, outlier treatment, and removal of data entry errors are also equally important before proceeding with regression methods.

Case study

We are going to take two datasets, `Cars93_1.csv` and `ArtPiece_2.csv`, to explain various regression methods with a detailed analysis of what regression to use where and under what circumstances. For each regression method, we will take a look at the assumptions, limitations, mathematical formulations, and interpretation of the results.

Linear regression

The linear regression model is used for explaining the relationship between a single dependent variable known as Y and one or more $X's$ independent variables known as input, predictor, or independent variables. The following two equations explain a linear regression scenario:

$$Y = f(X) + \varepsilon .....(1)$$
$$Y = f(X_1, X_2 ..... X_n) + \varepsilon .....(2)$$

Equation 1 shows the formulation of a simple linear regression equation and equation 2 shows the multiple linear equations where many independent variables can be included. The dependent variable must be a continuous variable and the independent variables can be continuous and categorical. We are going to explain the multiple linear regression analysis by taking the `Cars93_1.csv` dataset, where the dependent variable is the price of cars and all the other variables are explanatory variables. The regression analysis includes:

- Predicting the future values of the dependent variable, given the values for independent variables
- Assessment of model fit statistics and comparison of various models
- Interpreting the coefficients to understand the levers of change in the dependent variable
- The relative importance of the independent variables

The primary objective of a regression model is to estimate the beta parameters and minimize the error term epsilon:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 ..... \beta_n X_n) + \varepsilon .....(3)$$

Let's look at the relationship between various variables using a correlation plot:

```
#Scatterplot showing all the variables in the dataset
library(car);attach(Cars93_1)
> #12-"Rear.seat.room",13-"Luggage.room" has NA values need to remove them
> m<-cor(Cars93_1[,-c(12,13)])
> corrplot(m, method = "ellipse")
```

The preceding figure shows the relationship between various variables. From each ellipse we can get to know how the relationship: is it positive or negative, and is it strongly associated, moderately associated, or weakly associated? The correlation plot helps in taking a decision on which variables to keep for further investigation and which ones to remove from further analysis. The dark brown colored ellipse indicates perfect negative correlation and the dark blue colored ellipse indicates the perfect positive correlation. The narrower the ellipse, the higher the degree of correlation.

Given the number of variables in the dataset, it is quite difficult to read the scatterplot. Hence the correlation between all the variables can be represented using a correlation graph, as follows:

Correlation coefficients represented in bold blue color show higher degree of positive correlation and the ones with red color show higher degree of negative correlation.

The assumptions of a linear regression are as follows:

- **Normality**: The error terms from the estimated model should follow a normal distribution
- **Linearity**: The parameters to be estimated should be linear
- **Independence**: The error terms should be independent, hence no-autocorrelation
- **Equal variance**: The error terms should be homoscedastic
- **Multicollinearity**: The correlation between the independent variables should be zero or minimum

While fitting a multiple linear regression model, it is important to take care of the preceding assumptions. Let's look at the summary statistics from the multiple linear regression analysis and interpret the assumptions:

```
> #multiple linear regression model
> fit<-lm(MPG.Overall~.,data=Cars93_1)
> #model summary
> summary(fit)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1)
```

```
Residuals:
Min 1Q Median 3Q Max
-5.0320 -1.4162 -0.0538 1.2921 9.8889
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.808773 16.112577 0.174 0.86213
Price -0.053419 0.061540 -0.868 0.38842
EngineSize 1.334181 1.321805 1.009 0.31638
Horsepower 0.005006 0.024953 0.201 0.84160
RPM 0.001108 0.001215 0.912 0.36489
Rev.per.mile 0.002806 0.001249 2.247 0.02790 *
Fuel.tank.capacity -0.639270 0.262526 -2.435 0.01752 *
Length -0.059862 0.065583 -0.913 0.36459
Wheelbase 0.330572 0.156614 2.111 0.03847 *
Width 0.233123 0.265710 0.877 0.38338
Turn.circle 0.026695 0.197214 0.135 0.89273
Rear.seat.room -0.031404 0.182166 -0.172 0.86364
Luggage.room 0.206758 0.188448 1.097 0.27644
Weight -0.008001 0.002849 -2.809 0.00648 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.835 on 68 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.7533, Adjusted R-squared: 0.7062
F-statistic: 15.98 on 13 and 68 DF, p-value: 7.201e-16
```

In the previous multiple linear regression model using all the independent variables, we are predicting the mileage per gallon **MPG.Overall** variable. From the model summary, it is observed that few independent variables are statistically significant at 95% confidence level. The coefficient of determination R^2 which is known as the goodness of fit of a regression model is 75.33%. This implies that 75.33% variation in the dependent variable is explained by all the independent variables together. The formula for computing multiple R^2 is given next:

$$\boxed{R^2 = 1 - \frac{\sum\left(\hat{y}_i - y_i\right)}{\sum\left(y_i - \overline{y}_i\right)^2} \ldots\ldots (4)}$$

The preceding equation 4 calculated the coefficient of determination or percentage of variance explained by the regression model. The base line score to qualify for a good regression model is at least 80% as R square value; any R square value more than 80% is considered to be very good regression model. Since the R^2 is now less than 80%, we need to perform a few diagnostic tests on the regression results.

The estimated beta coefficients of the model:

```
> #estimated coefficients
> fit$coefficients
(Intercept) Price EngineSize Horsepower
2.808772930 -0.053419142 1.334180881 0.005005690
RPM Rev.per.mile Fuel.tank.capacity Length
0.001107897 0.002806093 -0.639270186 -0.059861997
Wheelbase Width Turn.circle Rear.seat.room
0.330572119 0.233123382 0.026694571 -0.031404262
Luggage.room Weight
0.206757968 -0.008001444
#residual values
fit$residuals
#fitted values from the model
fit$fitted.values
#what happened to NA
fit$na.action
> #ANOVA table from the model
> summary.aov(fit)
Df Sum Sq Mean Sq F value Pr(>F)
Price 1 885.7 885.7 110.224 7.20e-16 ***
EngineSize 1 369.3 369.3 45.959 3.54e-09 ***
Horsepower 1 37.4 37.4 4.656 0.03449 *
RPM 1 38.8 38.8 4.827 0.03143 *
Rev.per.mile 1 71.3 71.3 8.877 0.00400 **
Fuel.tank.capacity 1 147.0 147.0 18.295 6.05e-05 ***
Length 1 1.6 1.6 0.203 0.65392
Wheelbase 1 35.0 35.0 4.354 0.04066 *
Width 1 9.1 9.1 1.139 0.28969
Turn.circle 1 0.5 0.5 0.060 0.80774
Rear.seat.room 1 0.6 0.6 0.071 0.79032
Luggage.room 1 9.1 9.1 1.129 0.29170
Weight 1 63.4 63.4 7.890 0.00648 **
Residuals 68 546.4 8.0
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
11 observations deleted due to missingness
```

From the preceding **analysis of variance** (**ANOVA**) table, it is observed that the variables `Length`, `Width`, turn circle, rear seat room, and luggage room are not statistically significant at 5% level of alpha. ANOVA shows the source of variation in dependent variable from any independent variable. It is important to apply ANOVA after the regression model to know which independent variable has significant contribution to the dependent variable:

```
> #visualizing the model statistics
> par(mfrow=c(1,2))
> plot(fit, col="dodgerblue4")
```

The residual versus fitted plot shows the randomness in residual values as we move along the fitted line. If the residual values display any pattern against the fitted values, the error term is not probably normal. The residual versus fitted graph indicates that there is no pattern that exists among the residual terms; the residuals are approximately normally distributed. Residual is basically that part of the model which cannot be explained by the model. There are few influential data points, for example, 42nd, 39th, and 83rd as it is shown on the preceding graph, the normal quantile-quantile plot indicates except few influential data points all other standardized residual points follow a normal distribution. The straight line is the zero residual line and the red line is the pattern of residual versus fitted relationship. The scale versus location graph and the residuals versus leverage graph also validate the same observation that the residual term has no trend. The scale versus location plot takes the square root of the standardized residuals and plots it against the fitted values:

The confidence interval of the model coefficients at 95% confidence level can be calculated using the following code and also the prediction interval at 95% confidence level for all the fitted values can be calculated using the code below. The formula to compute the confidence interval is the model coefficient +/- standard error of model coefficient * 1.96:

```
> confint(fit,level=0.95)
2.5 % 97.5 %
(Intercept) -2.934337e+01 34.960919557
Price -1.762194e-01 0.069381145
EngineSize -1.303440e+00 3.971801771
Horsepower -4.478638e-02 0.054797758
RPM -1.315704e-03 0.003531499
Rev.per.mile 3.139667e-04 0.005298219
Fuel.tank.capacity -1.163133e+00 -0.115407347
Length -1.907309e-01 0.071006918
Wheelbase 1.805364e-02 0.643090600
Width -2.970928e-01 0.763339610
Turn.circle -3.668396e-01 0.420228772
Rear.seat.room -3.949108e-01 0.332102243
Luggage.room -1.692837e-01 0.582799642
Weight -1.368565e-02 -0.002317240
> head(predict(fit,interval="predict"))
fit lwr upr
1 31.47382 25.50148 37.44615
2 24.99014 18.80499 31.17528
3 22.09920 16.09776 28.10064
```

```
4 21.19989 14.95606 27.44371
5 21.62425 15.37929 27.86920
6 27.89137 21.99947 33.78328
```

The existence of influential data points or outlier values may deviate the model result, hence identification and curation of outlier data points in the regression model is very important:

```
# Deletion Diagnostics
influence.measures(fit)
```

The function belongs to the `stats` library, which computes some of the regression diagnostics for linear and generalized models. Any observation labeled with a star (`*`) implies an influential data point, that can be removed to make the model good:

```
# Index Plots of the influence measures
influenceIndexPlot(fit, id.n=3)
```

The influential data points are highlighted by their position in the dataset. In order to understand more about these influential data points, we can write the following command. The size of the red circle based on the cook's distance value indicates the order of influence. Cook's distance is a statistical method to identify data points which have more influence than other data points.

Generally, these are data points that are distant from other points in the data, either for the dependent variable or one or more independent variables:

```
> # A user friendly representation of the above
> influencePlot(fit,id.n=3, col="red")
StudRes Hat CookD
28 1.9902054 0.5308467 0.55386748
39 3.9711522 0.2193583 0.50994280
42 4.1792327 0.1344866 0.39504695
59 0.1676009 0.4481441 0.04065691
60 -2.1358078 0.2730012 0.34097909
77 -0.6448891 0.3980043 0.14074778
```

If we remove these influential data points from our model, we can see some improvement in the model goodness of fit and overall error for the model can be reduced. Removing all the influential points at one go is not a good idea; hence we will take a step-by-step approach to deleting these influential data points from the model and monitor the improvement in the model statistics:

```
> ## Regression after deleting the 28th observation
> fit.1<-lm(MPG.Overall~., data=Cars93_1[-28,])
> summary(fit.1)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1[-28, ])
Residuals:
Min 1Q Median 3Q Max
-5.0996 -1.7005 0.4617 1.4478 9.4168
```

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.314222 16.425437 -0.384 0.70189
Price -0.014859 0.063281 -0.235 0.81507
EngineSize 2.395780 1.399569 1.712 0.09156 .
Horsepower -0.022454 0.028054 -0.800 0.42632
RPM 0.001944 0.001261 1.542 0.12789
Rev.per.mile 0.002829 0.001223 2.314 0.02377 *
Fuel.tank.capacity -0.640970 0.256992 -2.494 0.01510 *
Length -0.065310 0.064259 -1.016 0.31311
Wheelbase 0.407332 0.158089 2.577 0.01219 *
Width 0.204212 0.260513 0.784 0.43587
Turn.circle 0.071081 0.194340 0.366 0.71570
Rear.seat.room -0.004821 0.178824 -0.027 0.97857
Luggage.room 0.156403 0.186201 0.840 0.40391
Weight -0.008597 0.002804 -3.065 0.00313 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.775 on 67 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.7638, Adjusted R-squared: 0.718
F-statistic: 16.67 on 13 and 67 DF, p-value: 3.39e-16
```

Regression output after deleting the most influential data point provides a significant improvement in the R2 value from 75.33% to 76.38%. Let's repeat the same activity and see the model's results:

```
> ## Regression after deleting the 28,39,42,59,60,77 observations
> fit.2<-lm(MPG.Overall~., data=Cars93_1[-c(28,42,39,59,60,77),])
> summary(fit.2)
Call:
lm(formula = MPG.Overall ~ ., data = Cars93_1[-c(28, 42, 39,
59, 60, 77), ])
Residuals:
Min 1Q Median 3Q Max
-3.8184 -1.3169 0.0085 0.9407 6.3384
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 21.4720002 13.3375954 1.610 0.11250
Price -0.0459532 0.0589715 -0.779 0.43880
EngineSize 2.4634476 1.0830666 2.275 0.02641 *
Horsepower -0.0313871 0.0219552 -1.430 0.15785
RPM 0.0022055 0.0009752 2.262 0.02724 *
Rev.per.mile 0.0016982 0.0009640 1.762 0.08307 .
Fuel.tank.capacity -0.6566896 0.1978878 -3.318 0.00152 **
Length -0.0097944 0.0613705 -0.160 0.87372
Wheelbase 0.2298491 0.1288280 1.784 0.07929 .
Width -0.0877751 0.2081909 -0.422 0.67477
```

```
Turn.circle 0.0347603 0.1513314 0.230 0.81908
Rear.seat.room -0.2869723 0.1466918 -1.956 0.05494 .
Luggage.room 0.1828483 0.1427936 1.281 0.20514
Weight -0.0044714 0.0021914 -2.040 0.04557 *
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.073 on 62 degrees of freedom
(11 observations deleted due to missingness)
Multiple R-squared: 0.8065, Adjusted R-squared: 0.7659
F-statistic: 19.88 on 13 and 62 DF, p-value: < 2.2e-16
```

Looking at the preceding output, we can conclude that the removal of outliers or influential data points adds robustness to the R-square value. Now we can conclude that 80.65% of the variation in the dependent variable is being explained by all the independent variables in the model.

Now, using studentised residuals against t-quantiles in a Q-Q plot, we can identify some more outliers data points, if any, to boost the model accuracy:

```
> # QQ plots of studentized residuals, helps identify outliers
> qqPlot(fit.2, id.n=5)
91 32 55 5 83
1 2 74 75 76
```

From the preceding Q-Q plot, it seems that the data is fairly normally distributed, except few outlier values such as 83rd, 91st, 32nd, 55th, and 5th data points:

```
> ## Diagnostic Plots ###
> influenceIndexPlot(fit.2, id.n=3)
> influencePlot(fit.2, id.n=3, col="blue")
StudRes Hat CookD
5 2.1986298 0.2258133 0.30797166
8 -0.1448259 0.4156467 0.03290507
10 -0.7655338 0.3434515 0.14847534
83 3.7650470 0.2005160 0.45764995
91 -2.3672942 0.3497672 0.44770173
```

The problem of multicollinearity, that is, the correlation between predictor variables, needs to be verified for the final regression model. **Variance Inflation Factor** (**VIF**) is the measure typically used to estimate the multicollinearity. The formula to compute the VIF is 1/ (1-R^2). Any independent variable having a VIF value of more than 10 indicates multicollinearity; hence such variables need to be removed from the model. Deleting one variable at a time and then again checking the VIF for the model is the best way to do this:

```
> ### Variance Inflation Factors
> vif(fit.2)
Price EngineSize Horsepower RPM
4.799678 20.450596 18.872498 5.788160
Rev.per.mile Fuel.tank.capacity Length Wheelbase
3.736889 5.805824 15.200301 11.850645
```

```
Width Turn.circle Rear.seat.room Luggage.room
10.243223 4.006895 2.566413 2.935853
Weight
24.977015
```

Variable weight has the maximum VIF, hence deleting the variable make sense. After removing the variable, let's rerun the model and calculate the VIF:

```
## Regression after deleting the weight variable
fit.3<-lm(MPG.Overall~ Price+EngineSize+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Length+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.3)
> vif(fit.3)
Price EngineSize Horsepower RPM
4.575792 20.337181 15.962349 5.372388
Rev.per.mile Fuel.tank.capacity Length Wheelbase
3.514992 4.863944 14.574352 11.013850
Width Turn.circle Rear.seat.room Luggage.room
10.240036 3.965132 2.561947 2.935690
## Regression after deleting the Enginesize variable
fit.4<-lm(MPG.Overall~ Price+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Length+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.4)
vif(fit.4)
## Regression after deleting the Length variable
fit.5<-lm(MPG.Overall~ Price+Horsepower+RPM+Rev.per.mile+
Fuel.tank.capacity+Wheelbase+Width+Turn.circle+
Rear.seat.room+Luggage.room, data=Cars93_1[-c(28,42,39,59,60,77),])
summary(fit.5)
> vif(fit.5)
Price Horsepower RPM Rev.per.mile
4.419799 8.099750 2.595250 3.232048
Fuel.tank.capacity Wheelbase Width Turn.circle
4.679088 8.231261 7.953452 3.357780
Rear.seat.room Luggage.room
2.393630 2.894959
The coefficients from the final regression model:
> coefficients(fit.5)
(Intercept) Price Horsepower RPM
29.029107454 -0.089498236 -0.014248077 0.001071072
Rev.per.mile Fuel.tank.capacity Wheelbase Width
0.001591582 -0.769447316 0.130876817 -0.047053999
Turn.circle Rear.seat.room Luggage.room
-0.072030390 -0.240332275 0.216155256
```

Now we can conclude that there is no multicollinearity in the preceding regression model. We can write the final multiple linear regression equation as follows:

$$MPG.Overall = 29.03 - 0.09 * Price - 0.014 * Horsepower + 0.001 * RPM + 0.001 * Rev.per.mile - 0.769 * Fuel.tank.capacity + 0.131 * Wheelbase + 0.047 * Width$$

The estimated model parameters can be interpreted as, for one unit change in the price variable, the MPG overall is predicted to change by 0.09 unit. Likewise, the estimated model coefficients can be interpreted for all the other independent variables. If we know the values of these independent variables, we can predict the likely value of the dependent variable MPG overall by using the previous equation.

Stepwise regression method for variable selection

In stepwise regression method, a base regression model is formed using the OLS estimation method. Thereafter, a variable is added or subtracted to/from the base model depending upon the **Akaike Information Criteria** (**AIC**) value. The standard rule is that a minimum AIC would guarantee a best fit in comparison to other methods. Taking the `Cars93_1.csv` file, let's create a multiple linear regression model by using the stepwise method. There are three different ways to test out the best model using the step formula:

- Forward method
- Backward method
- Both

In the forward selection method, initially a null model is created and variable is added to see any improvement in the AIC value. The independent variables is added to the null model till there is an improvement in the AIC value. In backward selection method, the complete model is considered to be the base model. An independent variable is deducted from the model and the AIC value is checked. If there is an improvement in AIC, the variable would be removed; the process will go on till the AIC value becomes minimum. In both the methods, alternatively forward and back selection method is used to identify the most relevant model. Let's look at the model result at first iteration:

```
> #base model
> fit<-lm(MPG.Overall~.,data=Cars93_1)
> #stepwise regression
> model<-step(fit,method="both")
```

```
Start: AIC=183.52
MPG.Overall ~ Price + EngineSize + Horsepower + RPM + Rev.per.mile +
Fuel.tank.capacity + Length + Wheelbase + Width + Turn.circle +
Rear.seat.room + Luggage.room + Weight
Df Sum of Sq RSS AIC
- Turn.circle 1 0.147 546.54 181.54
- Rear.seat.room 1 0.239 546.64 181.56
- Horsepower 1 0.323 546.72 181.57
- Price 1 6.055 552.45 182.43
- Width 1 6.185 552.58 182.45
- RPM 1 6.686 553.08 182.52
- Length 1 6.695 553.09 182.52
- EngineSize 1 8.186 554.58 182.74
- Luggage.room 1 9.673 556.07 182.96
<none> 546.40 183.52
- Wheelbase 1 35.799 582.20 186.73
- Rev.per.mile 1 40.565 586.96 187.40
- Fuel.tank.capacity 1 47.646 594.04 188.38
- Weight 1 63.400 609.80 190.53
```

In the beginning, AIC is **183.52**. In the final model, AIC is `175.51` and there are six independent variables, as shown next:

```
Step: AIC=175.51
MPG.Overall ~ EngineSize + RPM + Rev.per.mile + Fuel.tank.capacity +
Wheelbase + Width + Luggage.room + Weight
Df Sum of Sq RSS AIC
- Luggage.room 1 8.976 568.78 174.82
- Width 1 12.654 572.46 175.34
<none> 559.81 175.51
- EngineSize 1 14.022 573.83 175.54
- RPM 1 19.422 579.23 176.31
- Wheelbase 1 28.477 588.28 177.58
- Rev.per.mile 1 37.873 597.68 178.88
- Fuel.tank.capacity 1 52.516 612.32 180.86
- Weight 1 135.462 695.27 191.28
```

In the preceding table, “none” entry denotes the stage which is the final model, where further tuning or variable reduction is not possible. Hence, dropping more variables after that point would not make sense.

Logistic regression

The linear regression model based on the ordinary least square method assumes that the relationship between the dependent variable and the independent variables is linear; however, the logistic regression model assumes the relationship to be logarithmic. There are many real-life scenarios where the variable of interest is categorical in nature, such as buying a product or not, approving a credit card or not, tumor is cancerous or not, and so on. Logistic regression not only predicts a dependent variable class but it predicts the probability of a case belonging to a level in the dependent variable. The independent variables need not be normally distributed and need not have equal variance. Logistic regression belongs to the family of generalized linear regression. If the dependent variable has two levels then logistic regression can be applied, but if it has more than two levels, such as high, medium, and low, then multinomial logistic regression model can be applied. All the independent variables can be continuous, categorical, or nominal.

The logistic regression model can be explained using the following equation:

$$\ln\left[\frac{P(Y)}{1-P(Y)}\right] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 ..... \beta_n X_n) + \varepsilon .....(6)$$

Ln (P(Y)/1-P(Y)) is the log odds of the outcome. The beta coefficients mentioned in the previous equation explain how the log odds of the outcome variable increase or decrease for every one unit increase or decrease in the explanatory variable.

Let's take the `Artpiece.csv` dataset where the dependent variable is a good purchase or not need to be predicted based on the independent variables.

For a logistic regression model, the dependent variable needs to be a binary variable with two levels. If it is not then the first task is to convert the dependent variable to binary. For the independent variables it is important to check the type of the variable as well as the levels. If some independent variables are categorical then binary conversion (creating dummy variables for each category) is required. So here you go with data conversion in necessary format for the logistic regression model:

```
> #data conversion
> Artpiece$IsGood.Purchase<-as.factor(Artpiece$IsGood.Purchase)
> Artpiece$Is.It.Online.Sale<-as.factor(Artpiece$Is.It.Online.Sale)
> #removing NA, Missing values from the data
> Artpiece<-na.omit(Artpiece)
> ## 75% of the sample size
> smp_size <- floor(0.75 * nrow(Artpiece))
```

```
> ## set the seed to make your partition reproductible
> set.seed(123)
> train_ind <- sample(seq_len(nrow(Artpiece)), size = smp_size)
> train <- Artpiece[train_ind, ]
> test <- Artpiece[-train_ind, ]
```

The `train` dataset is used for building the logistic regression model and the `test` dataset would be used for testing the model:

```
> #Logistic regression Model
> model1<-glm(IsGood.Purchase ~.,family=binomial(logit),data=train)
> #Model results/ components
> summary(model1)
Call:
glm(formula = IsGood.Purchase ~ ., family = binomial(logit),
data = train)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.1596 -0.4849 -0.4096 -0.3401 3.8760
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.734e-01 1.250e+00 -0.539 0.58995
Critic.Ratings 1.372e-01 1.119e-02 12.267 < 2e-16 ***
Acq.Cost -7.998e-06 1.867e-06 -4.284 1.83e-05 ***
CurrentAuctionAveragePrice -1.460e-05 1.360e-06 -10.731 < 2e-16 ***
Brush.Size1 -1.378e+00 1.246e+00 -1.107 0.26840
Brush.Size2 -1.684e+00 1.246e+00 -1.352 0.17647
Brush.Size3 -1.128e+00 1.252e+00 -0.902 0.36730
Brush.SizeNULL 1.599e+00 1.246e+00 1.283 0.19946
CollectorsAverageprice -2.420e-05 4.677e-06 -5.175 2.28e-07 ***
Is.It.Online.Sale1 -1.710e-01 9.621e-02 -1.777 0.07552 .
Min.Guarantee.Cost 9.830e-06 3.329e-06 2.953 0.00314 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 40601 on 54500 degrees of freedom
Residual deviance: 34872 on 54490 degrees of freedom
AIC: 34894
Number of Fisher Scoring iterations: 5
```

From the previous model result, all the independent variables are statistically significant at 5% level except the `brush` variable, which is a categorical variable. The model automatically creates dummy variables for each of the levels of a categorical variable. It takes the first level from the categorical variable and compares with other level, if the row belongs to that category then 1 else 0. 95% confidence interval for model coefficients and exponentiated model coefficients can be calculated using the following script. The left hand side equation contains natural logarithm to remove that we have to make exponentiation of the right

hand side expression which is the model coefficient:

```
> # 95% CI for exponentiated coefficients
> exp(confint(model1))
Waiting for profiling to be done...
2.5 % 97.5 %
(Intercept) 0.02301865 5.5660594
Critic.Ratings 1.12226719 1.1725835
Acq.Cost 0.99998833 0.9999957
CurrentAuctionAveragePrice 0.99998274 0.9999881
Brush.Size1 0.02326462 5.5558138
Brush.Size2 0.01714345 4.0948557
Brush.Size3 0.02953703 7.1850301
Brush.SizeNULL 0.45631813 109.1655370
CollectorsAverageprice 0.99996666 0.9999850
Is.It.Online.Sale1 0.69545357 1.0142229
Min.Guarantee.Cost 1.00000327 1.0000163
> confint(model1)
2.5 % 97.5 %
(Intercept) -3.771450e+00 1.716687e+00
Critic.Ratings 1.153509e-01 1.592094e-01
Acq.Cost -1.166650e-05 -4.348562e-06
CurrentAuctionAveragePrice -1.726348e-05 -1.193180e-05
Brush.Size1 -3.760822e+00 1.714845e+00
Brush.Size2 -4.066139e+00 1.409731e+00
Brush.Size3 -3.522110e+00 1.972000e+00
Brush.SizeNULL -7.845651e-01 4.692865e+00
CollectorsAverageprice -3.333576e-05 -1.500118e-05
Is.It.Online.Sale1 -3.631910e-01 1.412267e-02
Min.Guarantee.Cost 3.268054e-06 1.631710e-05
```

ANOVA can be performed on the results of the logistic regression model using chi-square test statistics. From the ANOVA which shows analysis of deviance in case of a logistic regression here, we can conclude that the deviance of model change is statistically significant when we include more independent variables in the model. The independent variables are added sequentially to the model and at every iteration the deviance of the error term is compared with the previous iteration to see if the addition or deletion of any variable is useful in reducing the error for the model:

```
> #ANOVA
> anova(model1,test="Chisq")
Analysis of Deviance Table
Model: binomial, link: logit
Response: IsGood.Purchase
Terms added sequentially (first to last)
Df Deviance Resid. Df Resid. Dev Pr(>Chi)
NULL 54500 40601
```

```
Critic.Ratings 1 354.6 54499 40246 < 2.2e-16 ***
Acq.Cost 1 477.8 54498 39768 < 2.2e-16 ***
CurrentAuctionAveragePrice 1 166.4 54497 39602 < 2.2e-16 ***
Brush.Size 4 4690.9 54493 34911 < 2.2e-16 ***
CollectorsAverageprice 1 27.3 54492 34884 1.773e-07 ***
Is.It.Online.Sale 1 3.3 54491 34880 0.067877 .
Min.Guarantee.Cost 1 8.6 54490 34872 0.003417 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> #Plotting the model
> plot(model1$fitted)
```

The preceding plot displays the fitted versus actual values (index) in a regression model. The predicted probability from the model can be extracted using the `link` option and the class prediction can be extracted using the `response` option, using the `predict` function:

```
> #Predicted Probability
> test$goodP<-predict(model1,newdata=test,type="response")
> test$goodL<-predict(model1,newdata=test,type="link")
```

Automatic logistic regression model selection can be done in three different ways using the stepwise regression method via AIC criteria:

- **Backward selection method**: All the independent variables are part of the initial model. Using Wald chi-square test of significance, variables with lowest value of chi-square or highest p-value such that it exceeds 0.05 (given that 95% confidence interval is used in the model) are removed from the model. The backward selection process of variable removal stops when it is not possible to find a variable that fits the Wald test criteria.
- **Forward selection method**: Model starts with a null model. A variable is entered into the model based on the lowest p-value or highest chi-square test statistic value. The process stops when there are no such cases found.
- **Both**: The third approach is to apply both backward selection and forward selection:

```
    > #auto detection of model
    > fit_step<-stepAIC(model1,method="both")
    Start: AIC=34893.93
    IsGood.Purchase ~ Critic.Ratings + Acq.Cost +
CurrentAuctionAveragePrice +
    Brush.Size + CollectorsAverageprice + Is.It.Online.Sale +
    Min.Guarantee.Cost
    Df Deviance AIC
    <none> 34872 34894
    - Is.It.Online.Sale 1 34875 34895
    - Min.Guarantee.Cost 1 34880 34900
    - Acq.Cost 1 34890 34910
    - CollectorsAverageprice 1 34898 34918
    - CurrentAuctionAveragePrice 1 34988 35008
    - Critic.Ratings 1 35025 35045
    - Brush.Size 4 39554 39568
```

The method selected for variable selection is both forward and backward. For example, in the first step forward selection was used, then at the same time backward section was also applied to check if the said variable could be removed from the model or not.

While applying the stepwise model selection procedure, it is very important to keep in mind the limitations of the stepwise method. Sometimes model overfitting happens, sometimes when you have a large number of predictors and mostly the predictors are highly correlated, then stepwise regression provides higher variability and low accuracy. So to address this issue, we can apply penalized regression method, which we are going to discuss in the next section:

```
> summary(fit_step)
Call:
glm(formula = IsGood.Purchase ~ Critic.Ratings + Acq.Cost +
CurrentAuctionAveragePrice +
Brush.Size + CollectorsAverageprice + Is.It.Online.Sale +
```

```
Min.Guarantee.Cost, family = binomial(logit), data = train)
Deviance Residuals:
Min 1Q Median 3Q Max
-2.1596 -0.4849 -0.4096 -0.3401 3.8760
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.734e-01 1.250e+00 -0.539 0.58995
Critic.Ratings 1.372e-01 1.119e-02 12.267 < 2e-16 ***
Acq.Cost -7.998e-06 1.867e-06 -4.284 1.83e-05 ***
CurrentAuctionAveragePrice -1.460e-05 1.360e-06 -10.731 < 2e-16 ***
Brush.Size1 -1.378e+00 1.246e+00 -1.107 0.26840
Brush.Size2 -1.684e+00 1.246e+00 -1.352 0.17647
Brush.Size3 -1.128e+00 1.252e+00 -0.902 0.36730
Brush.SizeNULL 1.599e+00 1.246e+00 1.283 0.19946
CollectorsAverageprice -2.420e-05 4.677e-06 -5.175 2.28e-07 ***
Is.It.Online.Sale1 -1.710e-01 9.621e-02 -1.777 0.07552 .
Min.Guarantee.Cost 9.830e-06 3.329e-06 2.953 0.00314 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 40601 on 54500 degrees of freedom
Residual deviance: 34872 on 54490 degrees of freedom
AIC: 34894
Number of Fisher Scoring iterations: 5
```

To check the multicollinearity among the independent variables, the VIF measure is used. The VIF computed for the step AIC-based final model is as follows:

```
> library(MASS);library(plyr);library(car)
>
> vif(fit_step)
GVIF Df GVIF^(1/(2*Df))
Critic.Ratings 1.204411 1 1.097457
Acq.Cost 2.582229 1 1.606932
CurrentAuctionAveragePrice 2.527691 1 1.589871
Brush.Size 1.094967 4 1.011405
CollectorsAverageprice 1.084788 1 1.041532
Is.It.Online.Sale 1.007080 1 1.003534
Min.Guarantee.Cost 1.163249 1 1.078540
```

The probability score for the `training` dataset is computed using the response option and the probability score can be used to display the results of classification on an AUC curve, also known as **Receiver Operating Characteristic** (**ROC**) curve:

```
> train$prob=predict(fit_step,type=c("response"))
> library(pROC)
> g <- roc(IsGood.Purchase ~ prob, data = train)
> g
```

```
Call:
roc.formula(formula = IsGood.Purchase ~ prob, data = train)
Data: prob in 47808 controls (IsGood.Purchase 0) < 6693 cases
(IsGood.Purchase 1).
Area under the curve: 0.7201
> plot(g)
```

The curve derived from the model shows the area under the curve. The horizontal axis shows the false positive rate and the vertical axis shows the true positive rate. If the area under the curve is more than 70%, then the model is considered to be a good model as per industry standard:

```
> #Label the prediction result above the certain threshold as Yes and No
> #Change the threshold value and check which give the better result.
> train$prob <- ifelse(prob > 0.5, "Yes", "No")
>
> #print the confusion matrix between the predicted and actual response on
testdata.
> t<-table(train$prob,train$IsGood.Purchase)
>
> #accuracy
> prop.table(t)
0 1
No 0.86407589 0.09229188
Yes 0.01311903 0.03051320
```

The threshold for classification table is 0.50 (50%) as probability. If someone wants to change the percentage of correct classified objects, then he/she can do so by putting a filter on the probability score. The following code is used to create a confusion matrix using a different probability value. From the previous classification table, it can be concluded that 90% people correctly classified.

Cubic regression

Cubic regression is another form of regression where the parameters in a linear regression model are increased up to one or two levels of polynomial calculation. Using the `Cars93_1.csv` dataset, let's understand the cubic regression:

```
> fit.6<-lm(MPG.Overall~ I(Price)^3+I(Horsepower)^3+I(RPM)^3+
+ Wheelbase+Width+Turn.circle, data=Cars93_1[-c(28,42,39,59,60,77),])
> summary(fit.6)
Call:
lm(formula = MPG.Overall ~ I(Price)^3 + I(Horsepower)^3 + I(RPM)^3 +
Wheelbase + Width + Turn.circle, data = Cars93_1[-c(28, 42,
39, 59, 60, 77), ])
Residuals:
Min 1Q Median 3Q Max
-5.279 -1.901 -0.006 1.590 8.433
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 57.078025 12.349300 4.622 1.44e-05 ***
I(Price) -0.108436 0.065659 -1.652 0.1026
I(Horsepower) -0.024621 0.015102 -1.630 0.1070
I(RPM) 0.001122 0.000727 1.543 0.1268
Wheelbase -0.201836 0.079948 -2.525 0.0136 *
Width -0.104108 0.198396 -0.525 0.6012
Turn.circle -0.095739 0.158298 -0.605 0.5470
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 2.609 on 80 degrees of freedom
Multiple R-squared: 0.6974, Adjusted R-squared: 0.6747
F-statistic: 30.73 on 6 and 80 DF, p-value: < 2.2e-16
> vif(fit.6)
I(Price) I(Horsepower) I(RPM) Wheelbase Width Turn.circle
4.121923 7.048971 2.418494 3.701812 7.054405 3.284228
> coefficients(fit.6)
(Intercept) I(Price) I(Horsepower) I(RPM) Wheelbase Width
57.07802478 -0.10843594 -0.02462074 0.00112168 -0.20183606 -0.10410833
Turn.circle
-0.09573848
```

Keeping all other parameters constant, one unit change in the independent variable would bring a multiplicative change in the dependent variable and the multiplication factor is the cube of the beta coefficients.

Penalized regression

The problem of maximum likelihood estimation comes into picture when we include large number of predictors or highly correlated predictors or both in a regression model and its failure to provide higher accuracy in regression problems gives rise to the introduction of penalized regression in data mining. The properties of maximum likelihood cannot be satisfying the regression procedure because of high variability and improper interpretation. To address the issue, most relevant subset selection came into picture. However, the subset selection procedure has some demerits. To again solve the problem, a new method can be introduced, which is popularly known as penalized maximum likelihood estimation method. There are two different variants of penalized regression that we are going to discuss here:

- **Ridge regression**: The ridge regression is known as *L2* quadratic penalty and the equation can be represented as follows:

$$\boxed{f(\beta) = \sum_{j=1}^{n} \beta_j^2}$$

  In comparison to the maximum likelihood estimates, the *L1* and *L2* method of estimation shrinks the beta coefficients towards zero. In order to avoid the problem of multicollinearity and higher number of predictors or both typically in a scenario when you have less number of observations, the shrinkage methods reduces the overfitted beta coefficient values.

  Taking the `Cars93_1.csv` dataset, we can test out the model and the results from it. The lambda parameter basically strikes the balance between penalty and fit of log likelihood function. The selection of lambda value is very critical for the model; if it is a small value then the model may overfit the data and high variance would become evident, and if the lambda chosen is very large value then it may lead to biased result.

```
> #installing the library
> library(glmnet)
```

```
> #removing the missing values from the dataset
> Cars93_2<-na.omit(Cars93_1)
> #independent variables matrix
> x<-as.matrix(Cars93_2[,-1])
> #dependent variale matrix
> y<-as.matrix(Cars93_2[,1])
> #fitting the regression model
> mod<-glmnet(x,y,family = "gaussian",alpha = 0,lambda = 0.001)
> #summary of the model
> summary(mod)
Length Class Mode
a0 1 -none- numeric
beta 13 dgCMatrix S4
df 1 -none- numeric
dim 2 -none- numeric
lambda 1 -none- numeric
dev.ratio 1 -none- numeric
nulldev 1 -none- numeric
npasses 1 -none- numeric
jerr 1 -none- numeric
offset 1 -none- logical
call 6 -none- call
nobs 1 -none- numeric
#Making predictions
pred<-predict(mod,x,type = "link")
#estimating the error for the model.
mean((y-pred)^2)
> #Making predictions
> pred<-predict(mod,x,type = "link")
> #estimating the error for the model.
> mean((y-pred)^2)
[1] 6.663406
```

Given the preceding background, the regression model was able to show up with 6.66% error, which implies the model is giving 93.34% accuracy. This is not the final iteration. The model needs to be tested couple of times and various subsamples as out of the bag need to be used to conclude the final accuracy of the model.

- **Least Absolute Shrinkage Operator (LASSO)**: The LASSO regression is known as L1 absolute value penalty (Tibshirani, 1997). The equation for LASSO regression can be expressed as follows:

$$f(\beta) = \sum_{j=1}^{n} |\beta|$$

Taking the `Cars93_1.csv` dataset, we can test out the model and the results from it. The lambda parameter basically strikes the balance between penalty and fit of log likelihood function. The selection of the lambda value is very critical for the model; if it is a small value then the model may over fit the data and high variance would become evident, and if the lambda chosen is very large value then it may lead to biased result. Let's try to implement the model and verify the results:

```
> #installing the library
> library(lars)
> #removing the missing values from the dataset
> Cars93_2<-na.omit(Cars93_1)
> #independent variables matrix
> x<-as.matrix(Cars93_2[,-1])
> #dependent variale matrix
> y<-as.matrix(Cars93_2[,1])
> #fitting the LASSO regression model
> model<-lars(x,y,type = "lasso")
> #summary of the model
> summary(model)
LARS/LASSO
Call: lars(x = x, y = y, type = "lasso")
Df Rss Cp
0 1 2215.17 195.6822
1 2 1138.71 63.7148
2 3 786.48 21.8784
3 4 724.26 16.1356
4 5 699.39 15.0400
5 6 692.49 16.1821
6 7 675.16 16.0246
7 8 634.59 12.9762
8 9 623.74 13.6260
9 8 617.24 10.8164
10 9 592.76 9.7695
11 10 587.43 11.1064
12 11 551.46 8.6302
13 12 548.22 10.2275
14 13 547.85 12.1805
15 14 546.40 14.0000
```

The type option within the **Least Angle Regression** (LARS) provides opportunities to apply various variants of the lars model with `"lasso"`, `"lar"`, `"forward.stagewise"`, or `"stepwise"`. Using these we can create various models and compare the results.

From the preceding output of the model we can see the RSS and CP along with degrees of freedom, at each iterations so we need to find out a best model step where the RSS for the model is minimum:

```
> #select best step with a minin error
> best_model<-model$df[which.min(model$RSS)]
> best_model
14
> #Making predictions
> pred<-predict(model,x,s=best_model,type = "fit")$fit
> #estimating the error for the model.
> mean((y-pred)^2)
[1] 6.685669
```

The best model is available at step 14, where the RSS is minimum and hence using that best step we can make predictions. Using the predict function at that step, we can generate predictions and the error for the model chosen is 6.68%. The visualization for the fitted model can be displayed as shown next. The vertical axis shows the standardized coefficients and the horizontal axis shows the model coefficient progression for the dependent variable. The *x* axis shows the ratio of absolute value of beta coefficients upon the max of the absolute value of the beta coefficients. The numerator is the estimated beta coefficient current and the denominator is the beta coefficient for the OLS model. When the shrinkage parameter in LASSO takes a value of zero, the model will correspond to a OLS regression method. As the penalized parameter increases the sum of absolute values of the beta coefficients pulled to zero:

In the preceding graph, the vertical bars indicate when a variable has been pulled to zero. Vertical bar corresponding to 15 shows there are 15 predictor variables need to be reduced to zero, which corresponds to higher penalized parameter lambda value 1.0. When lambda the penalized parameter is very small, the LASSO would approach the OLS regression and as lambda increases you will see fewer variables in your model. Hence, after lambda value of 1.0, there are no variables left in the model.

L1 and *L2* are known as regularized regression methods. *L1* cannot zero out regression coefficients; either you will get all the coefficients or none of them. However, *L2* does parameter shrinkage and variable selection automatically.

Summary

In this chapter, we discussed how to create linear regression, logistic regression, and other nonlinear regression based methods for predicting the outcome of a variable in a business scenario. Regression methods are basically very important in data mining projects, in order to create predictive models to know the future values of the unknown independent variables. We looked at various scenarios where we understood which type of model can be applied where. In the next chapter, we are going to talk about association rules or market basket analysis to understand important patterns hidden in a transactional database.

5

Market Basket Analysis with Groceries Data

Think about a scenario from a retailer or e-commerce store manager when it comes to recommending the right products to customers. Product recommendation is one important domain in data mining practice. Product recommendation can happen in three different ways: by associating customer's behavior with their purchase history, by associating items that are being purchased on each visit, and lastly by looking at the gross sales in each category and then using the retailer's past experience. In this chapter, we will be looking at the second method of product recommendation, popularly known as **Market Basket Analysis** (**MBA**) also known as association rules, which is by associating items purchased at transaction level and finding out the sub-segments of users having similar products and hence, recommending the products.

In this chapter, we will cover the following topics:

- What is MBA
- Where to apply MBA
- Assumptions/prerequisites
- Modeling techniques
- Limitations
- Practical project

Introduction to Market Basket Analysis

Is MBA limited to retail and e-commerce domain only? Now, let's think about the problems where MBA or association rules can be applied to get useful insights:

- In the healthcare sector, particularly medical diagnosis, as an example, having high blood pressure and being diabetic is a common combination. So it can be concluded that people having high blood pressure are more likely to be diabetic and vice versa. Hence, by analyzing the prevalence of medical conditions or diseases, it can be said what other diseases they may likely get in future.
- In retail and e-commerce, promotions and campaigns can be designed once a merchandiser knows the relationship between the purchase patterns of different items. MBA provides insights into the relationship between various items so that product recommendation can be designed.
- In banking and financial services also, MBA can be used. Taking into account the number of products a bank is offering to the customers, such as insurance policies, mutual funds, loans, and credit cards, is there any association between buying insurance policies and mutual funds? If yes, that can be explored using market basket analysis. Before recommending a product, the officer/agent should verify what all products go together so that the upsell and cross-sell of financial products can be designed.
- In the telecommunication domain, analysis of customer usage and selection of product options provide insights into effective designing of promotions and cross-selling of products.
- In unstructured data analysis, particularly in text mining, what words go together in which domain and how the terminologies are related and used by different people can be extracted using association rules.

What is MBA?

Market Basket Analysis is the study of relationships between various products and products that are purchased together or in a series of transactions. In standard data mining literature, the task of market basket analysis is to discover actionable insights in transactional databases. In order to understand MBA or **association rules** (**arules**), it is important to understand three concepts and their importance in deciding rules:

- **Support**: A transaction can contain a single item or a set of items. Consider the item `set x = {bread, butter, jam, curd}`; the support of `x` implies the proportion of transactions when all four items are purchased together to the total number of transactions in the database:

Support of X = number of transactions involving X / total number of transactions

- **Confidence**: Confidence always refers to the confidence of a rule. It can be defined as confidence of *x => y*. Hence, *support (x U y)* implies the number of transactions in the database containing both item set `x` and item `y`:

Confidence of X => Y = support of (X U Y) / support of (X)

- **Lift**: Lift can be defined as the proportion of observed support to expected support:

*Lift of a rule X => Y = support of (X U Y) / support of (X) * support of (Y)*

An association rule stands supported in a transactional database if it satisfies the minimum support and minimum confidence criteria.

Let's take an example to understand the concepts.

The following is an example from the `Groceries.csv` dataset from the `arules` library in R:

Transaction ID	Items
001	Bread, banana, milk, butter
002	Milk, butter, banana
003	Curd, milk, banana
004	Curd, bread, butter
005	Milk, curd, butter, banana
006	Milk, curd, bread

The support of item set *X* where *X = {milk, banana} = proportion of transactions where milk and banana bought together; 4/6 = 0.67*, hence, the support of the *X* item set is *0.67*.

Let's assume *Y* is curd; the confidence of *X => Y*, which is *{milk, banana} => {curd}*, is the proportion of support *(X U Y) / support(X);* three products–milk, banana, and curd–are purchased together 2 times. Hence the confidence of the *rule = (2/6)/0.67 = 0.5*. This implies that 50% of the transactions containing milk and banana the rule are correct.

Confidence of a rule can be interpreted as a conditional probability of purchasing *Y* such that the item set *X* has already been purchased.

When we create too many rules in a transactional database, we need a measure to rank the rules; lift is a measure to rank the rules. From the preceding example, we can calculate the lift for the rule *X* to *Y*. Lift of *X => Y = support of (X U Y) / support of (X) * support of (Y) = 0.33/0.66*0.66 = 0.33/0.4356 = 0.7575*.

Where to apply MBA?

To understand the relationship between various variables in large databases, it is required to apply market basket analysis or association rules. It is the simplest possible way to understand the association. Though we have explained various industries where the concept of association rule can be applied, the practical implementation depends on the kind of dataset available and the length of data available. For example, if you want to study the relationship between various sensors integrated at a power generation plant to understand which all sensors are activated at a specific temperature level (very high), you may not find enough data points. This is because very high temperature is a rare event and to understand the relationship between various sensors, it is important to capture large data at that temperature level.

Data requirement

Product recommendation rules are generated from the results of the association rule model. For example, what is the customer going to buy next if he/she has already added a Coke, a chips packet, and a candle? To generate rules for product recommendation, we need frequent item sets and hence the retailer can cross-sell products. Therefore the input data format should be transactional, which in real-life projects sometimes happens and sometimes does not. If the data is not in transaction form, then a transformation needs to be done. In R, dataframe is a representation of mixed data. Can we transform the dataframe to transactional data so that we can apply association rules on that transformed dataset? This is because the algorithm requires that the input data format be transactional. Let's have a look at the methods to read transactions from a dataframe.

The `arules` library in R provides a function to read transactions from a dataframe. This function provides two options, single and basket. In single format, each line represents a single item; however, in basket format, each line represents a transaction comprising item levels and separated by a comma, space, or tab. For example:

```
> ## create a basket format
> data <- paste(
+ "Bread, Butter, Jam",
+ "Bread, Butter",
+ "Bread",
```

```
+ "Butter, Banana",
+ sep="\n")
> cat(data)
Bread, Butter, Jam
Bread, Butter
Bread
Butter, Banana
> write(data, file = "basket_format")
>
> ## read data
> library(arules)
> tr <- read.transactions("basket_format", format = "basket", sep=",")
> inspect(tr)
items
1 {Bread,Butter,Jam}
2 {Bread,Butter}
3 {Bread}
4 {Banana,Butter}
```

Now let's look at single format transactional data creation from a dataframe:

```
> ## create single format
> data <- paste(
+ "trans1 Bread",
+ "trans2 Bread",
+ "trans2 Butter",
+ "trans3 Jam",
+ sep ="\n")
> cat(data)
trans1 Bread
trans2 Bread
trans2 Butter
trans3 Jam
> write(data, file = "single_format")
>
> ## read data
> tr <- read.transactions("single_format", format = "single", cols =
c(1,2))
> inspect(tr)
items transactionID
1 {Bread} trans1
2 {Bread,Butter} trans2
3 {Jam} trans3
```

The preceding code explains how a typical piece of transactional data received in a tabular format, just like the spreadsheet format equivalent of R dataframe, can be converted to a transactional data format as required by the `arules` input data format.

Assumptions/prerequisites

Implementation of association rules for performing market basket analysis is based on some assumptions. Here are the assumptions:

- Assume that all data is categorical.
- There should not be any sparse data; the sparsity percentage should be minimum. Sparsity implies a lot of cells in a dataset with no values (with blank cells).
- The number of transactions required to find an interesting relationship between various products is a function of the number of items or products covered in the database. In other words, if you include more products with a less number of transactions, you will get higher sparsity in the dataset and vice versa.
- The higher the number of items you want to include in the analysis, the more number of transactions you will need to validate the rules.

Modeling techniques

There are two algorithms that are very popular for finding frequent item sets that exist in a retail transactional database:

- **Apriori algorithm**: It was developed by Agrawal and Srikant (1994). It considers the breadth first and provides counts of transactions.
- **Eclat algorithm**: This was developed by Zaki et. al. (1997b). It considers depth first; it provides intersection of transactions rather than count of transactions.

In this chapter, we are using the `Groceries.csv` dataset, which will be used in both the algorithms.

Limitations

Though association rules are a top choice among practitioners when it comes to understanding relationships in a large transactional database, they have certain inherent limitations:

- Association rule mining is a probabilistic approach in the sense that it computes the conditional probability of a product being purchased, such that a set of other items has already been purchased/added to the cart by the customer. It only estimates the likelihood of buying the product. The exact accuracy of the rule

may or may not be true.

- Association rule is a conditional probability estimation method using simple count of items as a measure.
- Association rules are not useful practically, even with a high support, confidence, and lift; this is because most of the times, trivial rules may sound genuine. There is no mechanism to filter out the trivial rules from the global set of rules but using a machine learning framework (which is out of the scope of this book).
- The algorithm provides a large set of rules; however, a few of them are important, and there is no automatic method to identify the number of useful rules.

Practical project

The `Groceries.csv` dataset that we are going to use comprises of 1 month of real-world **point-of-sale** (**POS**) transaction data from a grocery store. The dataset encompasses 9,835 transactions and there are 169 categories. Item sets are defined as a combination of items or products P_i $\{i = 1 \ldots\ldots n\}$ that customers buy on the same visit. To put it in a simpler way, the item sets are basically the grocery bills that we usually get while shopping from a retail store. The bill number is considered the transaction number and the items mentioned in that bill are considered the market basket. A snapshot of the dataset is given as follows:

brown bread	soda	fruit/vegetable ju	canned beer	newspapers	shopping bags
yogurt	beverages	bottled water	specialty bar		
hamburger meat	other vegetables	rolls/buns	spices	bottled water	hygiene articles
root vegetables	other vegetables	whole milk	beverages	sugar	
pork	berries	other vegetables	whole milk	whipped/sour cre	artif. sweetener
beef	grapes	detergent			

The columns represent items and the rows represent transactions in the sample snapshot just displayed. Let's explore the dataset to understand the features:

```
> # Load the libraries
> library(arules)
> library(arulesViz)
> # Load the data set
> data(Groceries) #directly reading from library
> Groceries<-read.transactions("groceries.csv",sep=",") #reading from local
computer
```

The transactional dataframe contains three columns. The first column represents the name of the product/item, the second column represents the `level2` categorization of the products with 55 levels, and the third variable is the `level3` segment of the product with 10 categories. This screenshot makes it clearer:

```
Groceries          Large transactions (9835 elements, 567.9 Kb)
  ..@ transactionInfo:'data.frame': 9835 obs. of 0 variables
  ..@ data :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
  .. .. ..@ i : int [1:43367] 13 60 69 78 14 29 98 24 15 29 ...
  .. .. ..@ p : int [1:9836] 0 4 7 8 12 16 21 22 27 28 ...
  .. .. ..@ Dim : int [1:2] 169 9835
  .. .. ..@ Dimnames:List of 2
  .. .. .. ..$ : NULL
  .. .. .. ..$ : NULL
  .. .. ..@ factors : list()
  ..@ itemInfo :'data.frame': 169 obs. of 3 variables:
  .. ..$ labels: chr [1:169] "frankfurter" "sausage" "liver loaf" "ha…
  .. ..$ level2: Factor w/ 55 levels "baby food","bags",..: 44 44 44 …
  .. ..$ level1: Factor w/ 10 levels "canned food",..: 6 6 6 6 6 6 6 …
  ..@ itemsetInfo :'data.frame': 9835 obs. of 1 variable:
  .. ..$ itemsetID: chr [1:9835] "1" "2" "3" "4" ...
```

The summary of a transactional data provides an idea about the total transactions existing in the database, number of items that are part of the database, sparsity of the data, and also the frequency of the top few items prominent in the transactional database:

```
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 yogurt
2513 1903 1809 1715 1372
(Other)
34055
```

There are 9,835 transactions in the dataset; out of 169 items, the most frequent items with their corresponding frequencies are previously represented. In a matrix of 9,835 cross 169, only `0.0261` or 2.61% of the cells are filled with values; the rest are empty. This 2.61% is the sparsity of the dataset.

Element (itemset/transaction) length distribution:

```
sizes
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55 46 29
18 19 20 21 22 23 24 26 27 28 29 32
```

```
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
1 abrasive cleaner
2 artif. sweetener
3 baby cosmetics
```

From the preceding item set length distribution, it can be interpreted that there is one transaction with 32 products bought together. There are 3 transactions with 29 items purchased together. Likewise, we can interpret the frequency of other item sets in the dataset. This can be better understood using the item frequency plot. To know the first three transactions from the dataset, we can use the following command:

```
> inspect(Groceries[1:3])
items
1 {citrus fruit,semi-finished bread,margarine,ready soups}
2 {tropical fruit,yogurt,coffee}
3 {whole milk}
```

To know the proportion of transactions containing each item in the groceries database, we can use the following command. The first item, frankfurter, is available in 5.89% of the transactions; sausage is available in 9.39% of the transactions; and so on and so forth:

```
> cbind(itemFrequency(Groceries[,1:10])*100)
[,1]
frankfurter 5.8973055
sausage 9.3950178
liver loaf 0.5083884
ham 2.6029487
meat 2.5826131
finished products 0.6507372
organic sausage 0.2236909
chicken 4.2907982
turkey 0.8134215
pork 5.7651246
```

Two important parameters dictate the frequent item sets in a transactional dataset, support and confidence. The higher the support the lesser would be the number of rules in a dataset, and you would probably miss interesting relationships between various variables and vice versa. Sometimes with a higher support, confidence, and lift values also, it is not guaranteed to get useful rules. Hence, in practice, a user can experiment with different levels of support and confidence to arrive at meaningful rules. Even at sufficiently good amounts of minimum support and minimum confidence levels, the rules seem to be quite trivial.

To address the issue of irrelevant rules in frequent item set mining, closed item set mining is considered as a method. An item set *X* is said to be closed if no superset of it has the same support as *X* and *X* is maximal at % support if no superset of *X* has at least % support:

```
> itemFrequencyPlot(Groceries, support=0.01, main="Relative ItemFreq Plot",
+ type="absolute")
```

In the preceding graph, the item frequency in the absolute count is shown with a minimum support of `0.01`. The following graph indicates the top 50 items with relative percentage count, showing what percentage of transactions in the database contain those items:

```
> itemFrequencyPlot(Groceries,topN=50,type="relative",main="Relative Freq
Plot")
```

Apriori algorithm

Apriori algorithm uses a downward closure property, which states that any subsets of a frequent item set and also frequent item sets. The apriori algorithm uses level-wise search for frequent item sets. This algorithm only creates rules with one item in the right-hand side of the equation (RHS), which is known as consequent; left-hand side of the equation (LHS) known as antecedent. This implies that rules with one item in RHS and blank LHS may appear as valid rules; to avoid these blank rules, the minimum length parameter needs to be changed from 1 to 2:

```
> # Get the association rules based on apriori algo
> rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.10))
Parameter specification:
confidence minval smax arem aval originalSupport support minlen maxlen
target ext
0.1 0.1 1 none FALSE TRUE 0.01 1 10 rules FALSE
Algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
```

```
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [435 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

There are 435 rules with a support value of 1% (proportion of items to be included in rule creation with a minimum of 1%) and confidence level of 10% using apriori algorithm. There are 88 items representing those 435 rules. Using the `summary` command, we can get to know the length of rules and the distribution of rules:

```
> summary(rules)
set of 435 rules
rule length distribution (lhs + rhs):sizes
1 2 3
8 331 96
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 2.000 2.000 2.202 2.000 3.000
summary of quality measures:
support confidence lift
Min. :0.01007 Min. :0.1007 Min. :0.7899
1st Qu.:0.01149 1st Qu.:0.1440 1st Qu.:1.3486
Median :0.01454 Median :0.2138 Median :1.6077
Mean :0.02051 Mean :0.2455 Mean :1.6868
3rd Qu.:0.02115 3rd Qu.:0.3251 3rd Qu.:1.9415
Max. :0.25552 Max. :0.5862 Max. :3.3723
mining info:
data ntransactions support confidence
Groceries 9835 0.01 0.1
```

By looking at the summary of the rules, there are 8 rules with 1 item, including LHS and RHS, which are not valid rules. Those are given as follows; to avoid blank rules, the minimum length needs to be two. After using a minimum length of two, the total number of rules decreased to 427:

```
> inspect(rules[1:8])
lhs rhs support confidence lift
1 {} => {bottled water} 0.1105236 0.1105236 1
2 {} => {tropical fruit} 0.1049314 0.1049314 1
3 {} => {root vegetables} 0.1089985 0.1089985 1
4 {} => {soda} 0.1743772 0.1743772 1
5 {} => {yogurt} 0.1395018 0.1395018 1
6 {} => {rolls/buns} 0.1839349 0.1839349 1
```

```
7 {} => {other vegetables} 0.1934926 0.1934926 1
8 {} => {whole milk} 0.2555160 0.2555160 1
> rules <- apriori(Groceries, parameter = list(supp = 0.01, conf = 0.10,
minlen=2))
Parameter specification:
confidence minval smax arem aval originalSupport support minlen maxlen
target ext
0.1 0.1 1 none FALSE TRUE 0.01 2 10 rules FALSE
Algorithmic control:
filter tree heap memopt load sort verbose
0.1 TRUE TRUE FALSE TRUE 2 TRUE
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [427 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
> summary(rules)
set of 427 rules
rule length distribution (lhs + rhs):sizes
2 3
331 96
```

How do we identify what is the right set of rules? After squeezing the confidence level, we can reduce the number of valid rules. Keeping the confidence level at 10% and changing the support level, we can see how the number of rules is changing. If we have too many rules, it is difficult to implement them; if we have small number of rules, it will not correctly represent the hidden relationship between the items. Hence, having right set of valid rules is a trade-off between support and confidence. The following scree plot shows the number of rules against varying levels of support, keeping the confidence level constant at 10%:

```
> support<-seq(0.01,0.1,0.01)
> support
[1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10
> rules_count<-c(435,128,46,26,14, 10, 10,8,8,8)
> rules_count
[1] 435 128 46 26 14 10 10 8 8 8
> plot(support,rules_count,type = "l",main="Number of rules at different
support %",
+ col="darkred",lwd=3)
```

Looking at the support 0.04 and confidence level 10%, the right number of valid rules for the dataset would be 26, based on the scree-plot result. The reverse can happen too to identify relevant rules; keeping the support level constant and varying the confidence level, we can create another scree plot:

```
> conf<-seq(0.10,1.0,0.10)
> conf
[1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> rules_count<-c(427,231,125,62,15,0,0,0,0,0)
> rules_count
[1] 427 231 125 62 15 0 0 0 0 0
> plot(conf,rules_count,type = "l",main="Number of rules at different
confidence %",
+ col="darkred",lwd=3)
```

Looking at the confidence level of `0.5` and confidence level changing by 10%, the right number of valid rules for the groceries dataset would be 15, based on the scree plot result.

Eclat algorithm

Eclat algorithm uses simple intersection operations for homogenous class clustering with a bottom-up approach. The same code can be re-run using the `eclat` function in R and the result can be retrieved. The `eclat` function accepts two arguments, support and maximum length:

```
> rules_ec <- eclat(Groceries, parameter = list(supp = 0.05))
parameter specification:
tidLists support minlen maxlen target ext
FALSE 0.05 1 10 frequent itemsets FALSE
algorithmic control:
sparse sort verbose
7 -2 TRUE
eclat - find frequent item sets with the eclat algorithm
version 2.6 (2004.08.16) (c) 2002-2004 Christian Borgelt
create itemset ...
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [28 item(s)] done [0.00s].
creating sparse bit matrix ... [28 row(s), 9835 column(s)] done [0.00s].
```

```
writing ... [31 set(s)] done [0.00s].
Creating S4 object ... done [0.00s].
> summary(rules_ec)
set of 31 itemsets
most frequent items:
whole milk other vegetables yogurt rolls/buns frankfurter
4 2 2 2 1
(Other)
23
element (itemset/transaction) length distribution:sizes
1 2
28 3
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 1.000 1.000 1.097 1.000 2.000
summary of quality measures:
support
Min. :0.05236
1st Qu.:0.05831
Median :0.07565
Mean :0.09212
3rd Qu.:0.10173
Max. :0.25552
includes transaction ID lists: FALSE
mining info:
data ntransactions support
Groceries 9835 0.05
```

Using the eclat algorithm, with a support of 5%, there are 31 rules that explain the relationship between different items. The rule includes items that have a representation in at least 5% of the transactions.

While generating a product recommendation, it is important to recommend those rules that have high confidence level, irrespective of support proportion. Based on confidence, the top 5 rules can be derived as follows:

```
> #sorting out the most relevant rules
> rules<-sort(rules, by="confidence", decreasing=TRUE)
> inspect(rules[1:5])
lhs rhs support confidence lift
36 {other vegetables,yogurt} => {whole milk} 0.02226741 0.5128806 2.007235
10 {butter} => {whole milk} 0.02755465 0.4972477 1.946053
3 {curd} => {whole milk} 0.02613116 0.4904580 1.919481
33 {root vegetables,other vegetables} => {whole milk} 0.02318251 0.4892704
1.914833
34 {root vegetables,whole milk} => {other vegetables} 0.02318251 0.4740125
2.449770
```

Rules also can be sorted based on lift and support proportion, by changing the argument in the sort function. The top 5 rules based on lift calculation are as follows:

```
> rules<-sort(rules, by="lift", decreasing=TRUE)
> inspect(rules[1:5])
lhs rhs support confidence lift
35 {other vegetables,whole milk} => {root vegetables} 0.02318251 0.3097826
2.842082
34 {root vegetables,whole milk} => {other vegetables} 0.02318251 0.4740125
2.449770
27 {root vegetables} => {other vegetables} 0.04738180 0.4347015 2.246605
15 {whipped/sour cream} => {other vegetables} 0.02887646 0.4028369 2.081924
37 {whole milk,yogurt} => {other vegetables} 0.02226741 0.3974592 2.054131
```

Visualizing association rules

How the items are related and how the rules can be visually represented is as much important as creating the rules:

```
> #visualizign the rules
> plot(rules,method='graph',interactive = T,shading = T)
```

The preceding graph is created using apriori algorithm. In maximum number of rules, at least you would find either whole milk or other vegetables as those two items are well connected by nodes with other items.

Using eclat algorithm for the same dataset, we have created another set of rules; the following graph shows the visualization of the rules:

Implementation of arules

Once a good market basket analysis or arules model is built, the next task is to integrate the model. arules provides a PMML interface, which is a predictive modeling markup language interface, to integrate with other applications. Other applications can be other statistical software such as, SAS, SPSS, and so on; or it can be Java, PHP, and Android-based applications. The PMML interface makes it easier to integrate the model. When it comes to rule implementation, two important questions a retailer would like to get answer are:

- What are customers likely to buy before buying a product?
- What are the customers likely to buy if they have already purchased some product?

Let's take a product `yogurt`, and the retailer would like to recommend this to customers. Which are the rules that can help the retailer? So the top 5 rules are as follows:

```
> rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.8),
+ appearance = list(default="lhs",rhs="yogurt"),
+ control = list(verbose=F))
> rules<-sort(rules, decreasing=TRUE,by="confidence")
> inspect(rules[1:5])
lhs rhs support confidence
4 {root vegetables,butter,cream cheese } => {yogurt} 0.001016777 0.9090909
10 {tropical fruit,whole milk,butter,sliced cheese} => {yogurt} 0.001016777
0.9090909
11 {other vegetables,curd,whipped/sour cream,cream cheese } => {yogurt}
0.001016777 0.9090909
13 {tropical fruit,other vegetables,butter,white bread} => {yogurt}
0.001016777 0.9090909
2 {sausage,pip fruit,sliced cheese} => {yogurt} 0.001220132 0.8571429
lift
4 6.516698
10 6.516698
11 6.516698
13 6.516698
2 6.144315
> rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf =
0.10,minlen=2),
+ appearance = list(default="rhs",lhs="yogurt"),
+ control = list(verbose=F))
> rules<-sort(rules, decreasing=TRUE,by="confidence")
> inspect(rules[1:5])
lhs rhs support confidence lift
20 {yogurt} => {whole milk} 0.05602440 0.4016035 1.571735
19 {yogurt} => {other vegetables} 0.04341637 0.3112245 1.608457
18 {yogurt} => {rolls/buns} 0.03436706 0.2463557 1.339363
15 {yogurt} => {tropical fruit} 0.02928317 0.2099125 2.000475
17 {yogurt} => {soda} 0.02735130 0.1960641 1.124368
```

Using the `lift` criteria also, product recommendation can be designed to offer to the customers:

```
> # sorting grocery rules by lift
> inspect(sort(rules, by = "lift")[1:5])
lhs rhs support confidence lift
1 {yogurt} => {curd} 0.01728521 0.1239067 2.325615
8 {yogurt} => {whipped/sour cream} 0.02074225 0.1486880 2.074251
15 {yogurt} => {tropical fruit} 0.02928317 0.2099125 2.000475
4 {yogurt} => {butter} 0.01464159 0.1049563 1.894027
11 {yogurt} => {citrus fruit} 0.02165735 0.1552478 1.875752
```

Finding out the subset of rules based on the availability of some item names can be done using the following code:

```
# finding subsets of rules containing any items
itemname_rules <- subset(rules, items %in% "item name")
inspect(itemname_rules[1:5])
> # writing the rules to a CSV file
> write(rules, file = "groceryrules.csv", sep = ",", quote = TRUE,
row.names = FALSE)
>
> # converting the rule set to a data frame
> groceryrules_df <- as(rules, "data.frame")
```

Summary

In this chapter, we discussed how to create association rules, what all factors determine the rules' existence, and how rules explain the underlying relationship between different items or variables. Also we looked at how we can get most compressed rules based on minimum support and confidence value. The objective of the association rules model was not to come up with rules but to implement the rules in business use cases for generating recommendation for cross-selling and upselling products, and designing campaign bundles based on association. The rules would provide necessary guidance for the store managers to place products and merchandising design in a retail setup. Having said this, in our next chapter, we are going to cover various methods of performing clustering for segmentation, which would provide more insights into product recommendation using clustering methods.

6
Clustering with E-commerce Data

In data mining literature, clustering plays an important role in bringing insights from a dataset that is actionable and provides important business directions. In simple language, clustering aims at bringing similar observations such as similar customers, similar patients, similar users, and so on. Clustering techniques are not limited to the domain of retail but can be extended to any domain. Segmentation is no more limited to the retail or e-commerce domain; it is also applicable to all domains and industries.

In this chapter, we are going to learn the following things:

- What is segmentation?
- How can clustering be applied to perform segmentation?
- What are the methods used for clustering?
- A comparative view of the various methods
- A practical project on segmentation

In this chapter, you will know the basics of segmentation using various clustering methods. There are different methods used to perform clustering. The following examples clarify where and how clustering can be used to create segments that can be used to drive business value:

- In the retail/e-commerce industry, it is not possible to understand the behavior of millions of customers to design the campaign, marketing strategy, and sales promotion engagements. These customers actually belong to a finite group of customers displaying similar behavior within a group and dissimilar behavior between groups.

- In the telecommunications industry, the positioning of towers to optimize user experience is driven by segmentation exercise. Also the design of telecom plans, such as custom offers (especially for students, older people, professionals, and so on), based on the usage of services are also made using clustering methods.
- In the healthcare industry, to decide the size of hospital beds for different departments, such as emergency care, acute care, and so on, clustering exercise helps. Also, reporting of various diseases across different states can be used to create a cluster of similar regions or places where a set of procedures can be adopted.
- In governance, law enforcement agencies strengthen their presence in those areas where crime evidence/instance is more. So clustering techniques are used to create groups of cities such as high, medium, and low-crime areas. Based on the data, they can align their forces accordingly.
- In insurance, clustering is used to decide differential premiums by grouping different cities based on parameters such as demographic information, geolocation, and risk zone.

Understanding customer segmentation

In standard data mining practice, customer segmentation is a way of dividing all customers into various subgroups relevant to the business and the business problem. That subgroup creation can be done either by using a subjective approach or by keeping the business objective in mind. In different industries, customer segmentation has different applications; for example, in retail, it is important to know the purchase behavior of customers, and different subgroups displaying unique purchase behavior is relevant to the business.

Why understanding customer segmentation is important

In the retail and e-commerce industry, offers, campaigns, loyalty programs, and discount strategies work based on the purchase behavior of the subgroup of customers. In other industries, sales, marketing strategy, and business plans run keeping in mind the customer's behavior, and the behavior drives the sales. That unique customer behavior one can be understood by performing segmentation on the data.

How to perform customer segmentation?

Having discussed various benefits of performing customer segmentation, it is important to know how to perform customer segmentation. There are two broad methods used to perform customer segmentation:

- Customer segmentation using clustering-based methods
- Customer segmentation using the **recency, frequency, and monetary** (**RFM**) model

In this chapter, we will discuss the clustering-based approach to perform customer segmentation; the second method is not in the scope of this chapter.

Various clustering methods available

All clustering algorithms can be classified into four groups, as follows:

- **Hierarchical clustering**: Using this method, a number of clusters are prepared from the dataset, forming a hierarchy, and the clusters are then grouped using a tree structure so that the entire dataset can be represented as a single tree structure. The advantage of this method is that unlike partitioning-based clustering, we don't have to specify the number of clusters to be created from the dataset.
- **Partitioning clustering**: In this approach, the dataset is divided into a finite group of clusters based on a distance function in such a way that the within-cluster similarity is maximum and between-cluster similarity is minimum. K-means, which is a popular method in data mining practice, falls under this category.
- **Model-based clustering**: Using the data model, a group of clusters is created from the dataset; the methods used in the model-based clustering approach can be divided into two groups:
    - Maximum likelihood estimation
    - Bayesian estimation
- **Other clustering algorithms**: There are other clustering techniques available such as fuzzy clustering, bagged clustering, and so on. Fuzzy clustering uses a fuzzy logic on the data so that the same data point can be a part of more than one cluster. In other words, there is no exclusivity in terms of cluster membership. This method is basically different from the other three forms of clustering, where a data point belongs to a single cluster. Another algorithm that falls under this group is known as **self-organizing maps** (**SOM**).

Before analyzing various methods used for clustering of similar objects or observations from a dataset, it is important to decide on a metric or distance method to measure the similarity or dissimilarity between the observations. Based on certain predefined characteristics known as input features, how one observation is different from or close to another observation can be known by using this metric.

A similarity metric measures the degree of closeness of data points and a distance metric measures how different the data points are. Both similarity and distance metric calculations are part of the clustering process. Let's take a look at various similarity/distance measures:

- **Euclidean distance**: The formula to compute the distance between two data points x and y is as follows:

$$Dist(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Manhattan distance**: This is the formula used to compute the Manhattan distance:

$$Dist(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Cosine similarity**: The formula used to compute the cosine similarity is as follows:

$$Sim(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Now let's start segmenting the dataset using various clustering methods by taking a dataset from healthcare (`PimaIndiansDiabetes.csv [ref: 1]`) and a dataset from retail/e-commerce (`Wholesalecustomers.csv [ref: 1]`). Before running a clustering exercise, data sufficiency conditions need to be checked.

K-means clustering

K-means clustering is an unsupervised learning algorithm that tries to combine similar objects into a group in such a way that the within-group similarity should be maximum and the between-group object similarity should be minimum. "Object" here implies the data points that are entered into an algorithm. The within-group similarity is computed based on a centrality measure called centroid (arithmetic mean) and a distance function that measures how close the objects within a group are to the center. The "K" in K-means clustering implies the number of clusters that a user might be interested in. The following steps need to be followed for k-means clustering. Since the mean is used as a measure of estimating the centroid, it is not free from the presence of extreme observations or outliers. Hence, it is required to check the presence of outliers in a dataset before running k-means clustering.

1. **Checking the presence of outliers**: Let's use the boxplot method of identifying the presence of any outliers in the `Wholesalecustomers.csv` dataset:

```
> r_df<-read.csv("Wholesalecustomers.csv")
> ####Data Pre-processing
> par(mfrow=c(2,3))
> apply(r_df[,c(-1,-2)],2,boxplot)
```

The dotted points after the whisker line in each preceding boxplot for all the variables indicate that there are lots of outliers in the dataset. The capping method is mostly used by practitioners to remove the outliers from the dataset. If any data point exceeds the quantiles, 90th percentile, 95th percentile, or 99th percentile for each variable, then those data points need to be capped at the respective percentiles:

```
> #computing quantiles
> quant<-function(x){quantile(x,probs=c(0.95,0.90,0.99))}
> out1<-sapply(r_df[,c(-1,-2)],quant)
```

Now, this `quant` function computes the three percentiles and the same are applied to the dataset, excluding the first two variables. The following code is used to modify the variables:

```
> #removing outliers
> r_df$Fresh<-ifelse(r_df$Fresh>=out1[2,1],out1[2,1],r_df$Fresh)
> r_df$Milk<-ifelse(r_df$Milk>=out1[2,2],out1[2,2],r_df$Milk)
> r_df$Grocery<-ifelse(r_df$Grocery>=out1[2,3],out1[2,3],r_df$Grocery)
> r_df$Frozen<-ifelse(r_df$Frozen>=out1[2,4],out1[2,4],r_df$Frozen)
> r_df$Detergents_Paper<-
ifelse(r_df$Detergents_Paper>=out1[2,5],out1[2,5],r_df$Detergents_Paper)
> r_df$Delicassen<-
ifelse(r_df$Delicassen>=out1[2,6],out1[2,6],r_df$Delicassen)
```

Next, let's check the dataset to find out whether the outliers have been removed, using the same boxplot method:

```
> #Checking outliers after removal
> apply(r_df[,c(-1,-2)],2,boxplot)
```

From the preceding graph, we can conclude that all the outliers have been removed by capping the values at 90th percentile value for all the variables.

2. **Scaling the dataset**: Inclusion of only continuous variables in the K-means clustering exercise does not guarantee that all the variables would be on the same scale. For example, the age of a customer ranges from 0-100 and the income of a customer ranges from 0-100,000. Since both units of measurement are different, they might fall into different clusters, which becomes more complex when we have more data points to be represented. In order to compare segments in a clustering process, all the variables need to be standardized on a scale; here we have applied standard normal transformation (Z-transformation) for all the variables:

```
> r_df_scaled<-as.matrix(scale(r_df[,c(-1,-2)]))
> head(r_df_scaled)
Fresh Milk Grocery Frozen Detergents_Paper Delicassen
[1,] 0.2351056 1.2829174 0.11321460 -0.96232799 0.1722993 0.1570550
[2,] -0.4151579 1.3234370 0.45658819 -0.30625122 0.4120701 0.6313698
[3,] -0.4967305 1.0597962 0.13425842 -0.03373355 0.4984496 1.8982669
[4,] 0.3041643 -0.9430315 -0.45821928 1.66113143 -0.6670924 0.6443648
[5,] 1.3875506 0.1657331 0.05110966 0.60623797 -0.1751554 1.8982669
[6,] -0.1421677 0.9153464 -0.30338465 -0.77076036 -0.1681831 0.2794239
```

3. **Choose initial cluster seeds**: The selection of initial random seeds decides the number of iterations required for convergence of the model. Typically, initial cluster seeds chosen at random are temporary means of the clusters. The objects closer to the centroid measured by the distance function are assigned the cluster membership. With the addition of a new member into the cluster, again the centroid is computed and each seed value is replaced by the respective cluster centroid. In K-means clustering, this process of adding more objects to the group and hence updating the centroid goes on until the centroids are not moving and the objects are not changing the group membership.
4. **Decide the number of cluster K**: In the R programming language, there are two libraries, `Rcmdr` (R Commander) and `stats`, that support K-means clustering and they use two different approaches. The `Rcmdr` library needs to be loaded to run the algorithm, but stats is already built into base R. To compute the number of clusters required to represent the data using K-means algorithm is decided by the scree plot using the following formula:

```
> library(Rcmdr)
> > sumsq<-NULL
> #Method 1
> par(mfrow=c(1,2))
> for (i in 1:15) sumsq[i] <-
sum(KMeans(r_df_scaled,centers=i,iter.max=500,
num.seeds=50)$withinss)
>
```

```
> plot(1:15,sumsq,type="b", xlab="Number of Clusters",
+ ylab="Within groups sum of squares",main="Screeplot using Rcmdr")
>
> #Method 2
> for (i in 1:15) sumsq[i] <-
sum(kmeans(r_df_scaled,centers=i,iter.max=5000,
        algorithm = "Forgy")$withinss)
>
> plot(1:15,sumsq,type="b", xlab="Number of Clusters",
+ ylab="Within groups sum of squares",main="Screeplot using Stats")
```

The following graph explains the scree plot. Using both approaches, the ideal number of clusters is 4, decided by looking at the "elbow" point on the scree plot. The vertical axis shows the within-group sum of squares dropping as the cluster number increases on the horizontal axis:

Once the number of clusters is decided, the cluster centers and distance can be computed, and the following code explains the functions to use in K-means clustering. The objective here is to classify the objects in such a way that they are as much dissimilar as possible from one group to another group and as much similar as possible within each group. The following function can be used to compute the within-group sum of squares from the cluster centers to the individual objects:

$$Sumsq = \sum_{i=1}^{n} \left(x_i - c_i \right)^2$$

The sum of squares from each of the clusters is computed and then added up to compute the total within-group sum of squares for the dataset. The objective here is to minimize the within-group sum of squares total.

5. **Grouping based on minimum distance**: The K-means syntax and explanation for the arguments are as follows:

```
kmeans(x, centers, iter.max = 10, nstart = 1,
algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
"MacQueen"), trace=FALSE)
```

x	**Numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns)**
`centers`	Either the number of clusters, say *k*, or a set of initial (distinct) cluster centers. If it's a number, a random set of (distinct) rows in *x* is chosen as the initial centers.
`iter.max`	The maximum number of iterations allowed.
`nstart`	If the center is a number, how many random sets should be chosen?
`algorithm`	Character. This may be abbreviated. Note that `"Lloyd"` and `"Forgy"` are alternative names for one algorithm.

Table 1: The description for the K-means syntax

```
> #Kmeans Clustering
> library(cluster);library(cclust)
> set.seed(121)
> km<-kmeans(r_df_scaled,centers=4,nstart=17,iter.max=50000, algorithm
=
"Forgy",trace = T)
```

The scaled dataset is used to create clusters, with four clusters and 17 initial starting points. Given the 50,000 iterations with the Forgy algorithm, the following results are obtained from the model:

```
#checking results
summary(km)
```

```
Length Class Mode
cluster 440 -none- numeric
centers 24 -none- numeric
totss 1 -none- numeric
withinss 4 -none- numeric
tot.withinss 1 -none- numeric
betweenss 1 -none- numeric
size 4 -none- numeric
iter 1 -none- numeric
ifault 0 -none- NULL
> km$centers
Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1 0.7272001 -0.4741962 -0.5839567 1.54228159 -0.64696856 0.13809763
2 -0.2327058 -0.6491522 -0.6275800 -0.44521306 -0.55388881 -0.57651321
3 0.6880396 0.6607604 0.3596455 0.02121206 -0.03238765 1.33428207
4 -0.6025116 1.1545987 1.3947616 -0.45854741 1.55904516 0.09763056
> km$withinss
[1] 244.3466 324.8674 266.3632 317.5866
```

The following code explains how to attach cluster information with the dataset and the profiling of various clusters with their respective group means:

```
> #attaching cluster information
> Cluster<-cbind(r_df,Membership=km$cluster)
> aggregate(Cluster[,3:8],list(Cluster[,9]),mean)
Group.1 Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1 1 16915.946 2977.868 3486.071 6123.576 558.9524 1320.4940
2 2 8631.625 2312.926 3231.096 1434.122 799.2500 660.5957
3 3 16577.977 7291.414 9001.375 2534.643 2145.5738 2425.0954
4 4 5440.073 9168.309 15051.573 1402.660 6254.0670 1283.1252
```

Now, let's look at visualization of the groups/clusters using a scatter plot:

From the preceding graph, we can see that all the four clusters are overlapping each other to a certain extent; however, the overall objective of a clustering exercise is to come up with non-overlapping groups. There are six variables; hence, six dimensions represented in a two-dimensional space show 65.17% variability explained by two components. This is acceptable because the degree of overlap is less. If the degree of overlap is more, then post clustering, the similar groups can be merged as part of a profiling exercise. Profiling is an exercise performed after K-means clustering to generate insights for business users; hence, while generating insights, similar groups can be merged.

6. **Predicting cluster membership for new data**: Once the K-means clustering model is created, using that model, cluster membership for a new dataset can be created by using the following predict function:

```
> #Predicting new data for KMeans
> predict.kmeans <- function(km, r_df_scaled)
+ {k <- nrow(km$centers)
+ d <- as.matrix(dist(rbind(km$centers, r_df_scaled)))[-(1:k),1:k]
+ n <- nrow(r_df_scaled)
+ out <- apply(d, 1, which.min)
+ return(out)}
```

For the same dataset, we can predict the cluster membership, and the actual cluster membership versus the predicted cluster membership can be represented in the following table:

```
> #predicting cluster membership
> Cluster$Predicted<-predict.kmeans(km,r_df_scaled)
> table(Cluster$Membership,Cluster$Predicted)
1 2 3 4
1 84 0 0 0
2 0 188 0 0
3 0 0 65 0
4 0 0 0 103
```

7. **Implementing K-means in live applications**: For implementation of K-means clustering with any other statistical software and web applications, PMML script can be used. **PMML** stands for **Predictive Model Markup Language**, which is recognized as an industry standard for deploying predictive models:

```
#pmml code
library(pmml);
library(XML);
pmml(km)
```

When we save a model in PMML format, it prepares an XML script for the model where the model parameters are embedded in the script. It is written in such a way that any other external statistical software can read the model.

Hierarchical clustering

There are two different ways of implementing the hierarchical clustering algorithm, but one thing is common; both use a distance measure to estimate the similarity between cluster members:

- **Agglomerative method**: Bottom-up approach
- **Divisive method**: Top-down approach

To perform hierarchical clustering, the input data has to be in a distance matrix form. The method selected has to be from one of `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"ward.D"`, `"ward.D2"`, `"centroid"`, or `"median"`.

The hierarchical algorithm follows these steps:

1. Start with N singleton clusters (nodes) labeled $-1, \ldots, -N$, which represent the initial set of input points.
2. Find a pair of nodes / cluster with minimal distance among all pairwise distances using a distance function.
3. Combine the two nodes into a new node and remove the two old nodes. The new nodes are labeled consecutively $1, 2, \ldots$.
4. The distances from the new node to all other nodes are determined by the method parameter.
5. Repeat $N - 1$ times from step 2 until there is one big node that contains all original input points.

If the method chosen to perform hierarchical clustering is ward, then the distance measure to be used for clustering is Euclidean. The following code shows how to perform agglomerative hierarchical clustering:

```
> hfit<-hclust(dist(r_df_scaled,method = "euclidean"),method="ward.D2")
> par(mfrow=c(1,2))
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "manhattan"),method="mcquitty")
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "minkowski"))
> plot(hfit,hang=-0.005,cex=0.7)
> hfit<-hclust(dist(r_df_scaled,method = "canberra"))
> plot(hfit,hang=-0.005,cex=0.7)
```

Here is the first dendrogram graph:

Here is the second dendrogram graph:

Various methods are used to show how to perform agglomerative hierarchical clustering in the preceding two dendrogram graphs. The parameter `method` is one of the strings `single`, `centroid`, `median`, `ward`, and the argument method is one of the strings `single`,

`complete`, `average`, `mcquitty`, `centroid`, `median`, `ward.D`, `ward.D2`. Each method is different. Single linkage uses the smallest minimum pairwise distance to merge two clusters. Complete linkage uses the similarity between the dissimilar objects in two different clusters. Likewise each method has a unique way to compute the similarity between different clusters:

```
> summary(hfit)
Length Class Mode
merge 878 -none- numeric
height 439 -none- numeric
order 440 -none- numeric
labels 0 -none- NULL
method 1 -none- character
call 3 -none- call
dist.method 1 -none- character
```

Divisive hierarchical clustering can be performed using the `diana` function from the library `cluster`. The syntax is shown as follows:

```
diana(x, diss = inherits(x, "dist"), metric = "euclidean", stand = FALSE,
keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

To apply the `diana` function, the input data should be in matrix or dataframe type. The `diana` function accepts both a distance matrix as well as a dataframe as an input. There is a logical flag argument meant to control the input structure to the function. The `metric` argument is a character string specifying the metric to be used to calculate distance; it has two options, `"euclidean"` and `"manhattan"`. Euclidean distances are root sums of squares of differences and manhattan distances are the sums of absolute differences, for which the mathematical formula has already been explained in this chapter. If `x`, the input dataset, is already a dissimilarity matrix, then this `diss` argument will be ignored from execution. The argument `stand` if takes the value as true, this implies that the raw dataset is used and further the distance matrix needs to be created.

The `diana` function constructs a hierarchy of clusters, as the name suggests, starting with one large cluster containing all observations. Clusters are divided until each cluster contains only a single observation. At each hierarchy, the cluster with the largest diameter based on the distance from the cluster center to the outermost data point as measured by the distance function is selected:

```
dfit<-diana(r_df,diss=F,metric = "euclidean",stand=T,keep.data = F)
summary(dfit)
plot(dfit)
```

The `plot` command on divisive clustering provides two graph options, a banner indicating the divisive coefficient and a dendrogram displaying the tree structure:

For the hierarchical clustering algorithms, one additional step is to cut the dendrogram tree structure into a finite group of clusters so that the results from various clustering methods can be compared. The following function is used to cut the tree structure:

```
> #cutting the tree into groups
> g_hfit<-cutree(hfit,k=4)
> plot(hfit)
> table(g_hfit)
g_hfit
1  2  3  4
77 163 119 81
> rect.hclust(hfit,k=4,border = "blue")
```

This graph shows the tree structure:

Model-based clustering

There is a library in R known as `mclust` that provides Gaussian finite mixture models fitted viaEM algorithm for model-based clustering, including Bayesian regularization. In this section, we discuss the syntax for `Mclust` and the outcome, and the rest of the results can be analyzed using profiling information. The model-based approach applies a set of data models, the maximum likelihood estimation method, and Bayes criteria to identify the cluster, and the `Mclust` function estimates the parameters through the expectation maximization algorithm.

The `Mclust` package provides various data models based on three parameters: *E*, *V*, and *I*. The model identifiers use the three letters *E*, *V*, and *I* to code geometric characteristics such as volume, shape, and orientation of the cluster. *E* means equal, *I* implies identity matrix in specifying the shape, and *V* implies varying variance. Keeping in mind these three letters, there is a set of models that is by default run using the `mclust` library, and those models can be classified into three groups such as spherical, diagonal and ellipsoidal. The details are given in the following table:

Model code	Description
`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
`EEV`	Ellipsoidal, equal volume and shape
`VEV`	Ellipsoidal, equal shape
`VVV`	Ellipsoidal' varying volume, shape, and orientation

Table 2: Model description using finite mixture approach

Now let's use the `Mclust` method of doing clustering and grouping of observations based on the *E*, *V*, and *I* parameters:

```
> clus <- Mclust(r_df[,-c(1,2)])
> summary(clus)
----------------------------------------------------
```

```
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
Mclust VEI (diagonal, equal shape) model with 9 components:
log.likelihood n df BIC ICL
-23843.48 440 76 -48149.56 -48223.43
Clustering table:
1 2 3 4 5 6 7 8 9
165 21 40 44 15 49 27 41 38
```

From the preceding summary, there are nine clusters created using the VEI model specification and the BIC is lowest for this model. Let's also plot all other model specifications using a plot:

From the preceding graph, it is clear that the VEI model specification is the best. Using the following code, we can print the cluster membership as well as the number of rows in each of the clusters:

```
# The clustering vector:
clus_vec <- clus$classification
clus_vec
clust <- lapply(1:3, function(nc) row.names(r_df)[clus_vec==nc])
clust # printing the clusters
```

Using the `summary` command with parameter is true, mean, variance and probability for each variable we can compute:

```
summary(clus, parameters = T)
```

Other cluster algorithms

Self-organizing map (**SOM**) is another approach to clustering; it uses the visual method of representing data. SOM is basically used to derive unique nodes. SOM creates a network structure where the data objects would be connected by nodes and edges. Using those nodes, the similarity between different objects can be drawn to create clustering solutions. The `kohonen` library contains the SOM function:

```
> library(kohonen)
> som_grid <- somgrid(xdim = 20, ydim=20, topo="hexagonal")
> som_model <- som(r_df_scaled,
+ grid=som_grid,
+ rlen=100,
+ alpha=c(0.05,0.01),
+ keep.data = TRUE,
+ n.hood="circular")
plot(som_model, type="changes",col="blue")
```

The preceding graph explains how the mean distance to the closest unit changes with increase in the iteration level:

```
> plot(som_model, type="count")
```

The preceding graph displays the counts plot using the SOM approach. The following graph displays the neighbor distance plot. The map is colored by different variables and the circles with colors denote where those input variables fall in n-dimensional space. When the plot type is counts, it shows the number of objects mapped to individual units. The units in gray color indicate empty. When the type is distance neighbors, it shows the sum of distances to all immediate neighbors. This is shown in the following graph:

```
> plot(som_model, type="dist.neighbours")
```

```
> plot(som_model, type="codes")
```

Finally, the next graph displays the code using all the variables:

In R, the `kohonen` library is used to create self-organizing maps using hierarchical clustering. When the type of graph is codes, as it is shown in the preceding graph, this shows the code book vectors; each color indicates a feature from the input dataset. The preceding results and graphs show how to create a clustering solution and how clustering can be visualized using the SOM plot function. SOM is a data visualization process used to show similar objects in a dataset based on input data vectors. Hence, the conclusion from the results would be how clusters or segments can be created by visual inspection.

Comparing clustering methods

Here are some of the advantages and disadvantages of various clustering methods:

For K-means clustering:

Merits and demerits of K-means clustering	
It is simple to understand and easy to interpret	It is not based on robust methodology. The big problem is initial random points
It is more flexible in comparison to other clustering algorithms	Different random points display different results. Hence, more iteration is required.
Performs well on a large number of input features	Since more iteration is required, it may not be efficient for large datasets from a computational efficiency point of view. It would require more memory to process.

For SOM:

Merits and Demerits of SOM	
Simple to understand visually	Only works on continuous data
New data points can be predicted visually	Difficult to represent when the dimensions increase

References

Lichman, M. (2013). *UCI Machine Learning Repository* (`http://archive.ics.uci.edu/ml`). Irvine, CA: University of California, School of Information and Computer Science.

Summary

In this chapter, we discussed various methods of segmentation for unsupervised data mining problems and their implementations. Clustering is a subjective segmentation method and it requires further investigation such as profiling by cluster and calculating different measures such as mean and median for different variables. If any cluster displays closeness to another cluster and from business point of view they make sense, then they can be combined together. We discussed which model to use where and what the data requirement for each of the models is. In this chapter, we specifically highlighted the importance of data visualization when it comes to representing clustering solutions.

7

Building a Retail Recommendation Engine

In this age of Internet, everything available over the Internet is not useful for everyone. Different companies and entities use different approaches to find out relevant content for their audiences. People started building algorithms to construct a relevance score, based on which recommendations could be built and suggested to the users. In my day to day life, every time I see an image on Google, 3-4 other images are recommended to me by Google, every time I look for some videos on YouTube, 10 more videos are recommended to me, every time I visit Amazon to buy some products, 5-6 products are recommended to me, and, every time I read one blog or article, a few more articles and blogs are recommended to me. This is an evidence of algorithmic forces at play to recommend certain things based on user's preferences or choices, since the user's time is precious and content available over Internet is unlimited. Hence, recommendation engine helps organizations customize their offerings based on the user's preferences so that the user does not have to spend time in exploring what is required.

In this chapter, the reader will learn the following things and their implementation using R programming language:

- What is recommendation and how does that work
- Types and methods for performing recommendation
- Implementation of product recommendation using R

What is recommendation?

Recommendation is a method or technique by which an algorithm detects what the user is looking at. Recommendation can be of products, services, food, videos, audios, images, news articles, and so on. What recommendation is can be made much clearer if we look around and note what we observe over the Internet day in and day out. For example, consider news aggregator websites and how they recommend articles to users. They look at the tags, timing of the article loaded into Internet, number of comments, likes, and shares for that article, and of course geolocation, among other details, and then model the metadata information to arrive at a score. Based on that score, they start recommending articles to new users. Same thing happens when we watch a video on YouTube. Similarly, beer recommendation system also works; the user has to select any beer at random and based on the first product chosen, the algorithm recommends other beers based on similar users' historical purchase information. Same thing happens when we buy products from online e-commerce portals.

Types of product recommendation

Broadly, there are four different types of recommendation systems that exist:

- Content-based recommendation method
- Collaborative filtering-based recommendation method
- Demographic segment-based recommendation method
- Association rule based-recommendation method

In content-based method, the terms, concepts which are also known as keywords, define the relevance. If the matching keywords from any page that the user is reading currently are found in any other content available over Internet, then start calculating the term frequencies and assign a score and based on that score whichever is closer represent that to the user. Basically, in content-based methods, the text is used to find the match between two documents so that the recommendation can be generated.

In collaborative filtering, there are two variants: user-based collaborative filtering and item-based collaborative filtering.

In user-based collaborative filtering method, the user-user similarity is computed by the algorithm and then based on their preferences the recommendation is generated. In item based collaborative filtering, item-item similarity is considered to generate recommendation.

In demographic segment based category, the age, gender, marital status, and location of the users, along with their income, job, or any other available feature, are taken into consideration to create customer segments. Once the segments are created then at a segment level the most popular product is identified and recommended to new users who belong to those respective segments.

Techniques to perform recommendation

To filter out abundant information available online and recommend useful information to the user at first hand is the prime motivation for creating recommendation engine. So how does the collaborative filtering work? Collaborative filtering algorithm generates recommendations based on a subset of users that are most similar to the active user. Each time a recommendation is requested, the algorithm needs to compute the similarity between the active user and all the other users, based on their co-rated items, so as to pick the ones with similar behaviour. Subsequently, the algorithm recommends items to the active user that are highly rated by his/her most similar users.

In order to compute the similarities between users, a variety of similarity measures have been proposed, such as Pearson correlation, cosine vector similarity, Spearman correlation, entropy-based uncertainty measure, and mean square difference. Let's look at the mathematical formula behind the calculation and implementation using R.

The objective of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users.

Memory-based algorithms utilize the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbours, who have a history of agreeing with the target user (that is, they either rate the same items similarly or they tend to buy similar sets of items). Once a neighbourhood of users is formed, these systems use different algorithms to combine the preferences of neighbours to produce a prediction or top-N recommendation for the active user. The techniques, also known as nearest-neighbour or user-based collaborative filtering, are very popular and widely used in practice.

Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items. The model building process is performed by different machine learning algorithms, such as Bayesian network, clustering, and rule-based approaches.

One critical step in the item-based collaborative filtering algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in similarity computation between two items *i* and *j* is to first isolate the users who have rated both of these items and then to apply a similarity computation technique to determine the similarity *S(i,j)*.

There are a number of different ways to compute the similarity between items. Here we present three such methods. These are cosine-based similarity, correlation-based similarity, and adjusted-cosine similarity:

- **Cosine-based similarity**: Two items are thought of as two vectors in the *m* dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Preceding is the formula for computing cosine similarity between two vectors. Let's take a numerical example to compute the cosine similarity:

Cos (d1, d2) = 0.44

d1	**d2**	**d1*d2**	**||d1||**	**||d2||**	**||d1||*||d2||**
6	1	6	36	1	36
4	0	0	16	0	0
5	3	15	25	9	225
1	2	2	1	4	4
0	5	0	0	25	0
2	0	0	4	0	0
4	6	24	16	36	576
0	5	0	0	25	0
3	0	0	9	0	0
6	1	6	36	1	36

		53	143	101	877
			11.95826	10.04988	

Table 5: Results of cosine similarity

So cosine similarity between `d1` and `d2` is 44%.

Cos (d1, d2) = 0.441008707

- **Correlation-based similarity**: Similarity between two items *i* and *j* is measured by computing the Pearson-r correlation `corr`. To make the correlation computation accurate, we must first isolate the co-rated cases.
- **Adjusted cosine similarity**: One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the rows of the matrix, but in case of the item-based CF the similarity is computed along the columns, that is, each pair in the co-rated set corresponds to a different user. Computing similarity using basic cosine measure in item-based case has one important drawback – the difference in rating scale between different users is not taken into account. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. The example is given in *Table 1*, where the co-rated set belongs to different users, users are in different rows of the table.

Assumptions

The dataset that we are going to use for this chapter contains 100 jokes and 5000 users, which is a built-in dataset in the library recommenderlab. The dataset contains real ratings expressed by users in a scale of -10 to +10, with -10 being the worst and +10 being the best joke. We will implement various recommendation algorithms using this dataset. Using this dataset, the objective is to recommend jokes to new users based on their past preferences.

What method to apply when

Though there are four different types of recommendation methods, now which one to apply when. If the products or items are bought in a batch, then it is preferred by practitioners to apply association rules, which is also known as market basket analysis. In a retail or e-commerce domain, the items are generally purchased in a lot. Hence, when a user adds a certain product to his/her cart, other products can be recommended to him/her based on the aggregate basket component as reflected by majority of the buyers.

If the ratings or reviews are explicitly given for a set of items or products, it makes sense to apply user based collaborative filtering. If some of the ratings for few items are missing still the data can be imputed, once the missing ratings predicted, the user similarity can be computed and hence recommendation can be generated. For user-based collaborative filtering, the data would look as follows:

User	Item1	Item2	Item3	Item4	Item5	Item6
user1	-7.82	8.79	-9.66	-8.16	-7.52	-8.5
user2	4.08	-0.29	6.36	4.37	-2.38	-9.66
user3					9.03	9.27
user4		8.35			1.8	8.16
user5	8.5	4.61	-4.17	-5.39	1.36	1.6
user6	-6.17	-3.54	0.44	-8.5	-7.09	-4.32
user7					8.59	-9.85
user8	6.84	3.16	9.17	-6.21	-8.16	-1.7
user9	-3.79	-3.54	-9.42	-6.89	-8.74	-0.29
user10	3.01	5.15	5.15	3.01	6.41	5.15

Table 1: User-based item sample dataset

If the binary matrix is given as an input, where the levels represent whether the product is bought or not, then it is recommended to apply item-based collaborative filtering. The sample dataset is given next:

User	Item1	Item2	Item3	Item4	Item5	Item6
user1	1	1	1	1	1	0
user2	1	1	0	0	1	
user3	0	0		1	0	0
user4	0	1	1		1	1
user5	1	0		1	0	
user6	0	0	1			0
user7		1	0	0	1	1
user8	1	0	1	1	0	

user9	0	1	0	0		0
user10	1	0	1	1	1	1

Table 2: Item-based collaborative filtering sample dataset

If the product description details and the item description are given and the user search query is collected, then the similarity can be measured using content-based collaborative filtering method:

Title	Search query
Celestron LAND AND SKY 50TT Telescope	good telescope
(S5) 5-Port Mini Fast Ethernet Switch S5	mini switch
(TE-S16)Tenda 10/100 Mbps 16 Ports Et...	ethernet ports
(TE-TEH2400M) 24-Port 10/100 Switch	ethernet ports
(TE-W300A) Wireless N300 PoE Access P...	ethernet ports
(TE-W311M) Tenda N150 Wireless Adapte...	wireless adapter
(TE-W311M) Tenda N150 Wireless Adapte...	wireless adapter
(TE-W311MI) Wireless N150 Pico USB Ad...	wireless adapter
101 Lighting 12 Watt Led Bulb - Pack Of 2	led bulb
101 Lighting 7 Watt Led Bulb - Pack Of 2	led bulb

Table 3: Content-based collaborative filtering sample dataset

Limitations of collaborative filtering

User-based collaborative filtering systems have been very successful in past, but their widespread use has revealed some real challenges, such as:

- **Sparsity**: In practice, many commercial recommender systems are used to evaluate large item sets (for example, Amazon.com recommends books and CDNow.com recommends music albums). In these systems, even active users may have purchased well under 1% of the items 1%). Accordingly, a recommender system based on nearest neighbor algorithms may be unable to make any item recommendations for a particular user. As a result, the accuracy of recommendations may be poor.

- **Scalability**: Nearest neighbor algorithms require computation that grows with both the number of users and the number of items. With millions of users and items, a typical web-based recommender system running existing algorithms will suffer serious scalability problems.

Practical project

The dataset contains a sample of 5000 users from the anonymous ratings data from the Jester Online Joke Recommender System collected between April 1999 and May 2003 (Goldberg, Roeder, Gupta, and Perkins 2001). The dataset contains ratings for 100 jokes on a scale from -10 to 10. All users in the data set have rated 36 or more jokes. Let's load the `recommenderlab` library and the `Jester5K` dataset:

```
> library("recommenderlab")
> data(Jester5k)
> Jester5k@data@Dimnames[2]
[[1]]
[1] "j1" "j2" "j3" "j4" "j5" "j6" "j7" "j8" "j9"
[10] "j10" "j11" "j12" "j13" "j14" "j15" "j16" "j17" "j18"
[19] "j19" "j20" "j21" "j22" "j23" "j24" "j25" "j26" "j27"
[28] "j28" "j29" "j30" "j31" "j32" "j33" "j34" "j35" "j36"
[37] "j37" "j38" "j39" "j40" "j41" "j42" "j43" "j44" "j45"
[46] "j46" "j47" "j48" "j49" "j50" "j51" "j52" "j53" "j54"
[55] "j55" "j56" "j57" "j58" "j59" "j60" "j61" "j62" "j63"
[64] "j64" "j65" "j66" "j67" "j68" "j69" "j70" "j71" "j72"
[73] "j73" "j74" "j75" "j76" "j77" "j78" "j79" "j80" "j81"
[82] "j82" "j83" "j84" "j85" "j86" "j87" "j88" "j89" "j90"
[91] "j91" "j92" "j93" "j94" "j95" "j96" "j97" "j98" "j99"
[100] "j100"
```

The following image shows the distribution of real ratings given by 2000 users:

```
> data<-sample(Jester5k,2000)
> hist(getRatings(data),breaks=100,col="blue")
```

The input dataset contains the individual ratings; normalization function reduces the individual rating bias by centering the row, which is a standard z-score transformation. Subtracting each element from the mean and then dividing by standard deviation. The following graph shows normalized ratings for the preceding dataset:

```
> hist(getRatings(normalize(data)),breaks=100,col="blue4")
```

To create a recommender system:

A recommendation engine is created using the `recommender()` function. A new recommendation algorithm can be added by the user using the `recommenderRegistry$get_entries()` function:

```
> recommenderRegistry$get_entries(dataType = "realRatingMatrix")
$IBCF_realRatingMatrix
Recommender method: IBCF
Description: Recommender based on item-based collaborative filtering (real
data).
Parameters:
k method normalize normalize_sim_matrix alpha na_as_zero minRating
1 30 Cosine center FALSE 0.5 FALSE NA
$POPULAR_realRatingMatrix
Recommender method: POPULAR
Description: Recommender based on item popularity (real data).
Parameters: None
$RANDOM_realRatingMatrix
Recommender method: RANDOM
Description: Produce random recommendations (real ratings).
Parameters: None
$SVD_realRatingMatrix
Recommender method: SVD
Description: Recommender based on SVD approximation with column-mean
imputation (real data).
Parameters:
k maxiter normalize minRating
1 10 100 center NA
$SVDF_realRatingMatrix
Recommender method: SVDF
Description: Recommender based on Funk SVD with gradient descend (real
data).
Parameters:
k gamma lambda min_epochs max_epochs min_improvement normalize
1 10 0.015 0.001 50 200 1e-06 center
minRating verbose
1 NA FALSE
$UBCF_realRatingMatrix
Recommender method: UBCF
Description: Recommender based on user-based collaborative filtering (real
data).
Parameters:
method nn sample normalize minRating
1 cosine 25 FALSE center NA
```

The preceding `Registry` command helps in identifying the methods available in `recommenderlab`, parameters for the model.

There are six different methods for implementing recommender system: popular, item-based, user based, PCA, random, and SVD. Let's start the recommendation engine using popular method:

```
> rc <- Recommender(Jester5k, method = "POPULAR")
> rc
Recommender of type 'POPULAR' for 'realRatingMatrix'
learned using 5000 users.
> names(getModel(rc))
[1] "topN" "ratings"
[3] "minRating" "normalize"
[5] "aggregationRatings" "aggregationPopularity"
[7] "minRating" "verbose"
> getModel(rc)$topN
Recommendations as 'topNList' with n = 100 for 1 users.
```

The objects such as top N, verbose, aggregation popularity, and others can be printed using names of the `getmodel()` command.

```
recom <- predict(rc, Jester5k, n=5)
recom
```

To generate recommendation, we can use the `predict` function against the same dataset and validate the accuracy of the predictive model. Here we are generating top 5 recommended jokes to each of the users. The result of the prediction is as follows:

```
> head(as(recom,"list"))
$u2841
[1] "j89" "j72" "j76" "j88" "j83"
$u15547
[1] "j89" "j93" "j76" "j88" "j91"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j89" "j72" "j93" "j76" "j88"
$u15994
character(0)
```

For the same `Jester5K` dataset, let's try to implement **item-based collaborative filtering** (**IBCF**):

```
> rc <- Recommender(Jester5k, method = "IBCF")
```

```
> rc
Recommender of type 'IBCF' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j85" "j86" "j74" "j84" "j80"
$u15547
[1] "j91" "j87" "j88" "j89" "j93"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j78" "j80" "j73" "j77" "j92"
$u15994
character(0)
```

Principal component analysis (**PCA**) method is not applicable for real rating-based dataset because getting correlation matrix and subsequent eigen vector and eigen value calculation would not be accurate. Hence, we will not show its application. Next we are going to show how the random method works:

```
> rc <- Recommender(Jester5k, method = "RANDOM")
> rc
Recommender of type 'RANDOM' for 'ratingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
[[1]]
[1] "j90" "j74" "j86" "j78" "j85"
[[2]]
[1] "j87" "j88" "j74" "j92" "j79"
[[3]]
character(0)
[[4]]
character(0)
[[5]]
[1] "j95" "j86" "j93" "j78" "j83"
[[6]]
character(0)
```

In recommendation engine, the SVD approach is used to predict the missing ratings so that recommendation can be generated. Using **singular value decomposition** (**SVD**) method, the following recommendation can be generated:

```
> rc <- Recommender(Jester5k, method = "SVD")
> rc
Recommender of type 'SVD' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j74" "j71" "j84" "j79" "j80"
$u15547
[1] "j89" "j93" "j76" "j81" "j88"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j80" "j73" "j100" "j72" "j78"
$u15994
character(0)
```

The result from the user-based collaborative filtering is shown next:

```
> rc <- Recommender(Jester5k, method = "UBCF")
> rc
Recommender of type 'UBCF' for 'realRatingMatrix'
learned using 5000 users.
> recom <- predict(rc, Jester5k, n=5)
> recom
Recommendations as 'topNList' with n = 5 for 5000 users.
> head(as(recom,"list"))
$u2841
[1] "j81" "j78" "j83" "j80" "j73"
$u15547
[1] "j96" "j87" "j89" "j76" "j93"
$u15221
character(0)
$u15573
character(0)
$u21505
[1] "j100" "j81" "j83" "j92" "j96"
$u15994
character(0)
```

Now let's compare the results obtained from all the five different algorithms, except PCA, because PCA requires a binary dataset and does not accept real ratings matrix.

Popular	**IBCF**	**Random method**	**SVD**	**UBCF**
> head(as(recom,"list"))	> `head(as(recom,"list"))`	> `head(as(recom,"list"))`	> `head(as(recom,"list"))`	> head(as(recom,"list"))
$u2841	`$u2841`	`[[1]]`	`$u2841`	$u2841
[1] "j89" "j72" "j76" "j88" "j83"	`[1] "j85" "j86" "j74" "j84" "j80"`	`[1] "j90" "j74" "j86" "j78" "j85"`	`[1] "j74" "j71" "j84" "j79" "j80"`	[1] "j81" "j78" "j83" "j80" "j73"
$u15547	`$u15547`	`[[2]]`	`$u15547`	$u15547
[1] "j89" "j93" "j76" "j88" "j91"	`[1] "j91" "j87" "j88" "j89" "j93"`	`[1] "j87" "j88" "j74" "j92" "j79"`	`[1] "j89" "j93" "j76" "j81" "j88"`	[1] "j96" "j87" "j89" "j76" "j93"
$u15221	`$u15221`	`[[3]]`	`$u15221`	$u15221
character(0)	`character(0)`	`character(0)`	`character(0)`	character(0)
$u15573	`$u15573`	`[[4]]`	`$u15573`	$u15573
character(0)	`character(0)`	`character(0)`	`character(0)`	character(0)
$u21505	`$u21505`	`[[5]]`	`$u21505`	$u21505
[1] "j89" "j72" "j93" "j76" "j88"	`[1] "j78" "j80" "j73" "j77" "j92"`	`[1] "j95" "j86" "j93" "j78" "j83"`	`[1] "j80" "j73" "j100" "j72" "j78"`	[1] "j100" "j81" "j83" "j92" "j96"
$u15994	`$u15994`	`[[6]]`	`$u15994`	$u15994
character(0)	`character(0)`	`character(0)`	`character(0)`	character(0)

Table 4: Results comparison between different recommendation algorithms

One thing clear from the table is that for users `15573` and `15221`, none of the five methods generate a recommendation. Hence, it is important to look at methods to evaluate the recommendation results. To validate the accuracy of the model let's implement accuracy measures and compare the accuracy of all the models.

For the evaluation of the model results, the dataset is divided into 90 percent for training and 10 percent for testing the algorithm. The definition of good rating is updated as `5`:

```
> e <- evaluationScheme(Jester5k, method="split",
+ train=0.9,given=15, goodRating=5)
> e
Evaluation scheme with 15 items given
Method: 'split' with 1 run(s).
Training set proportion: 0.900
Good ratings: >=5.000000
Data set: 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106
ratings.
```

The following script is used to build the collaborative filtering model, apply it on a new dataset for predicting the ratings and then the prediction accuracy is computed the error matrix is shown as follows:

```
> #User based collaborative filtering
> r1 <- Recommender(getData(e, "train"), "UBCF")
> #Item based collaborative filtering
> r2 <- Recommender(getData(e, "train"), "IBCF")
> #PCA based collaborative filtering
> #r3 <- Recommender(getData(e, "train"), "PCA")
> #POPULAR based collaborative filtering
> r4 <- Recommender(getData(e, "train"), "POPULAR")
> #RANDOM based collaborative filtering
> r5 <- Recommender(getData(e, "train"), "RANDOM")
> #SVD based collaborative filtering
> r6 <- Recommender(getData(e, "train"), "SVD")
> #Predicted Ratings
> p1 <- predict(r1, getData(e, "known"), type="ratings")
> p2 <- predict(r2, getData(e, "known"), type="ratings")
> #p3 <- predict(r3, getData(e, "known"), type="ratings")
> p4 <- predict(r4, getData(e, "known"), type="ratings")
> p5 <- predict(r5, getData(e, "known"), type="ratings")
> p6 <- predict(r6, getData(e, "known"), type="ratings")
> #calculate the error between the prediction and
> #the unknown part of the test data
> error <- rbind(
+ calcPredictionAccuracy(p1, getData(e, "unknown")),
+ calcPredictionAccuracy(p2, getData(e, "unknown")),
+ #calcPredictionAccuracy(p3, getData(e, "unknown")),
+ calcPredictionAccuracy(p4, getData(e, "unknown")),
+ calcPredictionAccuracy(p5, getData(e, "unknown")),
+ calcPredictionAccuracy(p6, getData(e, "unknown"))
+ )
> rownames(error) <- c("UBCF","IBCF","POPULAR","RANDOM","SVD")
> error
RMSE MSE MAE
UBCF 4.485571 20.12034 3.511709
IBCF 4.606355 21.21851 3.466738
POPULAR 4.509973 20.33985 3.548478
RANDOM 7.917373 62.68480 6.464369
SVD 4.653111 21.65144 3.679550
```

From the preceding result, `UBCF` has the lowest error in comparison to the other recommendation methods. Here, to evaluate the results of the predictive model, we are using k-fold cross validation method; `k` is assumed to be taken as `4`:

```
> #Evaluation of a top-N recommender algorithm
```

```
> scheme <- evaluationScheme(Jester5k, method="cross", k=4,
+ given=3,goodRating=5)
> scheme
Evaluation scheme with 3 items given
Method: 'cross-validation' with 4 run(s).
Good ratings: >=5.000000
Data set: 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106
ratings.
```

The results of the models from the evaluation scheme show the runtime versus the prediction time by different cross validation results for different models, the result is shown as follows:

```
> results <- evaluate(scheme, method="POPULAR", n=c(1,3,5,10,15,20))
POPULAR run fold/sample [model time/prediction time]
1 [0.14sec/2.27sec]
2 [0.16sec/2.2sec]
3 [0.14sec/2.24sec]
4 [0.14sec/2.23sec]
> results <- evaluate(scheme, method="IBCF", n=c(1,3,5,10,15,20))
IBCF run fold/sample [model time/prediction time]
1 [0.4sec/0.38sec]
2 [0.41sec/0.37sec]
3 [0.42sec/0.38sec]
4 [0.43sec/0.37sec]
> results <- evaluate(scheme, method="UBCF", n=c(1,3,5,10,15,20))
UBCF run fold/sample [model time/prediction time]
1 [0.13sec/6.31sec]
2 [0.14sec/6.47sec]
3 [0.15sec/6.21sec]
4 [0.13sec/6.18sec]
> results <- evaluate(scheme, method="RANDOM", n=c(1,3,5,10,15,20))
RANDOM run fold/sample [model time/prediction time]
1 [0sec/0.27sec]
2 [0sec/0.26sec]
3 [0sec/0.27sec]
4 [0sec/0.26sec]
> results <- evaluate(scheme, method="SVD", n=c(1,3,5,10,15,20))
SVD run fold/sample [model time/prediction time]
1 [0.36sec/0.36sec]
2 [0.35sec/0.36sec]
3 [0.33sec/0.36sec]
4 [0.36sec/0.36sec]
```

The confusion matrix displays the level of accuracy provided by each of the models, we can estimate the accuracy measures such as precision, recall and TPR, FPR, and so on the result is shown as follows:

```
> getConfusionMatrix(results)[[1]]
TP FP FN TN precision recall TPR FPR
1 0.2736 0.7264 17.2968 78.7032 0.2736000 0.01656597 0.01656597 0.008934588
3 0.8144 2.1856 16.7560 77.2440 0.2714667 0.05212659 0.05212659 0.027200530
5 1.3120 3.6880 16.2584 75.7416 0.2624000 0.08516269 0.08516269 0.046201487
10 2.6056 7.3944 14.9648 72.0352 0.2605600 0.16691259 0.16691259
0.092274243
15 3.7768 11.2232 13.7936 68.2064 0.2517867 0.24036802 0.24036802
0.139945095
20 4.8136 15.1864 12.7568 64.2432 0.2406800 0.30082509 0.30082509
0.189489883
```

Association rules as a method for recommendation engine for building product recommendation in a retail/e-commerce scenario, is used in `Chapter 4`, *Regression with Automobile Data*.

Summary

In this chapter, we discussed various methods of recommending products. We looked at different ways of recommending products to users, based on similarity in their purchase pattern, content, item to item comparison, and so on. As far as the accuracy is concerned, always the user-based collaborative filtering is giving better result in a real rating-based matrix as an input. Similarly, the choice of methods for a specific use case is really difficult, so it is recommended to apply all six different methods and the best one should be selected automatically and the recommendation should also get updated automatically.

8
Dimensionality Reduction

In this chapter, we are going to discuss about various methods to reduce data dimensions in performing analysis. In data mining, traditionally people used to apply **principal component analysis** (**PCA**) as a method to reduce the dimensionality in data. Though now in the age of big data, PCA is still valid, however along with that, many other techniques are being used to reduce dimensions. With the growth of data in volumes and variety, the dimension of data has been continuously on the rise. Dimensionality reduction techniques have many applications in different industries, such as in image processing, speech recognition, recommendation engines, text processing, and so on. The main problem in these application areas is not only high dimensional data but also high sparsity. Sparsity means that many columns in the dataset will have missing or blank values.

In this chapter, we will implement dimensionality reduction techniques such as PCA, **singular value decomposition** (**SVD**), and iterative feature selection method using a practical data set and R programming language.

In this chapter, we are going to discuss:

- Why dimensionality reduction is a business problem and what impact it may have on the predictive models
- What are the different techniques, with their respective positives and negatives, and data requirements and more
- Which technique to apply and in what situations, with little bit of mathematics behind the calculation
- An R programming based implementation and interpretation of results in a project

Why dimensionality reduction?

In various statistical analysis models, especially showing a cause and impact relationship between dependent variable and a set of independent variables, if the number of independent variables increases beyond a manageable stage (for example, 100 plus), then it is quite difficult to interpret each and every variable. For example, in weather forecast, nowadays low cost sensors are being deployed at various places and those sensors provide signals and data are being stored in the database. When 1000 plus sensors provide data, it is important to understand the pattern or at least which all sensors are meaningful in performing the desired task.

Another example, from a business standpoint, is that if more than 30 features are impacting a dependent variable (for example, `Sales`), as a business owner I cannot regulate all 30 factors and cannot form strategies for 30 dimensions. Of course, as a business owner I would be interested in looking at 3-4 dimensions that should explain 80% of the dependent variable in the data. From the preceding two examples, it is pretty clear that dimensionality is still a valid business problem. Apart from business and volume, there are other reasons such as computational cost, storage cost of data, and more, if a set of dimensions are not at all meaningful for the target variable or target function why would I store it in my database.

Dimensionality reduction is useful in big data mining applications in performing both supervised learning and unsupervised learning-based tasks, to:

- Identify the pattern how the variables work together
- Display the relationship in a low dimensional space
- Compress the data for further analysis so that redundant features can be removed
- Avoid overfitting as reduced feature set with reduced degrees of freedom does that
- Running algorithms on a reduced feature set would be much more faster than the base features

Two important aspects of data dimensionality are, first, variables that are highly correlated indicate high redundancy in the data, and the second, most important dimensions always have high variance. While working on dimension reduction, it is important to take note of these aspects and make necessary amendments to the process.

Hence, it is important to reduce the dimensions. Also, it is not a good idea to focus on many variables to control or regulate the target variable in a predictive model. In a multivariate study, the correlation between various independent variables affect the empirical likelihood function and affects the eigen values and eigen vectors through covariance matrix.

Techniques available for dimensionality reduction

The dimensionality reduction techniques can be classified into two major groups: parametric or model-based feature reduction and non-parametric feature reduction. In non-parametric feature reduction technique, the data dimension is reduced first and then the resulting data can be used to create any classification or predictive model, supervised or unsupervised. However, the parametric-based method emphasizes on monitoring the overall performance of a model and its accuracy by changing the features and hence deciding how many features are needed to represent the model. The following are the techniques generally used in data mining literature to reduce data dimensions:

- Non-parametric:
    - PCA method
    - Parametric method
    - Forward feature selection
    - Backward feature selection

We are going to discuss these methods in detail with the help of a dataset using R programming language. Apart from these techniques mentioned earlier, there are few more methods but not that popular, such as removing variables with low variance as they don't add much information to the target variable and also removing variables with many missing values. The latter comes under the missing value treatment method, but can still be used to reduce the features in a high dimensional dataset. There are two methods in R, `prcomp()` and `princomp()`, but they use two slightly different methods; `princomp()` uses eigen vectors and `prcomp()` uses SVD method. Some researchers favor `prcomp()` as a method over `princomp()` method. We are going to use two different methods here.

Which technique to apply where?

The usage of the technique actually depends upon the researcher; what he is looking at from the data. If you are looking at hidden features and want to represent the data in a low dimensional space, PCA is the method one should choose. If you are looking at building a good classification or prediction model, then it is not a good idea to perform PCA first; you should ideally include all the features and remove the redundant ones by any parametric method, such as forward or backward. Selection of techniques vary based on data availability, the problem statement, and the task that someone is planning to perform.

Principal component analysis

PCA is a multivariate statistical data analysis technique applicable for datasets with numeric variables only, which computes linear combinations of the original variables and those principal components are orthogonal to each other in the feature space. PCA is a method that uses eigen values and eigen vectors to compute the principal components, which is a linear combination of original variables. PCA can be used in regression modelling in order to remove the problem of multicollinearity in the dataset and can also be used in clustering exercise in order to understand unique group behaviors or segments existing in the dataset.

PCA assumes that the variables are linearly associated to form principal components and variables with high variance do not necessarily represent the best feature. The principal component analysis is based on a few theorems:

- The inverse of an orthogonal matrix is its transpose
- The original matrix times the transposed version of the original matrix are both symmetric
- A matrix is symmetric if it is an orthogonally diagonalizable matrix
- It uses variance matrix, not the correlation matrix to compute components

Following are the steps to perform principal component analysis:

1. Get a numerical dataset. If you have any categorical variable, remove it from the dataset so that mathematical computation can happen on the remaining variables.
2. In order to make the PCA work properly, data normalization should be done. This is done by deducting the mean of the column from each of the values for all the variables; ensure that the mean of the variables of the transformed data should be equal to 0.

   We applied the following normalization formula:

$$\boxed{norm(x) = x_i - \overline{x}}$$

3. Calculate the covariance matrix on the reduced dataset. Covariance is a measure of two variables. A covariance matrix is a display of all the possible covariance values between all the different dimensions:

$$\boxed{\operatorname{cov}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}}$$

The sign of covariance is more important than its value; a positive covariance value indicates that both dimensions increase together and vice versa. Similarly, a negative value of covariance indicates that with an increase in one dimension, the other dimension decreases. If the covariance value is zero, it shows that the two dimensions are independent of each other and there is no relationship.

4. Calculate the eigen vectors and eigen values from the covariance matrix. All eigen vectors always come from a square matrix but all square matrix do not always generate an eigen vector. All eigen vectors of a matrix are orthogonal to each other.
5. Create a feature vector by taking eigen values in order from largest to smallest. The eigen vector with highest eigen value is the principal component of the dataset. From the covariance matrix, the eigen vectors are identified and the eigen values are ordered from largest to smallest:

```
Feature Vector = (Eigen1, Eigen2......Eigen 14)
```

6. Create low dimension data using the transposed version of the feature vector and transposed version of the normalized data. In order to get the transposed dataset, we are going to multiply the transposed feature vector and transposed normalized data.

Let's apply these steps in the following section.

Practical project around dimensionality reduction

We are going to apply dimensionality reduction procedure, both model-based approach and principal component-based approach, on the dataset to come up with less number of features so that we can use those features for classification of the customers into defaulters and no-defaulters.

For practical project, we have considered a dataset default of credit card `clients.csv`, which contains 30,000 samples and 24 attributes or dimensions. We are going to apply two different methods of feature reduction: the traditional way and the modern machine learning way.

Attribute description

The following are descriptions for the attributes from the dataset:

- `X1`: Amount of the given credit (NT dollar). It includes both the individual consumer credit and his/her family (supplementary) credit.
- `X2`: Gender (`1 = male; 2 = female`).
- `X3`: Education (`1 = graduate school; 2 = university; 3 = high school; 4 = others`).
- `X4`: Marital status (`1 = married; 2 = single; 3 = others`).
- `X5`: Age (`year`).
- `X6 – X11`: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows:
    - `X6= the repayment status in September, 2005`
    - `X7 = the repayment status in August, 2005; . . .`
    - `X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . . 8 = payment delay for eight months; 9 = payment delay for nine months, and so on.`
- `X12-X17`: Amount of bill statement (NT dollar).
    - `X12 = amount of bill statement in September, 2005;`
    - `X13 = amount of bill statement in August, 2005; . . .`
    - `X17 = amount of bill statement in April, 2005.`
- `X18-X23`: Amount of previous payment (NT dollar).
    - `X18 = amount paid in September, 2005;`
    - `X19 = amount paid in August, 2005; . . .`
    - `X23 = amount paid in April, 2005.`

Let's get the dataset and do the necessary normalization and transformation:

```
> setwd("select the working directory")
> default<-read.csv("default.csv")
corrplot::corrplot(cor(df),method="ellipse")
```

From the preceding correlations matrix, it is clear that there are some strong correlations between different variables from `X12` to `X17`, which possibly can be clubbed together as linear combinations using PCA. The following graph shows the correlations between different variables:

In the previous correlations graph, the strength of the correlation is reflected by the size of the ellipse and the color varying from red to blue. The blue ellipses indicate positive correlation and the white ones show no correlation. Variable `X12` has a high degree of positive correlation with variables `X13` to `X17`, with a correlation value of more than 80%.

Row number 1 contains the variable description and there are some columns which are categorical; we need to remove those from our dataset. Some other variables are shown as categorical and hence need to be converted back to numeric mode:

```
> df<-default[-1,-c(1,3:5,7:12,25)]
> func1<-function(x){
+ as.numeric(x)
+ }
> df<-as.data.frame(apply(df,2,func1))
> normalize-function(x){
```

```
+ (x-mean(x))
+ }
```

Using `func1`, we convert the non-numeric variables into numeric so that we can apply the normalization function (normalize), which is doing data normalization. After applying normalization, the dataset will look like the one as shown next. To apply principal component analysis, either we can use mean normalized dataset as an input or we can use the base dataset with correlations or a covariance matrix as an input. If we are not going to normalize the dataset, the variable with the highest variance will become the first principal component and hence will dominate other relevant principal components:

```
> dn<-as.data.frame(apply(df,2,normalize))
> str(dn)
'data.frame': 30000 obs. of 14 variables:
$ X1 : num -147484 -47484 -77484 -117484 -117484 ...
$ X5 : num -11.49 -9.49 -1.49 1.51 21.51 ...
$ X12: num -47310 -48541 -21984 -4233 -42606 ...
$ X13: num -46077 -47454 -35152 -946 -43509 ...
$ X14: num -46324 -44331 -33454 2278 -11178 ...
$ X15: num -43263 -39991 -28932 -14949 -22323 ...
$ X16: num -40311 -36856 -25363 -11352 -21165 ...
$ X17: num -38872 -35611 -23323 -9325 -19741 ...
$ X18: num -5664 -5664 -4146 -3664 -3664 ...
$ X19: num -5232 -4921 -4421 -3902 30760 ...
$ X20: num -5226 -4226 -4226 -4026 4774 ...
$ X21: num -4826 -3826 -3826 -3726 4174 ...
$ X22: num -4799 -4799 -3799 -3730 -4110 ...
$ X23: num -5216 -3216 -216 -4216 -4537 ...
```

Running principal component analysis:

```
> options(digits = 2)
> pca1<-princomp(df,scores = T, cor = T)
> summary(pca1)
Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11
Standard deviation 2.43 1.31 1.022 0.962 0.940 0.934 0.883 0.852 0.841
0.514 0.2665
Proportion of Variance 0.42 0.12 0.075 0.066 0.063 0.062 0.056 0.052 0.051
0.019 0.0051
Cumulative Proportion 0.42 0.55 0.620 0.686 0.749 0.812 0.867 0.919 0.970
0.989 0.9936
Comp.12 Comp.13 Comp.14
Standard deviation 0.2026 0.1592 0.1525
Proportion of Variance 0.0029 0.0018 0.0017
Cumulative Proportion 0.9965 0.9983 1.0000
```

Principal component (**PC**) 1 explains 42% variation in the dataset and principal component 2 explains 12% variation in the dataset. This means that first PC has closeness to 42% of the total data points in the n-dimensional space. From the preceding results, it is clear that the first 8 principal components capture 91.9% variation in the dataset. Rest 8% variation in the dataset is explained by 6 other principal components. Now the question is how many principal components should be chosen. The general rule of thumb is 80-20, if 80% variation in the data can be explained by 20% of the principal components. There are 14 variables; we have to look at how many components explain 80% variation in the dataset cumulatively. Hence, it is recommended, based on the 80:20 Pareto principle, taking 6 principal components which explain 81% variation cumulatively.

In a multivariate dataset, the correlation between the component and the original variables is called the component loading. The loadings of the principal components are as follows:

```
> #Loadings of Principal Components
> pca1$loadings
Loadings:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
X1 -0.165 0.301 0.379 0.200 0.111 0.822
X5 0.870 -0.338 -0.331
X12 -0.372 -0.191 0.567 0.416 0.433 0.184 -0.316
X13 -0.383 -0.175 0.136 0.387 -0.345 -0.330 0.645
X14 -0.388 -0.127 -0.114 -0.121 0.123 -0.485 -0.496 -0.528
X15 -0.392 -0.120 0.126 -0.205 -0.523 0.490 0.362 0.346
X16 -0.388 -0.106 0.107 -0.420 0.250 -0.718 -0.227
X17 -0.381 0.165 -0.489 0.513 -0.339 0.428
X18 -0.135 0.383 -0.173 -0.362 -0.226 -0.201 0.749
X19 -0.117 0.408 -0.201 -0.346 -0.150 0.407 -0.280 -0.578 0.110 0.147 0.125
X20 -0.128 0.392 -0.122 -0.245 0.239 -0.108 0.785 -0.153 0.145 -0.125
X21 -0.117 0.349 0.579 -0.499 -0.462 0.124 -0.116
X22 -0.114 0.304 0.609 0.193 0.604 0.164 -0.253
X23 -0.106 0.323 0.367 -0.658 -0.411 -0.181 -0.316
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
SS loadings 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
1.000 1.000 1.000 1.000
Proportion Var 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071 0.071
0.071 0.071 0.071 0.071
Cumulative Var 0.071 0.143 0.214 0.286 0.357 0.429 0.500 0.571 0.643 0.714
0.786 0.857 0.929 1.000
> pca1
Call:
princomp(x = df, cor = T, scores = T)
Standard deviations:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
```

```
2.43 1.31 1.02 0.96 0.94 0.93 0.88 0.85 0.84 0.51
Comp.11 Comp.12 Comp.13 Comp.14
0.27 0.20 0.16 0.15
14 variables and 30000 observations.
```

The following graph indicates the percentage variance explained by the principal components:

Following scree plot shows how many principal components we should retain for the dataset:

From the preceding scree plot, it is concluded that the first principal component has the highest variance, then second and third respectively. From the graph, three principal components have higher variance than the other principal components.

Following biplot indicates the existence of principal components in a n-dimensional space:

The diagonal elements of the covariance matrix are the variances of the variables; the off-diagonal variables are covariance between different variables:

```
> diag(cov(df))
X1 X5 X12 X13 X14 X15 X16 X17 X18 X19 X20
1.7e+10 8.5e+01 5.4e+09 5.1e+09 4.8e+09 4.1e+09 3.7e+09 3.5e+09 2.7e+08
5.3e+08 3.1e+08
X21 X22 X23
2.5e+08 2.3e+08 3.2e+08
```

The interpretation of scores is little bit tricky because they do not have any meaning until and unless you use them to plot on a straight line as defined by the eigen vector. PC scores are the co-ordinates of each point with respect to the principal axis. Using eigen values you can extract eigen vectors which describe a straight line to explain the PCs. The PCA is a form of multidimensional scaling as it represents data in a lower dimension without losing much information about the variables. The component scores are basically the linear combination of loading times the mean centred scaled dataset.

When we deal with multivariate data, it is really difficult to visualize and build models around it. In order to reduce the features, PCA is used; it reduces the dimensions so that we can plot, visualize, and predict the future, in a lower dimension. The principal components are orthogonal to each other, which means that they are uncorrelated. At the data pre-processing stage, the scaling function decides what type of input data matrix is required. If the mean centring approach is used as transformation then covariance matrix should be used as input data for PCA. If a scaling function does a standard z-score transformation, assuming standard deviation is equal to `1`, then correlation matrix should be used as input data.

Now the question is where to apply which transformation. If the variables are highly skewed then z-score transformation and PCA based on correlation should be used. If the input data is approximately symmetric then mean centring approach and PCA based on covariance should be applied.

Now, let's look at the principal component scores:

```
> #scores of the components
> pca1$scores[1:10,]
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
[1,] 1.96 -0.54 -1.330 0.1758 -0.0175 0.0029 0.013 -0.057 -0.22 0.020
0.0169 0.0032 -0.0082 0.00985
[2,] 1.74 -0.22 -0.864 0.2806 -0.0486 -0.1177 0.099 -0.075 0.29 -0.073
-0.0055 -0.0122 0.0040 0.00072
[3,] 1.22 -0.28 -0.213 0.0082 -0.1269 -0.0627 -0.014 -0.084 -0.28 -0.016
0.1125 0.0805 0.0413 -0.05711
[4,] 0.54 -0.67 -0.097 -0.2924 -0.0097 0.1086 -0.134 -0.063 -0.60 0.144
0.0017 -0.1381 -0.0183 -0.05794
[5,] 0.85 0.74 1.392 -1.6589 0.3178 0.5846 -0.543 -1.113 -1.24 -0.038
-0.0195 -0.0560 0.0370 -0.01208
[6,] 0.54 -0.71 -0.081 -0.2880 -0.0924 0.1145 -0.194 -0.015 -0.58 0.505
0.0276 -0.1848 -0.0114 -0.14273
[7,] -15.88 -0.96 -1.371 -1.1334 0.3062 0.0641 0.514 0.771 0.99 -2.890
-0.4219 0.4631 0.3738 0.32748
[8,] 1.80 -0.29 -1.182 0.4394 -0.0620 -0.0440 0.076 -0.013 0.26 0.046
0.0484 0.0584 0.0268 -0.04504
[9,] 1.41 -0.21 -0.648 0.1902 -0.0653 -0.0658 0.040 0.122 0.33 -0.028
-0.0818 0.0296 -0.0605 0.00106
[10,] 1.69 -0.18 -0.347 -0.0891 0.5969 -0.2973 -0.453 -0.140 -0.74 -0.137
0.0248 -0.0218 0.0028 0.00423
```

Larger eigen values indicate larger variation in a dimension. The following script shows how to compute eigen values and eigen vectors from the correlation matrix:

```
> eigen(cor(df),TRUE)$values
```

```
[1] 5.919 1.716 1.045 0.925 0.884 0.873 0.780 0.727 0.707 0.264 0.071 0.041
0.025 0.023
head(eigen(cor(df),TRUE)$vectors)
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
[1,] -0.165 0.301 0.379 0.2004 -0.035 -0.078 0.1114 0.0457 0.822 -0.0291
-0.00617 -0.0157 0.00048
[2,] -0.033 0.072 0.870 -0.3385 0.039 0.071 -0.0788 -0.0276 -0.331 -0.0091
0.00012 0.0013 -0.00015
[3,] -0.372 -0.191 0.034 0.0640 -0.041 -0.044 0.0081 -0.0094 -0.010 0.5667
0.41602 0.4331 0.18368
[4,] -0.383 -0.175 0.002 -0.0075 -0.083 -0.029 -0.0323 0.1357 -0.017 0.3868
0.03836 -0.3452 -0.32953
[5,] -0.388 -0.127 -0.035 -0.0606 -0.114 0.099 -0.1213 -0.0929 0.019 0.1228
-0.48469 -0.4957 0.08662
[6,] -0.392 -0.120 -0.034 -0.0748 -0.029 0.014 0.1264 -0.0392 -0.019
-0.2052 -0.52323 0.4896 0.36210
[,14]
[1,] 0.0033
[2,] 0.0011
[3,] -0.3164
[4,] 0.6452
[5,] -0.5277
[6,] 0.3462
```

The standard deviation of the individual principal components and the mean of all the principal components is as follows:

```
> pca1$sdev
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
2.43 1.31 1.02 0.96 0.94 0.93 0.88 0.85 0.84 0.51 0.27 0.20 0.16 0.15
> pca1$center
X1 X5 X12 X13 X14 X15 X16 X17 X18 X19 X20 X21 X22 X23
-9.1e-12 -1.8e-15 -6.8e-13 -3.0e-12 4.2e-12 4.1e-12 -1.4e-12 5.1e-13
7.0e-14 4.4e-13 2.9e-13 2.7e-13 -2.8e-13 -3.9e-13
```

Using the normalized data, and with no correlations table as an input, the following result is obtained. There is no difference in the output as we got the `pca2` model in comparison to `pca1`:

```
> pca2<-princomp(dn)
> summary(pca2)
Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
Comp.11 Comp.12 Comp.13 Comp.14
Standard deviation 1.7e+05 1.2e+05 3.7e+04 2.8e+04 2.1e+04 2.0e+04 1.9e+04
1.7e+04 1.6e+04 1.2e+04 1.0e+04 8.8e+03 8.2e+03 9.1e+00
Proportion of Variance 6.1e-01 3.0e-01 3.1e-02 1.7e-02 9.4e-03 9.0e-03
```

```
7.5e-03 6.4e-03 5.8e-03 3.0e-03 2.4e-03 1.7e-03 1.5e-03 1.8e-09
Cumulative Proportion 6.1e-01 9.1e-01 9.4e-01 9.5e-01 9.6e-01 9.7e-01
9.8e-01 9.9e-01 9.9e-01 9.9e-01 1.0e+00 1.0e+00 1.0e+00 1.0e+00
```

The next screen plot indicates that three principal components constitute most of the variation in the dataset. The *x* axis shows the principal components and the *y* axis shows the standard deviation (*variance = square of the standard deviation*).

The following script shows the plot being created:

```
> result<-round(summary(pca2)[1]$sdev,0)
> #scree plot
> plot(result, main = "Standard Deviation by Principal Components",
+ xlab="Principal Components",ylab="Standard Deviation",type='o')
```

Using the `prcomp()` function, which uses the SVD method to perform dimensionality reduction can be analysed as follows. `prcomp()` method accepts raw dataset as input and it has a built in argument which requires to make the scaling as true:

```
> pca<-prcomp(df,scale. = T)
> summary(pca)
Importance of components:
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10
Standard deviation 2.433 1.310 1.0223 0.9617 0.9400 0.9342 0.8829 0.8524
0.8409 0.5142
Proportion of Variance 0.423 0.123 0.0746 0.0661 0.0631 0.0623 0.0557
```

```
0.0519 0.0505 0.0189
Cumulative Proportion 0.423 0.545 0.6200 0.6861 0.7492 0.8115 0.8672 0.9191
0.9696 0.9885
PC11 PC12 PC13 PC14
Standard deviation 0.26648 0.20263 0.15920 0.15245
Proportion of Variance 0.00507 0.00293 0.00181 0.00166
Cumulative Proportion 0.99360 0.99653 0.99834 1.00000
```

From the result we can see that there is little variation in the two methods. The SVD approach also shows that the first 8 components explain 91.91% variation in the dataset. From the documentation of `prcomp` and `princomp`, there is no difference in the type of PCA, but there is a difference in the method used to calculate PCA. The two methods are spectral decomposition and singular decomposition.

In spectral decomposition, as shown in the `princomp` function, the calculation is done using eigen on the correlation or covariance matrix. Using the `prcomp` method, the calculation is done by the SVD method.

To get the rotation matrix, which is equivalent to the loadings matrix in the `princomp()` method, we can use the following script:

```
> summary(pca)$rotation
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11
X1 0.165 0.301 -0.379 0.2004 -0.035 0.078 -0.1114 -0.04567 0.822 -0.0291
0.00617
X5 0.033 0.072 -0.870 -0.3385 0.039 -0.071 0.0788 0.02765 -0.331 -0.0091
-0.00012
X12 0.372 -0.191 -0.034 0.0640 -0.041 0.044 -0.0081 0.00937 -0.010 0.5667
-0.41602
X13 0.383 -0.175 -0.002 -0.0075 -0.083 0.029 0.0323 -0.13573 -0.017 0.3868
-0.03836
X14 0.388 -0.127 0.035 -0.0606 -0.114 -0.099 0.1213 0.09293 0.019 0.1228
0.48469
X15 0.392 -0.120 0.034 -0.0748 -0.029 -0.014 -0.1264 0.03915 -0.019 -0.2052
0.52323
X16 0.388 -0.106 0.034 -0.0396 0.107 0.099 0.0076 0.04964 -0.024 -0.4200
-0.06824
X17 0.381 -0.094 0.018 0.0703 0.165 -0.070 -0.0079 -0.00015 -0.059 -0.4888
-0.51341
X18 0.135 0.383 0.173 -0.3618 -0.226 -0.040 0.2010 -0.74901 -0.020 -0.0566
-0.04763
X19 0.117 0.408 0.201 -0.3464 -0.150 -0.407 0.2796 0.57842 0.110 0.0508
-0.14725
X20 0.128 0.392 0.122 -0.2450 0.239 0.108 -0.7852 0.06884 -0.153 0.1449
-0.00015
X21 0.117 0.349 0.062 0.0946 0.579 0.499 0.4621 0.07712 -0.099 0.1241
0.11581
```

```
X22 0.114 0.304 -0.060 0.6088 0.193 -0.604 -0.0143 -0.16435 -0.253 0.0601
0.09944
X23 0.106 0.323 -0.050 0.3672 -0.658 0.411 -0.0253 0.18089 -0.316 -0.0992
-0.03495
PC12 PC13 PC14
X1 -0.0157 0.00048 -0.0033
X5 0.0013 -0.00015 -0.0011
X12 0.4331 0.18368 0.3164
X13 -0.3452 -0.32953 -0.6452
X14 -0.4957 0.08662 0.5277
X15 0.4896 0.36210 -0.3462
X16 0.2495 -0.71838 0.2267
X17 -0.3386 0.42770 -0.0723
X18 0.0693 0.04488 0.0846
X19 0.0688 -0.03897 -0.1249
X20 -0.1247 -0.02541 0.0631
X21 -0.0010 0.08073 -0.0423
X22 0.0694 -0.09520 0.0085
X23 -0.0277 0.01719 -0.0083
```

The rotated dataset can be retrieved by using argument `x` from the summary result of the `prcomp()` method:

```
> head(summary(pca)$x)
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12
[1,] -1.96 -0.54 1.330 0.1758 -0.0175 -0.0029 -0.013 0.057 -0.22 0.020
-0.0169 0.0032
[2,] -1.74 -0.22 0.864 0.2806 -0.0486 0.1177 -0.099 0.075 0.29 -0.073
0.0055 -0.0122
[3,] -1.22 -0.28 0.213 0.0082 -0.1269 0.0627 0.014 0.084 -0.28 -0.016
-0.1125 0.0805
[4,] -0.54 -0.67 0.097 -0.2924 -0.0097 -0.1086 0.134 0.063 -0.60 0.144
-0.0017 -0.1381
[5,] -0.85 0.74 -1.392 -1.6588 0.3178 -0.5846 0.543 1.113 -1.24 -0.038
0.0195 -0.0560
[6,] -0.54 -0.71 0.081 -0.2880 -0.0924 -0.1145 0.194 0.015 -0.58 0.505
-0.0276 -0.1848
PC13 PC14
[1,] -0.0082 -0.00985
[2,] 0.0040 -0.00072
[3,] 0.0413 0.05711
[4,] -0.0183 0.05794
[5,] 0.0370 0.01208
[6,] -0.0114 0.14273
> biplot(prcomp(df,scale. = T))
```

The orthogonal red lines indicate the principal components that explain the entire dataset.

Having discussed various methods of variable reduction, what is going to be the output; the output should be a new dataset, in which all the principal components will be uncorrelated with each other. This dataset can be used for any task such as classification, regression, or clustering, among others. From the results of the principal component analysis how we get the new dataset; let's have a look at the following script:

```
> #calculating Eigen vectors
> eig<-eigen(cor(df))
> #Compute the new dataset
> eigvec<-t(eig$vectors) #transpose the eigen vectors
> df_scaled<-t(dn) #transpose the adjusted data
> df_new<-eigvec %*% df_scaled
> df_new<-t(df_new)
> colnames(df_new)<-c("PC1","PC2","PC3","PC4",
+ "PC5","PC6","PC7","PC8",
+ "PC9","PC10","PC11","PC12",
+ "PC13","PC14")
> head(df_new)
PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12
[1,] 128773 -19470 -49984 -27479 7224 10905 -14647 -4881 -113084 -2291 1655
2337
[2,] 108081 11293 -12670 -7308 3959 2002 -3016 -1662 -32002 -9819 -295 920
[3,] 80050 -7979 -24607 -11803 2385 3411 -6805 -3154 -59324 -1202 7596 7501
[4,] 37080 -39164 -41935 -23539 2095 9931 -15015 -4291 -92461 12488 -11
-7943
```

```
[5,] 77548 356 -50654 -38425 7801 20782 -19707 -28953 -88730 -12624 -9028
-4467
[6,] 34793 -42350 -40802 -22729 -2772 9948 -17731 -2654 -91543 37090 2685
-10960
PC13 PC14
[1,] -862 501
[2,] -238 -97
[3,] 2522 -4342
[4,] -1499 -4224
[5,] 3331 -9412
[6,] -1047 -10120
```

Parametric approach to dimension reduction

To some extent, we touched upon model-based dimension reduction in `Chapter 4`, *Regression with Automobile Data*, *Logistic regression*, where we tried to implement regression modelling and in that process we tried to reduce the data dimension by applying **Akaike Information Criteria** (**AIC**). **Bayesian Information Criteria** (**BIC**) such as AIC can also be used to reduce data dimensions. As far as the model-based method is concerned, there are two approaches:

- **The forward selection method**: In forward selection method, one variable at a time is added to the model and the model goodness of fit statistics and error are computed. If the addition of a new dimension reduces error and increases the model goodness of fit, then that dimension is retained by the model, else that dimension is removed from the model. This is applicable across different supervised based algorithms, such as random forest, logistic regression, neural network, and support vector machine-based implementations. The process of feature selection continues till all the variables get tested.
- **The backward selection method**: In backward selection method, the model starts with all the variables together. Then one variable is deleted from the model and the model goodness of fit statistics and error (any loss function pre-defined) is computed. If the deletion of a new dimension reduces error and increases the model goodness of fit, then that dimension is dropped from the model, else that dimension is kept by the model.

The problem in this chapter we are discussing is a case of supervised learning classification, where the dependent variable is default or no default. The logistic regression method, as we discussed in `Chapter 4`, *Regression with Automobile Data*, *Logistic regression*, uses a step wise dimension reduction procedure to remove unwanted variables from the model. The same exercise can be done on the dataset we discussed in this chapter.

Apart from the standard methods of data dimension reduction, there are some not so important methods available which can be considered, such as missing value estimation method. In a large data set with many dimension sparsity problem will be a common scenario, before applying any formal process of dimensionality reduction, if we can apply the missing value percentage calculation method on the dataset, we can drop many variables. The threshold to drop the variables failing to meet the minimum missing percentage has to be decided by the analyst.

References

Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2), 2473-2480.

Summary

In this chapter, we discussed various methods of performing dimensionality reduction in a sample dataset. Removing redundant features not only improves model accuracy, but also saves computational effort and time. From business user's point of with less number of dimensions, it is more intuitive to build strategies than to focus on large number of features. We discussed which technique to use where and what the data requirement for each of the methods is. The reduction in dimensions also provides meaningful insights into large datasets. In the next chapter, we are going to learn about neural network methods for classification, regression, and time series forecasting.

9

Applying Neural Network to Healthcare Data

Neural-network-based models are gradually becoming the backbone of artificial intelligence and machine learning implementations. The future of data mining will be governed by the usage of artificial neural-network-based advanced modeling techniques. One obvious question: why is neural network gaining so much importance recently though it was invented in 1950s? Borrowed from the computer science domain, a neural network can be defined as a parallel information processing system where the inputs are connected with each other like neurons in the human brain to transmit information so that activities such as face recognition, image recognition, and so on can be performed. In this chapter, we are going to learn about application of neural-network-based methods in various data mining tasks such as classification, regression, time series forecasting, and feature reduction. **Artificial Neural Network** (**ANN**) functions in a way that is similar to the human brain, where billions of neurons link to each other for information processing and insight generation.

In this chapter, you will learn about various types of neural networks, methods, and variants of neural networks with different functions to control the training of artificial neural networks in performing standard data mining tasks such as:

- Prediction of real valued output using regression-based methods
- Prediction of output levels in a classification-based task
- Forecasting future values of a numerical attribute based on historical data
- Compressing features to recognize important ones in order to perform prediction or classification

Introduction to neural networks

The brain's biological network provides the basis for connecting elements in a real-life scenario for information processing and insight generation. It's a hierarchy of neurons connected through layers, where the output of one layer becomes the input for another layer; information passes from one layer to another layer as weights. The weights associated with each neuron contain insights so that the recognition and reasoning become easier for the next level. Artificial neural network is a very popular and effective method that consists of layers associated with weights. The association between different layers is governed by a mathematical equation that passes information from one layer to the other. In fact, a bunch of mathematical equations are at work inside one artificial neural network model. The following graph shows the general architecture for a neural-network-based model:

Figure 1

In the preceding graph, there are three layers—**Input**, **Hidden** and **Output** layer—which are the core of any neural network-based architecture. ANNs are a powerful technique used to solve many real-world problems such as classification, regression, and feature selection. ANNs have the ability to learn from new experiences in the form of new input data in order to improve the performance of classification- or regression-based tasks and to adapt themselves to changes in the input environment. Each circle in the preceding figure represents a neuron.

There are different variants of neural networks that are used in multiple different scenarios; we are going to explain a few of them conceptually in this chapter, and also their usage in practical applications:

- **Single hidden layer neural network**: This is the simplest form of neural network, as shown in the preceding figure. In it, there is only one hidden layer.
- **Multiple hidden layer neural networks**: In this form, more than one hidden layer will connect the input data to the output data. The complexity of calculation

increases in this form as it requires more computational power in the system to process information.

- **Feed forward neural networks**: In this form of neural network architecture, the information passed is one-directional from one layer to another layer; there is no iteration from the first level of learning.
- **Back propagation neural networks**: In this form of neural network, there are two important steps. Feed forward works by passing information from the input to the hidden and from the hidden to the output layer; secondly, it calculates the error and propagates it back to the previous layers.

The feed-forward neural network model architecture is shown in the following figure, and backpropagation method is explained in *Figure 3*:

Figure 2

In the following figure, the red-colored arrows indicate information that has not passed through the output layer, and is again fed back to the input layer in terms of errors:

Figure 3

Having displayed the general architecture for different types of neural networks, let's visit the underlying math behind them.

Understanding the math behind the neural network

The neurons present in different layers–input, hidden, and output–are interconnected through a mathematical function called activation function, as displayed in *Figure 1*. There are different variants of the activation function, which are explained as follows. Understanding the activation function will help in implementation of the neural network model for better accuracy:

- **Sigmoid function**: This is frequently used by professionals in data mining and analytics, as it is easier to explain and implement too. The equation is mentioned here:

$$f(x) = \frac{1}{1+e^{-\beta x}}$$

  The sigmoid function, also known as the logistic function, is mostly used to transform the input data from the input layer to the mapping layer, or the hidden layer.

- **Linear function**: This is one of the simple functions typically used to transfer information from the de-mapping layer to the output layer. The formula is as follows:

$$f(x)=x$$

- **Gaussian function**: Gaussian functions are bell-shaped curves that are applicable for continuous variables, where the objective is to classify the output into multiple classes:

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

- **Hyperbolic tangent function**: This is another variant of transformation function; it is used to transform information from the mapping layer to the hidden layer:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- **Log sigmoid transfer function**: The following formula explains the log sigmoid transfer function used in mapping the input layer to the hidden layer:

$$f(x) = \log\left(\frac{1}{1 + e^{-\beta x}}\right)$$

- **Radial basis function**: This is another activation function; it is used to transfer information from the de-mapping layer to the output layer:

$$f(x) = \sum_{j-1}^{m} w_j h_j(x)$$

Different types of transfer functions, as previously discussed, can be interchangeable in neural network architectures. They can be used in different stages such as input to hidden, hidden to output, and so on, to improve the model accuracy.

Neural network implementation in R

R programming for statistical computing provides three different libraries to perform the neural network model for various tasks. These three are `nnet`, `neuralnet`, and `rsnns`. In this chapter, we will use `ArtPiece_1.csv` and two libraries, `nnet` and `neuralnet`, to perform various tasks. The syntax for neural networks in those two libraries can be explained as follows.

The `neuralnet` library depends on two other libraries, `grid` and `mass`; while installing the `neuralnet` library, you have to make sure that these two dependency libraries are installed properly. In fitting a neural network model, the desired level of accuracy in a model defines the required number of hidden layers with the number of neurons in it; the number of hidden layers increases as the level of complexity increases. This library provides an option for training of neural networks using backpropagation, resilient backpropagation with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993), or the modified globally convergent version by Anastasiadis et al. (2005). The package allows flexible settings through custom choice of error and activation functions. Now let's look at the syntax and the components in it that dictate the accuracy of the model:

Formula	**To define the input and output relationship**
`Data`	Dataset with the input and output variables.
`Hidden`	A vector specifying the number of hidden layers in it. For example, (10, 5, 2) means 10 hidden neurons in the first layer, 5 in the second, and 2 in the third layer.
`Stepmax`	Maximum number of steps to be used.
`Rep`	Number of iterations.
`Startweights`	Random weights for the connections.
`Algorithm`	The algorithm to be used to fit a neural network `object.'backprop'`- refers to backpropagation. The `'rprop+'` and `'rprop-'` refer to resilient backpropagation with and without weight backtracking respectively, while `'sag'` and `'slr'` refer to the smallest absolute gradient and smallest learning rate.
`Err.fct`	Two methods: **sum squared error** (SSE) for regression-based prediction and **cross entropy** (**CE**) for classification-based problems.
`Act.fct`	`"logistic"` and `"tanh"` are possible for the logistic function and tangent hyperbolicus.
`Linear.output`	If no activation function is applied to the output layer, it has to be TRUE.
`Likelihood`	If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated.

Table 1: neuralnet syntax description

Table 2 shows the syntax of the `nnet` library, which can also be used to create neural-network-based models:

Formula	**A formula defining the input-output relationship**
`X`	A matrix or data frame containing the input variables.
`Y`	A matrix or data frame containing the output variables.
`Weights`	Weights.
`Size`	Number of units in the hidden layer.
`Data`	Dataset.
`Na.action`	If NAs found what should be taken.
`Entropy`	Switch for entropy (is equal to maximum conditional likelihood) fitting. Default by leastsquares.
`Softmax`	Switch for softmax (log-linear model) and maximum conditional likelihood fitting. `Linout`, `Entropy`, `Softmax`, and `Censored` are mutually exclusive.
`Decay`	Parameter for weight decay.
`Maxit`	Maximum number of iterations.
`Trace`	Switch for tracing optimization.

Table 2: nnet syntax description

Having discussed the syntax in *Table 1* for both the libraries, let's have a look at the dataset to be used for prediction and classification-based tasks:

```
> library(neuralnet)
Loading required package: grid
Loading required package: MASS
Warning message:
package 'neuralnet' was built under R version 3.2.3
```

The structure function for the dataset shows the types of variables and the size of the dataset we are going to use for this chapter:

```
> art<- read.csv("ArtPiece_1.csv")
> str(art)
'data.frame': 72983 obs. of 26 variables:
$ Cid : int 1 2 3 4 5 6 7 8 9 10 ...
$ Art.Auction.House : Factor w/ 3 levels "Artnet","Christie",..: 3 3 3 3 3
3 3 3 3 3 ...
$ IsGood.Purchase : int 0 0 0 0 0 0 0 0 0 0 ...
```

```
$ Critic.Ratings : num 8.9 9.36 7.38 6.56 6.94 ...
$ Buyer.No : int 21973 19638 19638 19638 19638 19638 19638 19638 21973
21973 ...
$ Zip.Code : int 33619 33619 33619 33619 33619 33619 33619 33619 33619
33619 ...
$ Art.Purchase.Date : Factor w/ 517 levels "1/10/2012","1/10/2013",..: 386
386 386 386 386 386 386 386 386 386 ...
$ Year.of.art.piece : Factor w/ 10 levels "01-01-1947","01-01-1948",..: 6 4
5 4 5 4 4 5 7 7 ...
$ Acq.Cost : num 49700 53200 34300 28700 28000 39200 29400 31500 39200
53900 ...
$ Art.Category : Factor w/ 33 levels "Abstract Art Type I",..: 1 13 13 13
30 24 31 30 31 30 ...
$ Art.Piece.Size : Factor w/ 864 levels "10in. X 10in.",..: 212 581 68 837
785 384 485 272 485 794 ...
$ Border.of.art.piece : Factor w/ 133 levels " ","Border 1",..: 2 40 48 48
56 64 74 85 74 97 ...
$ Art.Type : Factor w/ 1063 levels "Type 1","Type 10",..: 1 176 287 398 509
620 731 842 731 953 ...
$ Prominent.Color : Factor w/ 17 levels "Beige","Black",..: 14 16 8 15 15
16 2 16 2 14 ...
$ CurrentAuctionAveragePrice: int 52157 52192 28245 12908 22729 32963 20860
25991 44919 64169 ...
$ Brush : Factor w/ 4 levels "","Camel Hair Brush",..: 2 2 2 2 4 2 2 2 2 2
...
$ Brush.Size : Factor w/ 5 levels "0","1","2","3",..: 2 2 3 2 3 3 3 3 3 2
...
$ Brush.Finesse : Factor w/ 4 levels "Coarse","Fine",..: 2 2 1 2 1 1 1 1 1
2 ...
$ Art.Nationality : Factor w/ 5 levels "American","Asian",..: 3 1 1 1 1 3 3
1 3 1 ...
$ Top.3.artists : Factor w/ 5 levels "MF Hussain","NULL",..: 3 1 1 1 4 3 3
4 3 4 ...
$ CollectorsAverageprice : Factor w/ 13193 levels "#VALUE!","0",..: 11433
11808 7802 4776 7034 8536 7707 7355 9836 483 ...
$ GoodArt.check : Factor w/ 3 levels "NO","NULL","YES": 2 2 2 2 2 2 2 2 2 2
...
$ AuctionHouseGuarantee : Factor w/ 3 levels "GREEN","NULL",..: 2 2 2 2 2 2
2 2 2 2 ...
$ Vnst : Factor w/ 37 levels "AL","AR","AZ",..: 6 6 6 6 6 6 6 6 6 6 ...
$ Is.It.Online.Sale : int 0 0 0 0 0 0 0 0 0 0 ...
$ Min.Guarantee.Cost : int 7791 7371 9723 4410 7140 4158 3731 5775 3374
11431 ...
```

Neural networks for prediction

The use of neural networks for prediction requires the dependent/target/output variable to be numeric, and all the input/independent/feature variables can be of any type. From the `ArtPiece` dataset, we are going to predict what is going to be the current auction average price based on all the parameters available. Before applying a neural-network-based model, it is important to preprocess the data, by excluding the missing values and any transformation if required; hence, let's preprocess the data:

```
library(neuralnet)
art<- read.csv("ArtPiece_1.csv")
str(art)
#data conversion for categorical features
art$Art.Auction.House<-as.factor(art$Art.Auction.House)
art$IsGood.Purchase<-as.factor(art$IsGood.Purchase)
art$Art.Category<-as.factor(art$Art.Category)
art$Prominent.Color<-as.factor(art$Prominent.Color)
art$Brush<-as.factor(art$Brush)
art$Brush.Size<-as.factor(art$Brush.Size)
art$Brush.Finesse<-as.factor(art$Brush.Finesse)
art$Art.Nationality<-as.factor(art$Art.Nationality)
art$Top.3.artists<-as.factor(art$Top.3.artists)
art$GoodArt.check<-as.factor(art$GoodArt.check)
art$AuctionHouseGuarantee<-as.factor(art$AuctionHouseGuarantee)
art$Is.It.Online.Sale<-as.factor(art$Is.It.Online.Sale)
#data conversion for numeric features
art$Critic.Ratings<-as.numeric(art$Critic.Ratings)
art$Acq.Cost<-as.numeric(art$Acq.Cost)
art$CurrentAuctionAveragePrice<-as.numeric(art$CurrentAuctionAveragePrice)
art$CollectorsAverageprice<-as.numeric(art$CollectorsAverageprice)
art$Min.Guarantee.Cost<-as.numeric(art$Min.Guarantee.Cost)
#removing NA, Missing values from the data
fun1<-function(x){
ifelse(x=="#VALUE!",NA,x)
}
art<-as.data.frame(apply(art,2,fun1))
art<-na.omit(art)
#keeping only relevant variables for prediction
art<-art[,c("Art.Auction.House","IsGood.Purchase","Art.Category",
"Prominent.Color","Brush","Brush.Size","Brush.Finesse",
"Art.Nationality","Top.3.artists","GoodArt.check",
"AuctionHouseGuarantee","Is.It.Online.Sale","Critic.Ratings",
"Acq.Cost","CurrentAuctionAveragePrice","CollectorsAverageprice",
"Min.Guarantee.Cost")]
#creating dummy variables for the categorical variables
library(dummy)
art_dummy<-
```

```
dummy(art[,c("Art.Auction.House","IsGood.Purchase","Art.Category",
"Prominent.Color","Brush","Brush.Size","Brush.Finesse",
"Art.Nationality","Top.3.artists","GoodArt.check",
"AuctionHouseGuarantee","Is.It.Online.Sale")],int=F)
art_num<-art[,c("Critic.Ratings",
"Acq.Cost","CurrentAuctionAveragePrice","CollectorsAverageprice",
"Min.Guarantee.Cost")]
art<-cbind(art_num,art_dummy)
## 70% of the sample size
smp_size <- floor(0.70 * nrow(art))
## set the seed to make your partition reproductible
set.seed(123)
train_ind <- sample(seq_len(nrow(art)), size = smp_size)
train <- art[train_ind, ]
test <- art[-train_ind, ]
fun2<-function(x){
as.numeric(x)
}
train<-as.data.frame(apply(train,2,fun2))
test<-as.data.frame(apply(test,2,fun2))
```

In the training dataset, there are 50,867 observations and 17 variables, and in the test dataset, there are 21,801 observations and 17 variables. The current auction average price is the dependent variable for prediction, using only four other numeric variables as features:

```
>fit<- neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
> fit
Call: neuralnet(formula = CurrentAuctionAveragePrice ~ Critic.Ratings +
Acq.Cost + CollectorsAverageprice + Min.Guarantee.Cost, data = train,
hidden = 15, err.fct = "sse", linear.output = F)
1 repetition was calculated.
Error Reached Threshold Steps
1 54179625353167 0.004727494957 23
```

A summary of the main results of the model is provided by `result.matrix`. A snapshot of the result.matrix is given as follows:

```
> fit$result.matrix
1
error 54179625353167.000000000000
reached.threshold 0.004727494957
steps 23.000000000000
Intercept.to.1layhid1 -0.100084491816
Critic.Ratings.to.1layhid1 0.686332945444
Acq.Cost.to.1layhid1 0.196864454378
CollectorsAverageprice.to.1layhid1 -0.793174429352
```

```
Min.Guarantee.Cost.to.1layhid1 0.528046199494
Intercept.to.1layhid2 0.973616842194
Critic.Ratings.to.1layhid2 0.839826678316
Acq.Cost.to.1layhid2 0.077798897157
CollectorsAverageprice.to.1layhid2 0.988149246218
Min.Guarantee.Cost.to.1layhid2 -0.385031389636
Intercept.to.1layhid3 -0.008367359937
Critic.Ratings.to.1layhid3 -1.409715725621
Acq.Cost.to.1layhid3 -0.384200569485
CollectorsAverageprice.to.1layhid3 -1.019243809714
Min.Guarantee.Cost.to.1layhid3 0.699876747202
Intercept.to.1layhid4 2.085203047278
Critic.Ratings.to.1layhid4 0.406934874266
Acq.Cost.to.1layhid4 1.121189503896
CollectorsAverageprice.to.1layhid4 1.405748076570
Min.Guarantee.Cost.to.1layhid4 -1.043884892202
Intercept.to.1layhid5 0.862634752109
Critic.Ratings.to.1layhid5 0.814364667751
Acq.Cost.to.1layhid5 0.502879862694
```

If the `error` function is equal to the negative log likelihood function, the error refers to the likelihood as it is used to calculate the Akaike Information Criterion (AIC). We can store the covariate and response data in a matrix:

```
> output<-cbind(fit$covariate,fit$result.matrix[[1]])
> head(output)
[,1] [,2] [,3] [,4] [,5]
[1,] 14953 49000 10727 5775 54179625353167
[2,] 35735 38850 9494 12418 54179625353167
[3,] 34751 43750 8738 9611 54179625353167
[4,] 31599 41615 5955 4158 54179625353167
[5,] 10437 34755 8390 4697 54179625353167
[6,] 13177 54670 13024 11921 54179625353167
```

To compare the results of a neural network model, we can use different tuning factors such as changing the algorithm, hidden layer, and learning rate. As an example, only four numeric features were used to generate the prediction; we could have used all the 91 features for prediction of the current auction average price variable. We can also use a different algorithm from the `nnet` library, as follows:

```
> fit<-nnet(CurrentAuctionAveragePrice~Critic.Ratings+Acq.Cost+
+ CollectorsAverageprice+Min.Guarantee.Cost,data=train,
+ size=100)
# weights: 601
initial value 108359809492660.125000
final value 108359250706334.000000
converged
```

```
> fit
a 4-100-1 network with 601 weights
inputs: Critic.Ratings Acq.Cost CollectorsAverageprice Min.Guarantee.Cost
output(s): CurrentAuctionAveragePrice
options were -
```

Both the libraries provide equal results; there is no difference in the model result, but to tune the results further, it is important to look at the model tuning parameters such as learning rate, hidden neurons, and so on. The following graph shows the neural network architecture:

The model for predicting the unseen data points can be implemented using the `compute` function available in the `neuralnet` library, and the `predict` function available in the `nnet` library.

Neural networks for classification

For classification-based projects, the dependent variable can be binary or can have multiple levels, such as credit card fraud detection, classification of customers into different clusters as far as the marketing is concerned, and so on. In the current scenario from the `ArtPiece` dataset, we are trying to predict whether a work of art is a good purchase, or not, by taking a few business-relevant variables. For demo purposes, we have considered only a few features, but other features present in the dataset can be used to generate a better result:

```
> fit<-neuralnet(IsGood.Purchase_1~Brush.Size_1+Brush.Size_2+Brush.Size_3+
+ Brush.Finesse_Coarse+Brush.Finesse_Fine+
+ Art.Nationality_American+Art.Nationality_Asian+
+ Art.Nationality_European+GoodArt.check_YES,data=train[1:2000,],
+ hidden = 25,err.fct = "ce",linear.output = F)
> fit
Call: neuralnet(formula = IsGood.Purchase_1 ~ Brush.Size_1 + Brush.Size_2 +
Brush.Size_3 + Brush.Finesse_Coarse + Brush.Finesse_Fine +
Art.Nationality_American + Art.Nationality_Asian + Art.Nationality_European
+ GoodArt.check_YES, data = train[1:2000, ], hidden = 25, err.fct = "ce",
linear.output = F)
1 repetition was calculated.
Error Reached Threshold Steps
1 666.1522488 0.009864324362 8254
> output<-cbind(fit$covariate,fit$result.matrix[[1]])
> head(output)
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 1 0 0 0 1 0 0 1 0 666.1522488
[2,] 1 0 0 0 1 1 0 0 0 666.1522488
[3,] 1 0 0 0 1 0 0 1 0 666.1522488
[4,] 0 1 0 1 0 0 0 1 0 666.1522488
[5,] 0 1 0 1 0 1 0 0 0 666.1522488
[6,] 1 0 0 0 1 1 0 0 0 666.1522488
```

The following graph shows the neural network model for classification:

Using another library, `nnet`, for classification-based problems, the following result is derived:

```
> fit.nnet<-
nnet(factor(IsGood.Purchase_1)~Brush.Size_1+Brush.Size_2+Brush.Size_3+
+ Brush.Finesse_Coarse+Brush.Finesse_Fine+
+ Art.Nationality_American+Art.Nationality_Asian+
+ Art.Nationality_European+GoodArt.check_YES,data=train[1:2000,],
+ size=9)
# weights: 100
initial value 872.587818
iter 10 value 684.034783
iter 20 value 667.751170
iter 30 value 667.027963
iter 40 value 666.337669
```

```
iter 50 value 666.156889
iter 60 value 666.138741
iter 70 value 666.137048
iter 80 value 666.136505
final value 666.136439
converged
> fit.nnet
a 9-9-1 network with 100 weights
inputs: Brush.Size_1 Brush.Size_2 Brush.Size_3 Brush.Finesse_Coarse
Brush.Finesse_Fine Art.Nationality_American Art.Nationality_Asian
Art.Nationality_European GoodArt.check_YES
output(s): factor(IsGood.Purchase_1)
options were - entropy fitting
```

Neural networks for forecasting

Neural networks can also be used to generate forecasts for a time series variable. There is a library called `forecast` in R that deploys feed-forward neural networks with a single hidden layer, and lagged inputs for forecasting univariate time series. For the forecasting example, we have taken an inbuilt dataset available in R called "Air Passengers" to apply the neural network.

The following table reflects the parameter arguments required by the `nnetar` function with the corresponding description of how they are being used in the model:

`X`	**Univariate time series with a time variable**
`p`	Number of non-seasonal lags used as input
`P`	Number of seasonal lags used as input
`Size`	Number of nodes in the hidden layer
`Repeats`	Number of networks to fit with different random settings of weights
`Lambda`	This is known as box-cox transformation parameter
`Xreg`	External regressors used in fitting the model
`Mean`	Point forecasts as mean

The actual time series looks as follows:

```
> fit<-nnetar(AirPassengers, p=9,P=,size = 10, repeats = 50,lambda = 0)>
plot(forecast(fit,10))
```

A neural-network-based forecasting model generates the following results as an output:

```
> summary(fit)
Length Class Mode
x 144 ts numeric
m 1 -none- numeric
p 1 -none- numeric
P 1 -none- numeric
scale 1 -none- numeric
size 1 -none- numeric
lambda 1 -none- numeric
model 50 nnetarmodels list
fitted 144 ts numeric
residuals 144 ts numeric
lags 10 -none- numeric
series 1 -none- character
method 1 -none- character
call 6 -none- call
```

With a forecast of the next 10 periods, the graph looks as follows:

Merits and demerits of neural networks

The neural network method for performing classification, prediction, and forecasting is still recognized as a black box methodology in different industries. People still provide more importance to logistic regression than neural network because of its complexity in explaining the relationship between the dependent and independent variable.

The limitations of the neural network model can be stated as follows:

- In contrast to decision trees and rule extraction techniques, the knowledge (patterns) "discovered" by neural networks is not represented in a form understandable by humans.
- Knowledge in a trained **neural network** (**NN**) is encoded in its connection weights; hence, NN cannot be used for descriptive data mining (exploration).
- If NN are used for decision making, it is impossible to explain their decisions. Often other techniques have to be combined with NN for explanation.

Here are the merits of neural networks:

- Though it is a bit complex to understand and interpret the results, still it is considered a powerful technique for classification and regression
- It is considered a powerful machine learning technique for automatic predictive modeling
- It captures complex relationships in datasets, which a traditional algorithm such as linear regression or logistic regression fails to understand and interpret

References

Lichman, M. (2013). *UCI Machine Learning Repository* (`http://archive.ics.uci.edu/ml`). Irvine, CA: University of California, School of Information and Computer Science

Summary

In this chapter, we discussed various methods of performing classification, regression, and forecasting using the neural network model. Be it supervised or unsupervised data mining problems, neural-network-based implementations are popular not only among users but also among business stakeholders. We discussed which model to use where and the data requirement for each of the models. In this chapter, we specifically highlighted the importance of a powerful technique with better accuracy always for classification- and regression-based scenarios.

Index

A

activation function
  about 224
  gaussian function 224
  hyperbolic tangent function 225
  linear function 224
  log sigmoid transfer function 225
  radial basis function 225
  sigmoid function 224
Akaike Information Criteria (AIC) 127, 219
analysis of variance (ANOVA) 118
apply function
  about 29
  eapply function 31
  lapply function 30
  mapply function 31
  rapply 31
  sapply function 30
  tapply function 30
Apriori algorithm 147, 152
Artificial Neural Network (ANN) 221
ArtPiece_2.csv 113
arules
  about 160
  implementation 159
association rules (arules) 143
association rules
  visualizing 158
assumptions 147

B

bar chart 83
Bayesian Information Criteria (BIC) 219
bivariate analysis 41, 42
bivariate statistics 51
boxplot 84
bubble chart 85
built-in functions 27

C

Cars93_1.csv 113
case study 113
Central Limit Theorem (CLT) 44
clustering methods 164
  advantages 184
  bagged clustering 164
  comparing 184
  disadvantages 184
  fuzzy clustering 164
  hierarchical clustering 164, 173
  K-means clustering 166
  model-based clustering 164, 179
  other cluster algorithms 181
  other clustering algorithms 164
  partitioning clustering 164
clustering
  with e-commerce data 162, 163
confidence 144
contingency tables 50, 51
continous data
  discretizing 50
  interpreting 47, 49
cosine similarity 165
cubic regression 136, 137
customer segmentation, methods
  using clustering-based methods 164
  using frequency model 164
  using monetary (RFM) model 164
  using recency model 164
customer segmentation
  about 163
  methods 164
  performing 164

D

data frames
  indexing 24
  merging 19, 21, 22, 24
  sorting 19, 21, 22, 24
  subsetting 24
data mining 8
  process 9
  relation, with analytics 11
  relation, with data science 11
  relation, with statistical modeling 11
data normality
  checking for 51, 52, 54, 55
data requirement 145
data type conversion 18, 19
data visualization 70
data visualization, using ggplot2
  bar chart 83
  boxplot 84
  bubble chart 85
  coxcomb plot 100
  donut chart 86
  geo mapping 87
  histogram 89
  line chart 90
  pie chart 91
  scatterplot 92
  stacked bar chart 97
  stem and leaf plot 98
  word cloud 99
data visualization, using plotly
  bar charts, using plotly 103
  boxplots, using plotly 104
  bubble plot 102
  polar area chart 108
  polar chart, using plotly 107
  polar scatterplot, using plotly 107
  scatterplot, using plotly 104
data visualization
  about 67, 68, 69
  contribution analysis 69
  data, with locations 68
  part-to-whole 69
  spread of range 69
  spread of values 69
  statistical distribution 69
  testing proportions 68
  textual data representation 69
  unseen patterns 69
  using ggplot2 71, 72
  using plotly 101
  variable hierarchy 68
  variables, comparisons 68
  variables, relationship 68
  viewing proportions 68
date and time formatting 25, 26
dimension reduction, practical project
  parametric approach 219, 220
dimensionality reduction, practical project
  attribute description 207, 208
dimensionality reduction, technique
  applying 204
  principal component analysis (PCA) 205
dimensionality reduction
  about 202, 203
  practical project 206
  technique 204
distribution
  binomial probability distribution 46
distributions
  interpreting 47
donut chart 86

E

Eclat algorithm 156, 158
Euclidean distance 165
exploratory data analysis
  about 34
  bivariate analysis 41, 42
  multivariate analysis 43
  univariate data analysis 35, 36, 38, 40

F

for loop 28
functions
  built-in functions 27
  creating 26
  user-defined functions 27

G

geo mapping
  about 87
  creating 109
ggplot2 library 70
graphics
  URL 70

H

hierarchical clustering
  about 173, 174, 177
  Agglomerative method 173
  Divisive method 173
histogram 89
hypothesis testing
  about 56
  population mean, testing 56
  steps 56
  two sample variance test 62, 63

I

indexing
  data frames 24

K

K-means clustering
  about 166
  cluster membership, predicting for new data 172
  dataset, scaling 168
  demerits 184
  implementing, in live applications 173
  initial cluster seeds, selecting 168
  merits 184
  minimum distance, grouping 170
  number of cluster K, deciding 168
  presence of outliers, checking 166, 167

L

L1 absolute value penalty 138
Least Angle Regression (LARS) 140
lift 144
limitations 147
line chart 90
linear regression 113, 119, 122, 125, 127
  assumptions 116
  equal variance 116
  independence 116
  linearity 116
  multicollinearity 116
  normality 116
logistic regression 129, 133, 136
  backward selection method 133
  both 133
  forward selection method 133
loop
  for loop 28
  repeat loop 28
  while loop 29

M

Manhattan distance 165
Market Basket Analysis (MBA)
  about 142, 143, 145
  applying 145
  assumptions 147
  data requirement 145
  limitations 147
  modeling techniques 147
  practical project 148, 149, 151
  prerequisites 147
  with groceries data 142
merging
  data frames 19
mileage per gallon (MPG) 117
missing value imputation techniques
  about 32
  clustering 33
  keeping that seperate 33
  local average method 33
  mean imputation 32
  model-based 33
model-based clustering 179
model-based method
  backward selection method 219
  forward selection method 219
modeling techniques 147
multivariate analysis 43

N

NA and missing value management 32
neural network (NN) 237
neural network
  about 222
  applying, to healthcare data 221
  back propagation neural networks 223
  demerits 237
  feed forward neural networks 223
  for classification 234
  for forecasting 235
  for prediction 229
  implementation, in R programming 225, 226, 227
  limitations 237
  math 224
  merits 237, 238
  multiple hidden layer neural networks 222
  single hidden layer neural network 222
neural networks
  for classification 233
neuralnet library 226
nnet library 225
non-parametric methods
  about 64
  Kruskal-Wallis test 65
  Mann-Whitney-Wilcoxon test 65
  Wilcoxon signed-rank test 64
normal probability distribution 44

P

penalized regression 137
  Least Absolute Shrinkage Operator (LASSO) 138
  ridge regression 137
pie chart 91
plotly library 70, 101
point-of-sale (POS) 148
poisson probability distribution 46
polar scatterplot 107
population mean
  one tail and two tail test 58, 59, 60
  one tail test, with known variance 56
  testing 56
practical project
  Apriori algorithm 152
  arules, implementation 159, 160
  association rules, visualizing 158
  Eclat algorithm 156, 158
prcomp() 204
Predictive Model Markup Language (PMML) 173
prerequisites 147
principal component (PC) 210
principal component analysis (PCA) 202
princomp() 204
probability distributions
  about 44
  binomial probability distribution 46
  normal probability distribution 44
  poisson probability distribution 46

R

R Commander (Rcmdr) 168
R programming language
  about 11, 12
  arrays 12, 13, 15
  data types 12, 13, 15
  data types, exporting 16, 17
  data types, importing 16, 17
  factors 15
  list management 15
  matrices 12, 13
  sequences 16
  URL 11
  vectors 12, 13
Rcmdr library 168
Receiver Operating Characteristic (ROC) 134
regression issues, formulation 112
regression
  about 112
  case study 113
  linear 112
  non-linear 112
  regression issues, formulation 112
  with automobile data 111
regularized regression methods 141
repeat loop 28
rsnns library 225

S

scatterplot 92
self-organizing map (SOM) 164, 181
  demerits 184
  merits 184
singular value decomposition (SVD) 202
sorting
  data frames 19
stacked bar chart 97
stats library 168
stepwise regression method
  for variable selection 127
string manipulation 31
subsetting
  data frames 25
support 143

T

technique
  for dimensionality reduction 204
transformations 44

U

univariate data analysis 35, 36, 38, 40
user-defined functions 27

V

variable binning 50
variable selection
  stepwise regression method 127
Variance Inflation Factor (VIF) 125

W

while loop 29

Module 3: Social Media Mining with R

Chapter 1: Going Viral 1

Social media mining using sentiment analysis 1
The state of communication 2
What is Big Data? 4
Human sensors and honest signals 6
Quantitative approaches 9
Summary 11

Chapter 2: Getting Started with R 13

Why R? 13
Quick start 16
Vectors, sequences, and combining vectors 19
A quick example – creating data frames and importing files 20
Visualization in R 22
Style and workflow 24
Additional resources 24
Summary 25

Chapter 3: Mining Twitter with R 27

Why Twitter data? 27
Obtaining Twitter data 28
Preliminary analyses 32
Summary 36

Chapter 4: Potentials and Pitfalls of Social Media Data 37

Opinion mining made difficult 37
Sentiment and its measurement 38
The nature of social media data 40
Traditional versus nontraditional social data 40
Measurement and inferential challenges 41
Summary 45

Chapter 5: Social Media Mining – Fundamentals 47
Key concepts of social media mining 47
Good data versus bad data 48
Understanding sentiments 50
Sentiment polarity – data and classification 51
Supervised social media mining – lexicon-based sentiment 53
Supervised social media mining – Naive Bayes classifiers 55
Unsupervised social media mining – Item Response Theory for text scaling 56
Summary 58
Chapter 6: Social Media Mining – Case Studies 59
Introductory considerations 59
Case study 1 – supervised social media mining – lexicon-based sentiment 61
Case study 2 – Naive Bayes classifier 80
Case study 3 – IRT models for unsupervised sentiment scaling 85
Summary 92
Appendix: Conclusions and Next Steps 93
Final thoughts 93
An expanding field 94
Further reading 95
Bibliography 96
Bibliography 99
Index 101

Module 3

Social Media Mining with R

Deploy cutting-edge sentiment analysis techniques to real-world social media data using R

1
Going Viral

In this chapter, we introduce readers to the concept of social media mining. We discuss sentiment analysis, the nature of contemporary online communication, and the facets of Big Data that allow social media mining to be such a powerful tool. Additionally, we discuss some of the potential pitfalls of socially generated data and argue for a quantitative approach to social media mining.

Social media mining using sentiment analysis

People are highly opinionated. We hold opinions about everything from international politics to pizza delivery. Sentiment analysis, synonymously referred to as opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions through written language. Practically speaking, this field allows us to measure, and thus harness, opinions. Up until the last 40 years or so, opinion mining hardly existed. This is because opinions were elicited in surveys rather than in text documents, computers were not powerful enough to store or sort a large amount of information, and algorithms did not exist to extract opinion information from written language.

The explosion of sentiment-laden content on the Internet, the increase in computing power, and advances in data mining techniques have turned social data mining into a thriving academic field and crucial commercial domain. Professor Richard Hamming famously pushes researchers to ask themselves, "What are the important problems in my field?" Researchers in the broad area of **natural language processing** (**NLP**) cannot help but list sentiment analysis as one such pressing problem. Sentiment analysis is not only a prominent and challenging research area, but also a powerful tool currently being employed in almost every business and social domain. This prominence is due, at least in part, to the centrality of opinions as both measures and causes of human behavior.

This book is an introduction to social data mining. For us, social data refers to data generated by people or by their interactions. More specifically, social data for the purposes of this book will usually refer to data in text form produced by people for other people's consumption. Data mining is a set of tools and techniques used to describe and make inferences about data. We approach social data mining with a potent mix of applied statistics and social science theory. As for tools, we utilize and provide an introduction to the statistical programming language R.

The book covers important topics and latest developments in the field of social data mining with many references and resources for continued learning. We hope it will be of interest to an audience with a wide array of substantive interests from fields such as marketing, sociology, politics, and sales. We have striven to make it accessible enough to be useful for beginners while simultaneously directing researchers and practitioners already active in the field towards resources for further learning. Code and additional material will be available online at `http://socialmediaminingr.com` as well as on the authors' GitHub account, `https://github.com/SocialMediaMininginR`.

The state of communication

The state of communication section describes the fundamentally altered modes of social communication fostered by the Internet. The interconnected, social, rapid, and public exchange of information detailed here underlies the power of social data mining. Now more than ever before, information can **go viral**, a phrase first cited as early as 2004.

By changing the manner in which we connect with each other, the Internet changed the way we interact—communication is now bi-directional and many-to-many. Networks are now self-organized, and information travels along every dimension, varying systematically depending on direction and purpose. This new economy with ideas as currency has impacted nearly every person. More than ever, people rely on context and information before making decisions or purchases, and by extension, more and more on peer effects and interactions rather than centralized sources.

The traditional modes of communication are represented mainly by radio and television, which are isotropic and one-to-many. It took 38 years for radio broadcasters and 13 years for television to reach an audience of 50 million, but the Internet did it in just four years (Gallup).

Not only has the nature of communication changed, but also its scale. There were 50 pages on the **World Wide Web** (**WWW**) in 1993. Today, the full impact and scope of the WWW is difficult to measure, but we can get a rough sense of its size: the Indexed Web contains at least 1.7 billion pages as of February 2014 (World Wide Web size). The WWW is the largest, most widely used source of information, with nearly 2.4 billion users (Wikipedia). 70 percent of these users use it daily to both contribute and receive information in order to learn about the world around them and to influence that same world—constantly organizing information around pieces that reflect their desires.

In today's connected world, many of us are members of at least one, if not more, social networking service. The influence and reach of social media enterprises such as Facebook is staggering. Facebook has 1.11 billion monthly active users and 751 million monthly active users of their mobile products (Facebook key facts). Twitter has more than 200 million (Twitter blog) active users. As communication tools, they offer a global reach to huge multinational audiences, delivering messages almost instantaneously.

Connectedness and social media have altered the way we organize our communications. Today we have dramatically more friends and more friends of friends, and we can communicate with these higher order connections faster and more frequently than ever before. It is difficult to ignore the abundance of mimicry (that is, copying or reposting) and repeated social interactions in our social networks. This mimicry is a result of virtual social interactions organized into reaffirming or oppositional feedback loops. We self-organize these interactions via (often preferential) attachments that form organic, shifting networks. There is little question of whether or not social media has already impacted your life and changed the manner in which you communicate. Our beliefs and perceptions of reality, as well as the choices we make, are largely conditioned by our neighbors in virtual and physical networks. When we need to make a decision, we seek out for opinions of others—more and more of those opinions are provided by virtual networks.

Information bounce is the resonance of content within and between social networks often powered by social media such as customer reviews, forums, blogs, microblogs, and other user-generated content. This notion represents a significant change when compared to how information has traveled throughout history; individuals no longer need to exclusively rely on close ties within their physical social networks. Social media has both made our close ties closer and the number of weak ties exponentially greater. Beyond our denser and larger social networks is a general eagerness to incorporate information from other networks with similar interests and desires. The increased access to networks of various types has, in fact, conditioned us to seek even more information; after all, ignoring available information would constitute irrational behavior.

These fundamental changes to the nature and scope of communication are crucial due to the importance of ideas in today's economic and social interactions. Today, and in the future, ideas will be of central importance, especially those ideas that bounce and go viral. Ideas that go viral are those that resonate and spur on social movements, which may have political and social purposes or reshape businesses and allow companies such as Nike and Apple to produce outsized returns on capital. This book introduces readers to the tools necessary to measure ideas and opinions derived from social data at scale. Along the way, we'll describe strategies for dealing with Big Data.

What is Big Data?

People create 2.5 quintillion bytes (2.5 * 1018) of data, or nearly 2.3 million Terabytes of data every day, so much that 90 percent of the data in the world today has been created in the last two years alone. Furthermore, rather than being a large collection of disparate data, much of this data flow consists of data on similar things, generating huge data-sets with billions upon billions of observations. Big Data refers not only to the deluge of data being generated, but also to the astronomical size of data-sets themselves. Both factors create challenges and opportunities for data scientists.

This data comes from everywhere: physical sensors used to gather information, human sensors such as the social web, transaction records, and cell phone GPS signals to name a few. This data is not only big but is growing at an increasing rate. The data used in this book, namely, Twitter data, is no exception. Twitter was launched in March 21, 2006, and it took 3 years, 2 months, and 1 day to reach 1 billion tweets. Twitter users now send 1 billion tweets every 2.5 days.

What proportion of data is Big Data? It turns out that most data-sets are (relatively) small. This may come as a surprise in light of the contemporary excitement surrounding Big Data. The reason for the large number of small data-sets is that data that is not socially generated and publicly displayed is time consuming and expensive to collect. As such, academics, businesses, and other organizations with data needs tend to collect only the minimum amount of information necessary to gain purchase on their questions. These data-sets are usually small and focused and are curated by the organizations that use them; they usually do not plan on updating or adding fresh data to them. The poor management of these data often leads to their misplacement, thereby generating dark data—data that is suspected to exist or ought to exist but is difficult or impossible to find. The problem of dark data is real and prevalent in the myriad of small, locally collected data-sets. The utter lack of central management of data in the tail of the data size distribution invariably causes these sets of data to be forgotten. In spite of the fact that most data is not big, it is primarily the Big Data sets that exhibit exponential growth, propelling the number of bytes created by humans moving upwards daily.

Big Data differs substantially from other data not only in its size and velocity, but also in its scope and density. Big Data is large in scope, that is, it is created by everyone and by itself and thus is informative about a wide audience. This characteristic makes it very useful for studying populations, as the inferences we can make generalize to large groups of people. Compare that with, say, opinions gleaned from a focus group or small survey. These opinions, while highly accurate and easy to obtain, may or may not be reflective of the views of the wider public. Thus, Big Data's scope is a real benefit, at least in terms of generalizing evidence to wide populations.

However, Big Data's density is fairly low. By density, we mean the degree to which Big Data, and especially social data, is directly applicable to questions we want to answer. Again, a comparison to small data is useful. Prior to the explosion of Big Data and the proliferation of tools used to harness it, companies or political campaigns largely used focus groups or surveys to obtain information about public sentiments relevant to their endeavors. The focus groups and surveys furnished organizations with data that was directly applicable to their purpose, and often this data would already be measured with meaningful units. For instance, respondents would describe how much they liked or disliked a new product, or rate a political candidate's TV appearances from 1 to 5. Compare that with social data, where opinion-laden text is buried among terabytes of unrelated information and comes in a form that must be subjected to analysis just to generate a measure of the opinion. Thus, low density of big social data presents unique challenges to organizations trying to utilize opinion data.

The size and scope of Big Data helps us overcome some of the hurdles caused by its low density. For instance, even though each unique piece of social data may have little applicability to our particular task, these small bits of information quickly become useful as we aggregate them across thousands or millions of people. Like the proverbial bundle of sticks—none of which could support inferences alone—when tied together, these small bits of information can be a powerful tool for understanding the opinions of the online populace.

The sheer scope of Big Data has other benefits as well. The size and coverage of many social data-sets creates coverage overlaps in time, space, and topic. This allows analysts to cross-refer socially generated sets against one another or against small-scale data-sets designed to examine niche questions. This type of cross-coverage can generate consilience (Osborne)—the principle that states evidence from independent, unrelated sources can converge to form strong conclusions. That is, when multiple sources of evidence are in agreement, the conclusion can be very strong even when none of the individual sources of evidence are very strong on their own. A crucial characteristic of socially generated data is that it is opinionated. This point underpins the usefulness of big social data for sentiment analysis, and is novel. For the first time in history, interested parties can put their fingers to the pulse of the masses because the masses are frequently opining about what is important to them. They opine with and for each other and anyone else who cares to listen. In sum, opinionated data is the great enabler of opinion-based research.

Human sensors and honest signals

Opinion data generated by humans in real time presents tremendous opportunities. However, big social data will only prove useful to the extent that it is valid. This section tackles the extent to which socially generated data can be used to accurately measure individual and/or group-level opinions head-on.

One potential indicator of the validity of socially generated data is the extent of its consumption for factual content. Online media has expanded significantly over the past 20 years. For example, online news is displacing print and broadcast. More and more Americans distrust mainstream media, with a majority (60 percent) now having little to no faith in traditional media to report news fully, accurately, and fairly. Instead, people are increasingly turning to the Internet to research, connect, and share opinions and views. This was especially evident during the 2012 election where social media played a large role in information transmission (Gallup).

Politics is not the only realm affected by social Big Data. People are increasingly relying on the opinions of others to inform about their consumption preferences. Let's have a look at this:

- 91 percent of people report having gone into a store because of an online experience
- 89 percent of consumers conduct research using search engines
- 62 percent of consumers end up making a purchase in a store after researching it online
- 72 percent of consumers trust online reviews as much as personal recommendations
- 78 percent of consumers say that posts made by companies on social media influence their purchases

If individuals are willing to use social data as a touchstone for decision making in their own lives, perhaps this is prima facie evidence of its validity. Other Big Data thinkers point out that much of what people do online constitutes their genuine actions and intentions. The breadcrumbs left from when people execute online transactions, send messages, or spend time on web pages constitute what *Alex Petland* of MIT calls honest signals. These signals are honest insofar as they are actions taken by people with no subtext or secondary intent. Specifically, he writes the following:

> *"Those breadcrumbs tell the story of your life. It tells what you've chosen to do. That's very different than what you put on Facebook. What you put on Facebook is what you would like to tell people, edited according to the standards of the day. Who you actually are is determined by where you spend time, and which things you buy."*

To paraphrase, *Petland* finds some web-based data to be valid measures of people's attitudes when that data is without subtext or secondary intent; what he calls data exhaust. In other words, actions are harder to fake than words. He cautions against taking people's online statements at face value, because they may be nothing more than cheap talk.

Anthony Stefanidis of George Mason University also advocates for the use of social data mining. He favorably speaks about its reliability, noting that its size inherently creates a preponderance of evidence. This book takes neither the strong position of *Pentland* and honest signals nor *Stefanidis* and preponderance of evidence. Instead, we advocate a blended approach of curiosity and creativity as well as some healthy skepticism.

Generally, we follow the attitude of *Charles Handy* (*The Empty Raincoat*, 1994), who described the steps to measurement during the Vietnam War as follows:

> *"The first step is to measure whatever can be easily measured. This is OK as far as it goes. The second step is to disregard that which can't be easily measured or to give it an arbitrary quantitative value. This is artificial and misleading. The third step is to presume that what can't be measured easily really isn't important. This is blindness. The fourth step is to say that what can't be easily measured really doesn't exist. This is suicide."*

The social web may not consist of perfect data, but its value is tremendous if used properly and analyzed with care. 40 years ago, a social science study containing millions of observations was unheard of due to the time and cost associated with collecting that much information. The most successful efforts in social data mining will be by those who "measure (all) what is measurable, and make measurable (all) what is not so" (*Rasinkinski*, 2008).

Ultimately, we feel that the size and scope of big social data, the fact that some of it is comprised of honest signals, and the fact that some of it can be validated with other data, lends it validity. In another sense, the "proof is in the pudding". Businesses, governments, and organizations are already using social media mining to good effect; thus, the data being mined must be at least moderately useful.

Another defining characteristic of big social data is the speed with which it is generated, especially when considered against traditional media channels. Social media platforms such as Twitter, but also the web generally, spread news in near-instant bursts. From the perspective of social media mining, this speed may be a blessing or a curse. On the one hand, analysts can keep up with the very fast-moving trends and patterns, if necessary. On the other hand, fast-moving information is subject to mistakes or even abuse.

Following the tragic bombings in Boston, Massachusetts (April 15, 2013), Twitter was instrumental in citizen reporting and provided insight into the events as they unfolded. Law enforcement asked for and received help from general public, facilitated by social media. For example, *Reddit* saw an overall peak in traffic when reports came in that the second suspect was captured. Google Analytics reports that there were about 272,000 users on the site with 85,000 in the news update thread alone. This was the only time in *Reddit's* history other than *Obama AMA* that a thread beat the front page in the ratings (*Reddit*).

The downside of this fast-paced, highly visible public search is that masses can be incorrect. This is exactly what happened; users began to look at the details and photos posted and pieced together their own investigation—as it turned out, the information was incorrect. This was a charged event and created an atmosphere that ultimately undermined the good intentions of many. Other efforts such as those by governments (Wikipedia) and companies (Forbes) to post messages favorable to their position is less than well intentioned. Overall, we should be skeptical of tactical (that is, very real time) uses of social media. However, as evidence and information are aggregated by social media, we expect certain types of data, especially opinions and sentiments, to converge towards the truth (subject to the caveats set out in *Chapter 4*, *Potentials and Pitfalls of Social Media Data*).

Quantitative approaches

In this research, we aim to mine and summarize online opinions in reviews, tweets, blogs, forum discussions, and so on. Our approach is highly quantitative (that is, mathematical and/or statistical) as opposed to qualitative (that is, involving close study of a few instances). In social sciences, these two approaches are sometimes at odds, or at least their practitioners are. In this section, we will lay out the rationale for a quantitative approach to understanding online opinions. Our use of quantitative approaches is entirely pragmatic rather than dogmatic. We do, however, find the famous *Bill James*' words relating to the quantitative and qualitative tension to resonate with our pragmatic voice.

> *"The alternative to good statistics is not "no statistics", it's bad statistics. People who argue against statistical reasoning often end up backing up their arguments with whatever numbers they have at their command, over- or under-adjusting in their eagerness to avoid anything systematic."*

One traditional rationale for using qualitative approaches to sentiment analysis, such as focus groups, is lack of available data. Looking closely at what a handful of consumers think about a product is a viable way to generate opinion data if none, or very little, exists. However, in the era of big social data, analysts are awash in opinion-laden text and online actions. In fact, the use of statistical approaches is often necessary to handle the sheer volume of data generated by the social web. Furthermore, the explosion of data is obviating traditional hypothesis-testing concerns about sampling, as samples converge in size towards the population of interest.

The exploration of large sets of opinion data is what *Openshaw* (1988) would call a data-rich but theory-poor environment. Often, qualitative methods are well suited for inductively deriving theories from small numbers of test cases. However, our aim as sentiment analyzers is usually less theoretical and more descriptive; that is, we want to measure opinions and not understand the process by which they are generated. As such, this book covers important quantitative methods that reflect the state of discipline and that allow data to have a voice. This type of analysis accomplishes what *Gould* (1981) refers to as "letting the data speak for itself."

Perhaps the strongest reason to choose quantitative methods over qualitative ones is the ability of quantitative methods, when coupled with large and valid data-sets, to generate accurate measures in the face of analyst biases. Qualitative methods, even when applied correctly, put researchers at risk of a plethora of inferential problems. Foremost is apophenia, the human tendency to discover patterns where there are none; for example, a Type I error of sorts and dubbed patternicity by *Michael Shermer* (2008). A second pitfall of qualitative work is the atomistic fallacy, that is, the problem of generalizing based on an insufficient number of individual observations. The atomistic fallacy is real. Most people rely on advice from only a few sources, over-weighting information from within their networks rather than third parties such as Consumers Reports. Allowing an individual observation (for example, an opinion) to influence our actions or decisions is unreliably compared to what constitutes sensible samples in Consumers Reports.

The natural sciences benefited from the invention and proliferation of a host of new measurement tools during the twentieth century. For example, advances in microscopes led to a range of discoveries. The advent of the social web, with its seemingly endless amounts of opinionated data, and new measurement tools such as the ones covered in this book calls for a set of new discoveries. This book introduces readers to tools that will assist in that pursuit.

Summary

In this chapter, we introduced readers to the concepts of social media, sentiment analysis, and Big Data. We described how social media has changed the nature of interpersonal communication and the opportunities it presents for analysts of social data. This chapter also made a case for the use of quantitative approaches to measure all that is measurable, and make the one which is not so measurable.

In the next chapter, we will introduce R, which is the main tool through which we will illustrate techniques for harvesting, analyzing, and visualizing social media data.

2
Getting Started with R

In this chapter, we lay out the case for using R for social media mining. We then walk readers through the processes of installing, getting help for, and using R. By the end of this chapter, readers will have gained familiarity with data import/export, arithmetic, vectors, basic statistical modeling, and basic graphing using R.

Why R?

We strongly prefer using the R statistical computing environment for social data mining. This chapter highlights the benefits of using R, presents an introductory lesson on its use, and provides pointers towards further resources for learning the R language.

At its most basic, R is simply a calculator. You can ask it what 2 + 2 is, and it will provide you with 4 as the answer. However, R is more flexible than the calculator you used in high school. In fact, its flexibility leads it to be described as a statistical computing environment. As such, it comes with functions that assist us with data manipulation, statistics, and graphing. R can also store, handle, and perform complex mathematical operations on data as well as utilize a suite of statistics-specific functions, such as drawing samples from common probability distributions. Most simply, R is data analysis software adoringly promoted as being made by statisticians for statisticians. The R programming language is used by data scientists, statisticians, formal scientists, physical scientists, social scientists, and others who need to make sense of data for statistical analysis, data visualization, and predictive modeling. Fortunately, with the brief guidance provided by this chapter, you too will be using R for your own research. R is simple to learn, even for people with no programming or statistics experience.

R is a **GNU** (**GNU's Not Unix**) project, where GNU's Not Unix is a recursive acronym for GNU and is less commonly referred to as GNU S. R is freely available under the GNU General Public License, and precompiled binary versions are provided for most common operating systems. R uses a command-line interface; however, several integrated development environments are available for use with R, including our preferred one, RStudio.

The following nine important questions ought to drive whether to use R or some other statistical language:

- Does the software run natively on your computer?
    - R compiles and runs on a variety of Unix platforms as well as on Windows and Mac OS.
- Does the software provide the methods needed?
    - R comes with a moderate compliment of built-in functions and is wildly extensible through user-generated packages from a variety of disciplines.
- If not, how extensible is the software, if at all?
    - R is extremely extensible and extending it is simple. Packages are provided by a robust academic and practitioner community and are available for inclusion through simple downloads.
- Does the software fully support programming versus point-and-click?
    - Users can utilize R as an interactive programming language or a scripting language. There are also packages, such as Rcmdr, that allow limited point-and-click functionality.
- Are the visualization options adequate for your needs?
    - R has a very powerful, simple-to-use suite of graphical capabilities. Additionally, these capabilities are extensible just like R's other capabilities.
- Does the software provide output in the form you prefer?
    - R can output data files in many formats and can produce graphics in a wide range of formats as well.
- Does the software handle large datasets?
    - R handles data in memory; thus, users are constrained by the memory of their local system. However, within that constraint, R can handle vectors of up to 2 gigabytes in length. Packages can extend R to work in cloud computing environments.

- Can you afford the software?
    - R is free, as in free beer.

R is an open source software, which means that members of the public invented it and they now maintain and distribute it, as opposed to a corporation or other private entity. Mainstream reasons to use open source software have historically hinged on the free aspect, that is, free as in free beer. In the past, open source projects have often been plagued with serious drawbacks such as having limited functionality, being buggy, not staying up-to-date, and being difficult to get help with. However, open source projects such as R attract a large community of developers and users to overcome these issues. Furthermore, R has an expansive (and expanding) functionality and is constantly updated; thanks to the large number of people using and developing it, help is nearly always just a Google away. The open source nature of R makes it free, as in free beer, and also free, as in freedom from vendor lock-in, which is what Richard Stallman advocates as the best reason for moving to open source projects. As Mozilla's Firefox browser has commandingly demonstrated, open source software can be excellent and approachable as opposed to being aimed at niche users.

The excellence of R has several consequences, each of which in turn cause R to become better. First and foremost, R is extensible. Individuals can contribute add-on components called packages to R, which execute algorithms, create graphics, or perform other tasks. The number of these packages has grown exponentially over time; as of early 2014, there were over 5,000. Furthermore, many of these packages are multiplicatively useful when combined, making them more valuable as a whole than the sum of their individual utilities.

Secondly, R has a large and growing community of users and contributors, largely due to its excellence and broad utility. R has proven useful to so many that the traffic flow about it on e-mail discussion forums now outstrips the traffic on all of its main commercial contemporaries such as Stata, SAS, and SPSS. Similarly, the traffic related to R on Stack Overflow (`http://stackoverflow.com`), a software help forum, has outstripped SAS as well as some generic computing languages, such as PERL. Perhaps what's most telling is the fact that, at the time of writing this book (early 2014), more than half of the users on Kaggle (`http://www.r-bloggers.com/how-kaggle-competitors-use-r/`)—a site that promotes high-end data analysis competitions—use R.

R's popularity is indicative of its quality and broad utility. Additionally, the large number of active users make it much easier to get help with R through forums such as Stack Overflow and others (if R's built-in help documentation doesn't already answer your questions). Additionally, there are many books currently available in print that walk users through how to perform intermediate and advanced general programming in R as well as demonstrate R's use for particular domains (such as this one).

The justification for using R is overwhelming. We find R to have an excellent combination of freedom (both kinds), flexibility, and power. In addition, R has growing capabilities in handling Big Data in distributed systems or in parallel; some examples include **Distributed Storage and List** (**dsl**), **HadoopInteractiVE** (**hive**), **Text Mining Distributed Corpus Plug-In** (**tm.plug.dc**), **Hadoop Steaming** (**HadoopSteaming**), and **Amazon Web Services** (**AWS.tools**). So, let's get started.

Quick start

To install R, simply navigate to `http://www.r-project.org` and choose a mirror near you. Then, select whether you want R for Linux, Windows, or Mac. Finally, just follow the instructions from there, and you'll be up in no time. For additional FAQs, refer to `http://cran.r-project.org/faqs.html`.

In addition to installing R, you will almost certainly want to install an **integrated development environment** (**IDE**). An IDE is a programming environment that offers features beyond what is found by using the terminal or a command-line environment. These features can include code editing/highlighting/completion/generation, code compiling, code debugging, a file manager, a package manager, and a **graphical user interface** (**GUI**). These features will make generating and managing your R code simpler. There are a plethora of options, but we have a slight preference for RStudio, which is shown in the previous screenshot. It is recommended that you install RStudio (`http://www.rstudio.com`) before working through the examples discussed in this book. As we move forward, note that all of the following R code will be available online on the authors' web page and GitHub.

The basics – assignment and arithmetic

R allows access to the central math operators through their standard character representations: exponentiation (^), multiplication (*), division (/), addition (+), and subtraction (-). As you will see in the next example, R respects the order of operations.

The carrot (>) symbol denotes lines of code being inputted to R, while lines without (>) denote the output from R. In some circumstances, R will number its output; we'll point that out as it arises. Finally, R does not read code following a pound sign (#), which allows users to write comments for themselves and others right in the code itself:

```
> 2^4-3
13
> 2^(4-3)
2
```

Functions, arguments, and help

R has many built-in functions. A function is a programming construct that takes input, calls arguments, and turns them into output. Some functions only take one argument. An example of this is as follows:

```
> sqrt(16)
4
```

Other functions take several arguments. Generally, R can use the order in which you provide the arguments to understand the arguments respectively; however, it is a good practice to explicitly label your arguments, which are generally separated by commas. R does not read or care about spaces, but it is a good practice to include spaces between operators and after commas for better readability, as follows:

```
# Take the log of 100 with base 10
> log(100, 10)
2

# Though not necessary, it is best practice to label arguments
# This avoids confusion when functions take many arguments
> log(100, base=10)
2
```

To get help with a function, you can use the help function or type a question mark before a term. Using double question marks broadens the search, as shown in the following code:

```
> help(log)
> ?log
> ??log
```

Assignment is an important concept in R. We can assign values to an object, then treat that object as if it were the value it stores. An example should make this much more clear. Note the use of the left-facing arrow (<-) for assignment. Although you can assign with a right arrow or a single equals sign, only using the left arrow helps avoid confusion. An example of the assignment concept is shown as follows:

```
# Assign the value 3 to the object called 'my.variable'
> my.variable<- 3
# Work with the object
> my.variable * 2
[1] 6
# Create a new variable
> other.object<- my.variable + 7
> other.object * 2
[1] 20
```

R utilizes logical operators in addition to arithmetic operators. Logical operators are those that compare entities and return values of either `TRUE` or `FALSE`. Note that the double equals sign is used to ask a question, while the single equals sign is used for assignment (though the arrow should be strongly preferred for assignment to avoid confusion):

```
> 2==3
[1] FALSE
> 4 >= 4
[1] TRUE
"hello" != "HELLO"
[1] TRUE
```

Vectors, sequences, and combining vectors

Many R operations can be performed, or performed more efficiently, on vectors or matrices. Vectors are strings of objects; matrices are two-dimensional collections of objects, usually numbers. The `c` operator, which means concatenate, creates simple vectors, while the colon (`:`) operator generates simple sequences. To construct matrices, one simply passes a vector of data, the dimensions of the matrix to be created, and whether to input the data by row or by column (the default behavior is to input data by row). Examples of vectors, sequences, and matrices are given as follows:

```
> c(1,2,3,4,5)
1 2 3 4 5

> 1:4
1 2 3 4

> 5:-1
5 4 3 2 1 0 -1

> matrix(data=c(1, 2, 3, 4), byrow=TRUE, nrow=2)
1 2
3 4
```

For more complex sequence-like vectors, you can use the `seq()` function. At a minimum, it takes two arguments: `from` and `to`. You can additionally specify a `by` argument as well:

```
> seq(from=1, to=5)
1 2 3 4 5

> seq(from=2, to=6, by=2)
2 4 6
```

R also contains several constructs that allow access to individual elements or subsets through indexing operations. In the case of basic vector types, one can access the *i* th element by using `x[i]`, but there is also indexing of lists (which are simply collections of other data types), matrices, and multidimensional arrays (that is, matrices with more than two dimensions). In addition, R has a data type called a data frame, which is what many readers familiar with Stata, SPSS, or Microsoft Excel would think of as a dataset or spreadsheet. Data frames have column and possibly row names as well. R has three basic indexing operators, which is displayed in the following examples:

```
x[i]    # read the i-th element of a vector
x[i, j] # read i-th row, j-th column element of a matrix
x[[i]]  # read the i-th element of a list
x$a     # read the variable named "a" in a data frame named x
```

For lists, one generally uses `[[` to select any single element, whereas `[` returns a list of the selected elements. Many operators can work over vectors, as shown in the following code:

```
# divides each number in vector by 2
> c(1,2,3,4,5) / 2
0.5 1.0 1.5 2.0 2.5

# first vector divided by second
> c(1,2,3,4,5) / c(5,4,3,2,1)
0.2 0.5 1.0 2.0 5.0

# log base 10 of vector
> log(c(1,2.5,5), base=10)
0.00000 0.39794 0.69897

# new variable x is assigned resultant set
> x <- c(1,2,3,4,5) / 2
> x
0.5 1.0 1.5 2.0 2.5

# generic function 'summary' on variable x
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.5     1.0     1.5     1.5     2.0     2.5

# function to find mean
# notice mean is also captured by the generic function 'summary'
> mean(x)
1.5
```

A quick example – creating data frames and importing files

Getting data into R is often the first step in an analysis. R has a suite of functions called `read`, such as `read.csv()`, to help import data. Here, we assign the values read from a CSV file to an object called `mydata` as shown in the following code:

```
> mydata<- read.csv("http://www.ats.ucla.edu/stat/data/binary.csv")

# returns the first few rows of the data
> head(mydata)
```

```
  admit  gre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
```

To the initial confusion of some, several R functions behave differently depending on the type of object on which they act. As we saw earlier, the `summary()` function outputs descriptive statistics when it is given a vector. When given a data frame, it outputs summary statistics for each variable, as shown in the following code:

```
> summary(mydata)
     admit             gre             gpa
Min.   :0.0000   Min.   :220.0   Min.   :2.260
 1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130
Median :0.0000   Median :580.0   Median :3.395
 Mean   :0.3175   Mean   :587.7   Mean   :3.390
 3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670
 Max.   :1.0000   Max.   :800.0   Max.   :4.000
```

R has many built-in functions for fitting statistical models. For example, we can estimate a linear regression model, that is, a model that predicts the level of a continuous variable with another continuous variable(s), by **ordinary least squares** (**OLS**) with the first two lines of the next code. Note that the tilde (~) in the following code is used to separate the left-hand side of the equation from the right-hand side of the equation. In this simple regression example, we are regressing y on x, or gre (`mydata$gre`) on gpa (`mydata$gpa`). When the `summary` command is used with a model as the argument, parameter estimates are displayed along with other auxiliary information. Finally, we present the regression example as a demonstration of a classical method in social science used on structured data. This book departs from these classical methods and structured data:

```
> mydata.model<- lm(mydata$gre~mydata$gpa)
> summary(mydata.model)

Call:
lm(formula = mydata$gre ~ mydata$gpa)

Residuals:
     Min       1Q   Median       3Q      Max
-302.394  -62.789   -2.206   68.506  283.438

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   192.30      47.92   4.013 7.15e-05 ***
mydata$gpa    116.64      14.05   8.304 1.60e-15 ***
---
Signif.codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

One of R's great features is its extensibility. For instance, the `foreign` package allows users to import data formats other than those that R supports natively. To install a package, simply enter the following command in the terminal:

```
> install.packages("foreign", dependencies=TRUE)
```

The first argument to the function is the package name, and the second argument tells R to additionally install any other packages on which the one being installed is dependent. You will be asked to pick a mirror, that is, a location to download from. Choose one (it doesn't really matter which) and then input the following command to load the package:

```
> library("foreign")
```

To see all the different uses for this package, type `?foreign` as a command. One package that is particularly useful is the `sos` package, which allows you to search for other packages using colloquial search terms with the `findFn()` function. For example, when searching for a package that does non-linear regression, one could use the following command:

```
> library("sos")
findFn("non-linear regression")
```

Visualization in R

Visualization is a powerful tool for analyzing data and for presenting results. Many relationships and patterns that are obscured by summary statistics can be brought to light through visualization. The next graph shows a potent example of this. To begin with, let's look at some data that R comes with on the stopping distance of cars. This variable is contained in a dataset called cars, in a variable called **dist**. Histograms provide an informative way to visualize single variables. We can make a histogram with one line of code:

```
> hist(cars$dist)
```

R makes the histogram, decides how to break up the data, and provides default labels for the graph title and the x and y axes. Type the `?hist()` command to see other arguments to this function that change the number of bars, the labels, and other features of the histogram.

Anscombe's quartet comprises four small datasets with two variables each. Each of the sets has similar mean and variance for both variables, and regressions of y on x in each dataset generate nearly identical regression estimates. Overall, we might be tempted to infer that these datasets are nearly identical. However, bivariate visualization (of the x and y variables from each dataset) using the generic `plot()` function shows otherwise. At a minimum, the `plot()` function takes two arguments, each as a vector of the same length. To create the following four plots, enter the following commands for each pair of x and y (x1 and y1, x2 and y2, x3 and y3, and finally x4 and y4):

```
# par can be used to set or query graphical parameters.
# subsequent figures will be drawn in a n-row-by-n-column array (e.g.
2,2)
#par(mfrow=c(2,2))
> plot(anscombe$x1, anscombe$y1, xlab="x1", ylab="y1",
  main="Anscombe 1")
> abline(lm(anscombe$y1~anscombe$x1)
```

The code discussed earlier shows how to create your own x and y axis labels and plot titles. The `abline` call adds straight lines to an existing plot, in this case, the best fit (or regression) line of y on x. The output of the previous code is shown as follows:

This exercise not only demonstrates some simple graphical commands, but also the importance of visualization generally. In later chapters, we elaborate on these graphical methods to enhance analyses and the presentation of analytical results.

Style and workflow

Statistical programmers can think of R code—like other languages—as being dysfunctional, functional but awkward, or graceful. Graceful code is clear and readable, which helps prevent errors. Here are a few tips on writing graceful R code:

- Filenames should end in `.R` and be meaningful.
- Variable names should be short. If necessary, use a period to delineate multiword variable names (for example, `my.variable`).
- Keep every line short. R will not terminate a command until all parentheses are closed, so feel free to wrap commands across lines for brevity.
- Use spaces before and after assignment, after commas, and around parentheses and operators (such as +) for readability.
- Use the left arrow with hyphen (<-) for assignment, never the single equals sign.

For more details on writing good R code, refer to the guide at `http://google-styleguide.googlecode.com/svn/trunk/Rguide.xml`. Again, though R can be used interactively from within the terminal, it is best practice to develop code within an IDE, such as RStudio, so that it can be saved, changed, and rerun. Additionally, building version controls and persistence into your code by storing it on GitHub may be important, especially if you find yourself working in a group environment. Finally, many users will find the creation of projects useful—the RStudio documentation offers useful tips on this topic.

Additional resources

This chapter provides what we hope is a useful, though necessarily brief, introduction to R. There are many resources available that will help you expand your R programming skill set. What follows is a short list of our favorites:

- *A First Course in Statistical Programming with R* by *Braun and Murdoch* (2007)
- *The R Cookbook* by *Teetor* (2011)
- Quick-R: `http://www.statmethods.net/`

Summary

This chapter has set out a case for using R, the statistical computing environment for data handling and analysis due to R's zero cost feature, flexibility, and large support community. By now, you've seen how to import, summarize, and visualize datasets as well as run and plot simple regression models. In the next chapter, we discuss how to obtain the social media source Twitter.

3
Mining Twitter with R

An obvious prerequisite to gleaning insight from social media data is obtaining the data itself. Rather than presuming readers have social media data at their disposal, we show them how to obtain and process such data. Specifically, this chapter lays a technical foundation for collecting Twitter data in order to perform social data mining.

Why Twitter data?

In this chapter, we introduce a technical foundation for mining social data. We do so through a set of examples that focus on Twitter data, though the analyses are equally appropriate for data from other venues. That being said, our use of Twitter warrants a bit more explanation: what's so interesting about Twitter?

One answer to this question is **nothing**. Twitter is one of several social media networks, and there is little reason to suspect that data from Twitter is fundamentally different from other socially generated data. Another answer is that Twitter is different in subtle but important ways. One distinction is Twitter's ability to foster second-order connections, or what *Granovetter* (1973-1983) calls **weak ties**. These weak ties are important as they bring information to individuals from those with whom they share less, thus dramatically increasing information exposure. Second, Twitter, perhaps more than some other social networks, allows users to self-organize. A third answer is that Twitter users actively use Twitter to gather insight, make recommendations, and lodge public complaints. The extent to which users find this information valuable gives credence to the notion of its validity.

Obtaining Twitter data

As expected, before analyzing Twitter data, one must obtain the data. One reason we like R for social media mining is that it makes obtaining targeted portions of Twitter data (relatively) simple. Besides having the capacity to read standard data types and files from traditional statistical software packages, R can also read many other specialized formats. For instance, R can read relational databases, Hadoop, and some web formats such as Twitter. This chapter first covers how to obtain Twitter data before describing some simple exploratory data analysis techniques.

To begin ingesting social media data from Twitter, you will need a developer account on Twitter. You can start one (free of cost) at `https://dev.twitter.com/apps`. Once you have a Twitter account, return to that page and enter your username and password. Now, simply click on the **Create New Application** button and enter the requested information. Note that these inputs are neither important nor binding. You simply need to provide a name, description, and website (even just a personal blog) in the required fields.

Once finished, you should see a page with a lot of information about your application. Included here is a section called **OAuth settings**. These are crucial in helping you authenticate your application with Twitter, thus allowing you to mine tweets. More specifically, these bits of information will authenticate you with the Twitter **application programming interface** (**API**). You'll want to copy the consumer key, consumer secret, request token URL, authorize URL, and access token URL to a file and keep them handy.

Now that we have set up an application with Twitter, we need to download the R package that allows us to pull tweets into our local R session. Though there are several packages that do this, we prefer the `twitteR` package for its ease of use and flexibility. Instructions for downloading packages can be found in *Chapter 2*, *Getting Started with R*, but in general, installing packages is done by invoking `install.packages("…")`. You can download the `twitteR` package and load it into your R session as follows:

```
> install.packages("twitteR")
> library(twitteR)
```

Now, we are just a few lines of R code away from pulling in Twitter data. If you are using a Windows machine, there is an additional prestep of downloading a `cecert.pem` file, which forms a portion of certain types of certification schemes for Internet transfers, as shown in the following code snippet:

```
> download.file(url="http://curl.haxx.se/ca/cacert.pem",
destfile="C:/.../cacert.pem")
```

In this example, we have saved the file to the `C:` directory, but you can save it to wherever you have the appropriate permissions on your machine. Also, note the use of the backslash instead of the Windows-standard forward slash in the file locations. Next, create R objects from your own consumer information, filled in here with `XX` to indicate a placeholder, as seen in the following lines of code:

```
> my.key <- "XX"
> my.secret <- "XX"
```

With that done, pass this information to a function called `OAuthFactory`. The `requestURL`, `accessURL`, and `authURL` in the following code snippet are demonstrative, but you should verify this information with that provided by Twitter as a part of authorizing your application:

```
> cred <- OAuthFactory$new(consumerKey=my.key,
    consumerSecret=my.secret,
    requestURL='https://api.twitter.com/oauth/request_token',
    accessURL='https://api.twitter.com/oauth/access_token',
    authURL='https://api.twitter.com/oauth/authorize')
```

Finally, input the `cred$handshake` call that follows this paragraph, including the full path to where you saved your `cacert.pem` file. This will bring up a URL in the R console that you will have to copy and paste into a browser. Doing so will take you to a Twitter page that will supply you with a numeric code that you can copy and paste into your instance of R after the `cred$handshake` call.

```
> cred$handshake(cainfo="C:/.../cacert.pem")
```

Finally, save your authentication settings as follows:

```
> save(cred, file="twitter authentication.Rdata")
> registerTwitterOAuth(cred)
```

The `registerTwitterOAuth` function returns a value of `TRUE` on success; you are now ready to begin mining Twitter data, and after all of these steps, it will seem very simple. The workhorse of the `twitteR` package is a function called, appropriately, `searchTwitter`. The standard arguments to the function are a search term, a number of tweets to return, and providing the `cacert.pem` file downloaded previously. More information about the function, including how to search specific time frames, geographic locations, and more, can be found by typing `?searchTwitter`. For now, let's pull in some tweets with the `#bigdata` hashtag and save them to an object called `bigdata` as follows (note that you may leave off the `cainfo` argument on non-Windows machines):

```
> bigdata <- searchTwitter("#bigdata", n=1500, cainfo="cacert.pem")
```

We can find out what class or type of object `bigdata` is by using the `class` function as follows:

```
> class(bigdata)
[1] list
```

We easily discover that `bigdata` is a list or a collection of objects. We can access the first few objects in a list using the `head()` function as follows:

```
> head(bigdata)

[[1]]
[1] "Timothy_Hughes: RT @MarketBuildr: Prescriptive versus #predictive
#analytics http://t.co/oy7rS691Ht #BigData #Marketing"

[[2]]
[1] "DanVesset: already have on my schedule 3 upcoming business trips
to #Texas .... where all data is #BigData"

[[3]]
[1] "imagineCALGARY: Excited to be working on our methodology for
turning #bigdata into the story of #yyc's #sustainability journey:
http://t.co/lzPMAEQIbN"

[[4]]
[1] "ozyind: RT @timoelliott: #BigData Will Save the Planet!!!
http://t.co/Tumfrse5Kc by @jamesafisher #analytics #bi #marketing"

[[5]]
[1] "BernardMarr: Mining Big Data For Sales Leads http://t.co/
Xh5pBGskaG\n\n#bigdata\n#datamining\n#analytics"

[[6]]
[1] "mobiusmedia: RT @carmenaugustine: One size does not fit all:
\"It's up to each professional to determine what they mean by #bigdata
#discovery\" @jaimefit…"
```

You can access a particular object within a list by using double braces as follows:

```
> bigdata[[4]]
[1] "ozyind: RT @timoelliott: #BigData Will Save the Planet!!!
http://t.co/Tumfrse5Kc by @jamesafisher #analytics #bi #marketing"
```

There is no guarantee that `searchTwitter` pulled in the number of tweets requested. We may have specified a small date range or an uncommon search term. Either way, we can check the length of the `bigdata` list-type object with the `length()` function as follows:

```
> length(bigdata)
1500
```

Before we get too search-happy, it should be noted that the Twitter REST API (v1.1) limits the number of searches that can be performed in any given time period. The limits vary based on the type of search, the type of application making the search, as well as other criteria. Generally speaking, however, when using `searchTwitter`, you will be limited to 15 searches every 15 minutes, so make them count! More specific information on Twitter's rate limits can be found at `https://dev.twitter.com/docs/rate-limiting/1.1/limits`.

The main tip to avoid the rate limit becoming a hindrance is to search judiciously for particular users, themes, or hashtags. Another option is to more frequently search for users and/or themes that are more active and reserve less active users or themes to intermittent search windows. It is best practice to keep track of your searches and rate limit ceilings by querying in R, or by adding rate limit queries directly to your code. If you plan to create applications rather than merely analyze data in R, other options such as caching may prove useful. The following two lines of code return the current number of each type of search that remains in a user's allotment, as well as when each search limit will reset:

```
> rate.limit <- getCurRateLimitInfo(c("lists"))
> rate.limit
    resource                 limit    remaining   reset
1  /lists/subscribers        180      180         2013-07-23 21:49:49
2  /lists/memberships        15       15          2013-07-23 21:49:49
3  /lists/list               15       15          2013-07-23 21:49:49
4  /lists/ownerships         15       15          2013-07-23 21:49:49
5  /lists/subscriptions      15       15          2013-07-23 21:49:49
6  /lists/members            180      180         2013-07-23 21:49:49
7  /lists/subscribers/show   15       15          2013-07-23 21:49:49
8  /lists/statuses           180      180         2013-07-23 21:49:49
9  /lists/show               15       15          2013-07-23 21:49:49
10 /lists/members/show       15       15          2013-07-23 21:49:49
```

To limit the number of searches we have to undertake, it can be useful to convert our search results to a data frame and then save them for later analysis. Only two lines of code are used, one to convert the `bigdata` list to a `data frame` and another to save that data frame as a comma-separated value file:

```
# conversion from list to data frame
> bigdata.df <- do.call(rbind, lapply(bigdata, as.data.frame))

# write to csv; fill in the … with a valid path
> write.csv(bigdata.df, "C:/…/bigdata.csv")
```

Preliminary analyses

Text data, such as tweets, comes with little structure compared to spreadsheets and other typical types of data. One very useful way to impose some structure on text data is to turn it into a document-term matrix. This is a matrix where each row represents a document and each term is represented as a column. Each element in the matrix represents the number of times a particular term (column) appears in a particular document (row). Put differently, the i, jth element counts the number of times the term j appears in the document i. Document-term matrices get their length from the number of input documents and their width from the number of unique words used in the collection of documents, which is often called a **corpus**. Throughout this book, we utilize the `tm` package to create document-term matrices and for other utilities. The following lines of code install the `tm` package, preprocess our list object, `bigdata`, and turn it into a document-term matrix:

```
> install.packages("tm", dependencies=TRUE)
> library("tm")

> bigdata_list <- sapply(bigdata, function(x) x$getText())
> bigdata_corpus <- Corpus(VectorSource(bigdata_list))
> bigdata_corpus <- tm_map(bigdata_corpus, tolower)
> bigdata_corpus <- tm_map(bigdata_corpus, removePunctuation)
> bigdata_corpus <- tm_map(bigdata_corpus,
    function(x)removeWords(x,stopwords()))
```

The first line uses a common family of functions (`apply`, `lapply`, and `sapply`). These functions, in general, deploy a function across a range of structured data. `Sapply` traverses a matrix, performs a function to retrieve text from the tweets, and turns the text into a list object, `bigdata_list`. The next line of code uses the `Corpus` function to turn the list of tweets into a corpus. A corpus is an abstraction in R to represent a collection of documents. The final three lines of code convert all of the words to lowercase, remove all punctuation, and drop `stopwords`, respectively. `stopwords` are very common words that do not carry much meaning, such as "a," "are," "that," and "the." A more complete list of stop words can be found at `http://www.ranks.nl/resources/stopwords.html`.

At this point, we can take a first look at our data using the `WordCloud` package, which, unsurprisingly, creates word clouds. Let's have a look at the following code snippet:

```
> install.packages("wordcloud")
> library("wordcloud")
> wordcloud(bigdata_corpus)
```

Now, let's go on to build the document-term matrix (or in this case, term-document matrix).

```
> bigdata.tdm <- TermDocumentMatrix(bigdata_corpus)
```

Simply calling the name of a term-document or document-term matrix pulls up some basic information about it, including the number of terms and documents, the number of non-zero (that is, non-sparse) cells, and some other information. Using the `findFreqTerms` function, we can access some of the most common terms in our matrix as follows:

```
> bigdata.tdm
A term-document matrix (532 terms, 99 documents)
Non-/sparse entries: 1024/51644

Sparsity           : 98%
Maximal term length: 18
Weighting          : term frequency (tf)

# identify terms used at least 10 times
> findFreqTerms(bigdata.tdm, lowfreq=10)
[1] "analytics"   "big"   "bigdata"   "data"   "people"   "via"
```

We can also explore the data in an associational sense by looking at collocation, or those terms that frequently co-occur. From the previous list, `"people"` seems to be unexpected; so, we can explore the association of `"people"` and other terms in the corpus as follows:

```
> findAssocs(bigdata.tdm, 'people', 0.50)
2013    best    analytics
0.77     0.64          0.55
```

In the preceding code, each number is the correlation in the term-document matrix between the term it references and the word `"people"`. Another way to get a visual sense of a set of documents is to cluster them. Clustering, in general, is a way of finding associations between items (for example, documents). This necessitates a measure of how far each observation is from every other one. Nearby observations are binned together in groups, whereas the ones further apart are put into separate groups. There are many ways to implement clustering; here, we use a variant called **hierarchical agglomerative clustering**. Interested readers can find out more about clustering methods at `http://www.statmethods.net/advstats/cluster.html`.

We implement our chosen method by first removing the sparsest terms from our term-document matrix. Sparse terms are those which only occur in a small proportion of documents. By removing sparse terms, we reduce the length of the term-document matrix dramatically without losing relations inherent in the matrix. The sparse argument in the following line of code details the proportion of zeroes a term must have before being considered sparse:

```
# Remove sparse terms from the term-document matrix
> bigdata2.tdm <-removeSparseTerms(bigdata.tdm, sparse=0.92)

# Convert the term-document matrix to a data frame
> bigdata2.df <- as.data.frame(bigdata2.tdm)
```

Next, we scale the term-document matrix because clustering is sensitive to the scale of the data used. Specifically, the `scale` function subtracts every element in a vector from the vector's mean and divides each element by the vector's standard deviation. Once scaled, we use the term-document matrix to compute a distance matrix, where each row is a document and each column is the same set of documents. Each cell represents the distance between each pair of documents.

```
 # scale the data
> bigdata2.df.scale <- scale(bigdata2.df)

# Create the distance matrix
> bigdata.dist <- dist(bigdata2.df.scale, method = "euclidean")
# Cluster the data
```

```
> bigdata.fit <- hclust(bigdata.dist, method="ward")
# Visualize the result
> plot(bigdata.fit, main="Cluster - Big Data")

# An example with five (k=5) clusters
> groups <- cutree(bigdata.fit, k=5)
# Dendogram with blue clusters (k=5).
> rect.hclust(bigdata.fit, k=5, border="blue")
```

The preceding dendrogram shows our clusters in a tree diagram. The bottom row shows individual observations with groupings increasing in size as we traverse up the tree. Our sample was small and so our dendrogram is small, and our clusters may reflect groupings that do not generalize outside of our sample. For example, **bccmpls** and **lisarosie** reflect a conference (hashtag: bccmpls) in July 2013 where lisarosie was mentioned. This is mainly an artifact of our small sample and may not have been evident if we analyzed all Twitter data for 2013. Much less surprising is the clustering of **big** and **data**, both of which are closely related to analytics.

This chapter has laid the groundwork for the analysis of social media data. We have given you the tools to search, save, manipulate, and visualize social data. We urge readers at this point to go find and visualize some data for themselves before moving on to the next chapter.

Lastly, as a best practice, consider local- and cloud-based version control and code repositories. Both are great for professional, repeatable research and analysis efforts, and provide redundancy in the case of cloud-based version control systems. Git, GitHub, Dropbox, and Google Drive can all facilitate version control and code repositories. Git handles local version control and coupled with GitHub can move repositories to the cloud. GitHub stores data on the cloud (that is, remotely) to ensure that your code is preserved. Both Git and GitHub work well with RStudio. Dropbox and Google Drive offer only lightweight capability to do both version control and code repositories, but both are a bit more intuitive than Git. Understanding where your files are stored is a critical and often overlooked element in research and analysis. RStudio has some great mechanisms in place to manage files and projects. Whatever IDE you choose, learning more about managing data files, code, and plots are crucial to doing high quality work. In fact, these practices are so important that open source projects have been developed around this notion; one noteworthy example is ProjectTemplate (`http://projecttemplate.net`).

Summary

This chapter has provided an introduction to obtaining, manipulating, summarizing, and visualizing social media data. By now, you should be able to obtain Twitter data, save it, and store it locally in several formats. You should also be familiar with document-term matrices and several text exploration and visualization schemes. The next chapter discusses social media data from the perspective of social science research.

4
Potentials and Pitfalls of Social Media Data

Socially generated data, and especially social media data, comes with many complexities. Our ability to navigate these complexities as we describe and draw inferences from this data hinges on our thinking carefully about the potentials and pitfalls that arise in social media data. This chapter highlights some of the potentials and pitfalls of social media data.

Opinion mining made difficult

In this chapter, we highlight some of the potentials and pitfalls inherent in using social media data, and also in the tools we use to process it. These pitfalls are serious enough to warrant devoting an entire chapter to their enumeration and description. Though some of them cannot be ameliorated, we at least hope to give would-be analysts a fair warning so that they can enjoy their findings with an appropriately sized grain of salt.

In *The Empty Raincoat*, Handy described the first step to measurement as follows:

> "*The first step is to measure whatever can be easily measured. This is OK as far as it goes.*"

Twitter data is often used as the first step in measurement as it is easily obtained. University Professor and Princeton Fellow, Zeynep Tufekci (Tufekci, 2013), notes the danger in this practice using an analogy to the biological testing on Drosophila flies, better known as fruit flies. She points out that fruit flies are often used in laboratory settings because they are easy to breed and have simple life cycles. Tufekci cautions through a question: what if the fruit fly is systematically different than other organisms in ways beyond just their conveniently quick lifecycle? If so, the lessons learned from studying fruit flies may not generalize to other species. This would be doubly

problematic because their ease of use causes them to be the focus of a preponderance of genetic and other research.

Similarly, Zeynep notes that Twitter may be a model organism, that is, model in the sense of a model plane, not in the sense of a model student. Twitter data and other social media data is accumulated quickly and is easily mined from various websites' APIs. The pervasiveness and perceived richness of these sources of data make them obvious choices for researchers seeking to take the pulse of the public. However, the ease with which this data can be obtained does not necessarily correlate with quality inferences. Similar to the data obtained from Drosophila flies, social media data may have limited generalizing ability due to the fact that social media users are often young, urban, and middle-class. This and other such caveats to the use of social media forms the focus of this chapter. Layered on top of these inferential concerns is the special requirement that we be sensitive to the privacy of social media users—just because they have made their thoughts public does not mean we should shine a spotlight on individuals, lest we misrepresent them. In addition to discussing the inferential challenges surrounding the use of social media, this chapter also sets down some fundamentals about sentiment extraction and measurement.

The conversation in the remainder of this chapter takes a social science tilt. We feel that this approach and this background is especially important given that important segments of the social data mining community are steeped in the workings of computers and algorithms. Even though this is the case, they may not have had an occasion to be introduced to measurement theory, descriptive and causal inference, and the many fallacies that can plague our best efforts at deriving insights from data.

Sentiment and its measurement

Fundamentally, this book is about measuring the sentiments of human beings expressed through social media. To further this aim, we pause to set forth a structured definition of sentiment and characterize some ancillary components of sentiments. A sentiment is a view or attitude towards a person, place, or thing. It is an opinion that is directed and often has a valence. For our purposes, we consider seven different factors surrounding a sentiment, expanding on the classic quintuple (5-part object). The classic quintuple includes the holder h that expresses the sentiment; the target of the sentiment, g; the sentiment itself, s; the polarity or magnitude of the sentiment, p; and the source of the data, c (as distinct from the source of the sentiment).

As an initial foray into sentiment analysis, we could analyze the sentence, "Mary thinks her new sneakers are really delightful." Mary is the holder or expresser of the sentiment *delight*. The target is obviously her sneakers, and the magnitude of her sentiment is strong or high as indicated by the adverb *really*. The source of this datum is the authors' imagination. In addition to these factors, we often find it useful to note the time and location of the expression of the sentiment, though these are absent from this example. The previous example's simplicity belies many extraordinary challenges in the field of sentiment analysis. Consider the following tweet:

> *@<username> I looooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.*

The target of the sentiment is *Kindle2*, but a subtarget or the secondary target is Kindle DX; both are Amazon products and are viewed favorably. Kindle2 is loved by the user, but this is described atypically using *looooooovvvvvveee* instead of love. Many sentiment analysis techniques rely on dictionaries, which may include love, but almost certainly not *looooooovvvvvveee*.

Negation is also difficult to handle; consider sentences that include not, or more troublingly, not good or not bad. As we will see, negation can sometimes be partially captured by studying bigrams (two word pairs) instead of single words. This and other complexities serve to put something of an upper limit on the accuracy of our estimates of sentiment extracted from texts.

Not good and not bad negations complicate sentiment detection; intensification further compounds the difficulty of the task. Adjectives such as very, extremely, or hardly add or detract from the sentiment intensity. Superlatives form additional interesting cases such as "This wine is the most mediocre bottle I've ever purchased." Weighted terms in lexicons help mitigate negation and intensification, as does text scaling, discussed in *Chapter 6*, *Social Media Mining – Case Studies*.

Another example (Lui, 2012) that highlights further challenges is shown as follows:

> "*I bought a Canon G12 camera six months ago. I simply love it. The picture quality is amazing. The battery life is also long. However, my wife thinks it is too heavy for her.*"

This example contains a sentiment that is broken up across subtopics within the overall review of the camera, some with positive valence and some with negative. This review is useful to depict the classic quintuple that defines sentiment analysis. Here, the source is the person expressing the opinion, the targets are the various subcomponents of the camera, and the sentiments about the various targets are "love", "amazing", "long", and "heavy." The source *c* refers to where the textual sentiment information was found, be it Twitter, a blog, or a sentence or article in a magazine.

The nature of social media data

The mechanical differences between blogs and/or consumer reviews and Twitter are obvious. Blogs and consumer review sites allow users to contribute finite but possibly large amounts of text data, whereas Twitter has a document length limit of 140 characters. While at first this may seem like a major limitation of Twitter data, in fact, it proves to be quite useful. When faced with this stark limit, users tend to be pithy and accurate rather than loquacious and artful. This brevity makes sentiment extraction much simpler than it is for longer documents; however, it comes with its own unique challenges.

Throughout this book, social media data, and especially Twitter data, will be examined in increasing detail and with increasing sophistication. In order to accomplish this aim, we first need to get a sense of some of the complicating factors involved in studying sentiments. The first factor might well be that sentiment is often nonhomogeneous. For instance, groups of people may be split in their opinions about a topic. In fact, a single person may hold conflicting views as well. However, trying to condense group-level or individual-level sentiments into a single numeric measure may obscure this heterogeneity. For instance, a group of people with neutral views on a topic and a group with two vehemently opposed subgroups would both score nearly zero (or neutral) on many additive scales.

Traditional versus nontraditional social data

We compare social media data with traditional social science data in an effort to provide context for its analysis. In this section, we also make readers aware of the general classes of pitfalls that affect any measurements or inferences drawn from data. The ultimate goal of this discussion is to set out a sound analytical framework as well as generate an awareness of the limitations of social data and methods. This is so that analysts can "measure (all) what is measurable, and make measurable (all) what is not so" (Galileo, 2008) responsibly.

Prior to the era of high-velocity data, computational social scientists relied on two sources of observational (that is, nonexperimental) data: they could hand-code it themselves or use large, slow-moving datasets collated by governments and organizations. Collecting one's own data is slow and costly. It involves interviewing experts and checking for reference sources. This type of data collection is also prone to errors due to differences between coders or within coders over time if the collection is ever done again. An alternative is to use data sources that are large but slow to change.

For instance, the United States Census Bureau serves as a leading provider of demographic and economic data. The Census Bureau compiles mountains of data with broad coverage and high accuracy. However, the Bureau's best known collection, the Census of Population and Housing, is only collected once a decade and is often not appropriate for the sophisticated questions that we have today.

These surveys have been crucial to social scientific research. The multivariate nature of this data provides rich sets of independent and dependent variables. However, the aggregate and infrequent nature of institutional data often results in pooled cross sections of randomly sampled individuals at different points in time. Rather than analyzing groups at discrete points in time, or aggregating them into heterogeneous bundles, social media data allows the possibility of tracking groups and individuals over time.

Social media is captured at the individual level and at an extraordinary rate. Their continuous nature over time and space makes them ideal for multivariate analysis and cross-correlation. Furthermore, aggregating social data often proves to be much simpler than attempting to disaggregate institutional data. Put differently, inferential challenges surrounding the study of individuals are much more easily overcome than those surrounding the study of groups. We explore these issues one by one in the forthcoming sections.

The conversation about social media data often becomes a conversation about Big Data and how Big Data is hard to analyze. While some are caught up in defining Big Data in terms of storage (too big to fit in a relational database) or analysis (too big to use standard maximum likelihood techniques), we find that these definitional points are less crucial than a deep understanding of the measurement and inferential challenges inherent in dealing with social media data.

Measurement and inferential challenges

Many of the activities that fall under the umbrella term of data mining involve either measurement or inference, or possibly both. This section details some of the challenges researchers face when attempting to measure difficult social science concepts or trying to infer general patterns from subpopulation sets of data. These tasks, measurement and inference, are often one and the same in the social sciences. While one can use a ruler to measure height, there is no way to directly measure sentiment or affinity. Instead, we create proxy measures for these concepts and hope to make accurate inferences about these quantities.

Overfitting is a common problem in social science research, especially in Big Data. The goal of many analyses is to quantify a relationship accurately. In an effort to do so, many researchers make a model of a particular dataset that fits that data very nicely. That is, the model explains a high proportion of the variation in the data at hand. These researchers often erroneously conclude that a model that fits their data well is a high-quality model. However, of crucial importance is how well they have captured the relationships in the data generally, not just in their small sample of data. A set of relationships that characterize a sample (also called training data) but do not fit other data of the same type or source (commonly referred to as test data) is said to be overfit. Though overfitting is a common problem in small data settings, researchers should be wary of it when using Big Data as well. To avoid overfitting, data miners ought to use parsimonious models and cross-validate their models on different data or subsets of their initial data. We discuss these methods throughout the book as examples of good practice.

Big Data further complicates a pair of issues that pose challenges in smaller datasets as well. The first is technically referred to as mixtures of relationships. In plain language, this refers to there not being one pattern in a set of data, but several possibly conflicting patterns. For instance, suppose a drug has a positive effect on men but a negative effect on women. An imprudent researcher might look for an effect of the drug and find zero effect, assuming about half of the data belongs to each gender. Here, a mixture of patterns mask each other. This challenge in data analysis is exacerbated by the large number of possible patterns in data that includes many variables and data that covers broad swaths of time, space, groups, and individuals. Identifying mixtures is often best done by exploring interactions between variables and by visualizing your data.

The second concern exacerbated by very large datasets is the recovery of findings that are statistically significant but substantively tiny. Many statistical techniques not only quantify relationships, say, between variables x and y, but also provide measures of the extent to which those relationships are unlikely to be zero. Relationships that are found to be unlikely to equal zero are said to be statistically significant. However, researchers should be aware of the fact that relationships can simultaneously be non-zero and trivially small. This is especially true in Big Data applications, because our ability to discriminate effect sizes from zero usually increases with data size. To avoid this pitfall, make sure to assess the substantive importance of any findings you generate, not just their statistical robustness. For instance, if you find an increase of 2 percent in consumer sentiment related to your product, would this increase, even if statistically robust, be important? To whom and why? What if the effect size was estimated to be non-zero, still only at 1 or 0.5 percent?

The **modifiable areal unit problem** (**MAUP**) is a source of statistical bias that comes in two flavors: scale and zonal problems. The issue was discovered by *Gehlke* and *Biehl* (1934) and described more completely by Robinson (1950) and Openshaw (1984), who lamented the following:

> "*The areal units (zonal objects) used in many geographical studies are arbitrary, modifiable, and subject to the whims and fancies of whoever is doing, or did, the aggregating.*"

Openshaw and *Taylor* (1979) described how they had constructed all possible groupings of counties in Iowa into larger districts. When considering the correlation between percent of Republican voters and percent of elderly voters, they could produce a million or so correlation coefficients. A set of 12 districts could be contrived to produce correlations that ranged from -0.97 to +0.99. The modifiable temporal unit problem is the temporal companion of MAUP, but instead of geographic aggregation, the problem relates to temporal aggregation. To avoid finding spurious relationships when aggregating data, try to aggregate to a natural or substantively meaningful unit. Additionally, try aggregating to several units of varying sizes to ensure the robustness of your results.

Another concern stems from how you choose what data to analyze. When you build your sample based on the value of the variable of interest (that is, the dependent variable), you bias your study in a way that leads to low or even zero explanatory power. A toy example helps illustrate that if you want to study the factors that lead to business success, and thus examine 50 successful businesses and find that all 50 have CEOs that drive sports cars, you might conclude that this is a key cause of success. Obviously, such a study is flawed because of sampling of the dependent variable; to understand the causes of success, you would have to compare successful companies with unsuccessful ones. Though this seems obvious, even academic researchers fall victim to this error, such as the study of suicide terrorism done by *Robert Pape* (2003). Pape only looked at cases of suicide terrorism in an effort to make claims about the causes of suicide terrorism.

Self-selection is a constant concern in social science research. We must keep in mind that social data is volunteered by people, and that these people may not be indicative of the population generally. For instance, if we pull down geo-located Twitter data about President Obama, we must keep in mind that these tweets would almost entirely comprise users on Twitter (perhaps a young, upper-middle class group). This sample may not be representative of, say, likely US voters (whose modal member is older and middle class).

When interpreting our findings, we need to be careful not to fall victim to the ecological fallacy—the incorrect assumption that facts about about a group apply equally to members of the group. For example, if a researcher finds that men are more aggressive drivers than women, this does not imply that all men are aggressive drivers. Conversely, finding no relationship between income and infant mortality at the county or state level does not mean that there is no relationship between these factors at the household level. Ecological fallacy ought to be considered a special case of MAUP. In general, we can avoid these fallacies by making inferences only for the level of data that we have—inferences about groups from group-level data and inferences about individuals from individual-level data. Even more concrete advice is to try and match the level of data you collect with your research question of interest. For example, if you are studying household-level economic decisions, then attempt to capture household data!

As *Zeynep* (2013) astutely noted, social media mining and sentiment analysis often hinge on a plus one additive property where polarity is counted as the cumulative frequency of positive and negative words. However, not all words have equal impact, and some words scale differently than others. Later in this book, we discuss an unsupervised method that outlines this scaling problem and offers a solution.

The final pitfall we warn against is to watch out for dissimilar denominators that lead to intrinsic heterogeneity. This issue affects researchers who attempt to compare the sizes of different phenomena when a better measure would be a phenomenon's rate. For instance, if one tweet is retweeted 50 times and another is retweeted 100 times, upon first glance, we might be tempted to conclude that the second tweet was more interesting. However, what if the first tweet was only seen by a total of 100 people, while the second was seen by 10,000 people? Then, we could more accurately say that the first had a 50 percent retweet rate, while the second only had a 1 percent retweet rate. While the example makes it clear that the way to avoid dissimilar denominators is to pay appropriate attention to them, it is not always easy to obtain the appropriate denominator for any given metric.

Summary

Social data miners need to become more explicit in discussing, and where possible addressing, the pitfalls and assumptions that underpin their data and methods. Doing so will force researchers to be cautious and stay within appropriate inferential boundaries, while also allowing consumers of insights derived from social data mining to be aware of their limitations.

The upside of the model organism, Twitter, is that it provides a common, shared laboratory of interest that cannot be ignored by scientists, researchers, or practitioners; however, as was hopefully made clear in this chapter, harnessing it should be done with caution. Only through careful, thorough contemplation of the nature and structure of social data will mining provide us with answers to today's pressing social and human questions.

The last note we wish to make is that all data has shortcomings. Data is produced from an imperfect world, and we would therefore expect it to be imperfect. Regardless of whether your data is perceived to be from an authoritative source or not, one should still think in the terms outlined earlier. Measurement, reliability, validity, and potentials and pitfalls are pervasive considerations of data whether that data is small or big, good or bad, or traditional or nontraditional.

5
Social Media Mining – Fundamentals

Techniques used to extract sentiment from social media data are complex, at times counterintuitive, and often laden with assumptions. Before providing readers with a how-to guide to implement these models, we think it is critical to explain the techniques in depth so users can deploy them appropriately. This chapter explains the theoretical grounds for the techniques developed in the next chapter and serves as a bridge between the discussion of the pitfalls of social media mining and the execution of that mining.

Key concepts of social media mining

We find it useful to situate social media mining within the context of traditional social science research. While defining social science is difficult, Jean Anyon's perspective is a nice starting point. She suggests that socially explicit theory, and thus social science, should be empirically constructed, theoretically defensible, and socially critical. More generally, social science's main aims are to generate theories that explain individual-and group-level behaviors and then to examine the veracity of those theories with evidence. Generally, these theories are more valuable insofar as they allow a deeper understanding of human behavior, and especially so if they provide an understanding sufficient to allow for intervention. Our approach to social media mining strives to take this challenge to heart; thus, throughout this book, we use social media data to ask and answer questions of pressing social relevance.

Traditional social science not only focuses on important questions, but also seeks to uncover relationships that are interesting and unexpected. The world around us is full of complex social behavior; though identifying mundane facts is sometimes helpful in the name of basic research, it does little to help us understand social behavior. We take to heart the mandate to find interesting relationships as we mine social media data—a particularly complex and rich source.

At heart, however, social science is not a focus on the important or the interesting. It is science, which means that it is a set of methods and practices designed to generate and verify facts. The logic of science, regardless of whether it proceeds quantitatively or in a qualitative fashion, is fundamentally about knowledge discovery and accumulation. This logic helps mitigate several shortcomings in reasoning that frequently hinder our ability to make correct inferences. Some examples include illusory correlations (perceiving correlations that do not exist), selective observation (inadvertently cherry-picking data), illogical reasoning, and over or under generalizing (assuming that facts discovered in one domain apply to others as well). Generally, the scientific process helps avoid the discovery of false truths often arrived at through deduction, speculation, justification, and groupthink.

Good data versus bad data

Traditional social science data differs markedly from social media data in several respects. First and foremost, traditional social science data is most often collected in targeted and rigorous ways. For instance, the US census targets nearly the entire US population and has a strong methodology for attaining this target. Researchers interested in the sentiments of particular demographics can target them specifically through surveys or polls, and can additionally tune survey instruments to carefully elicit the information they desire.

The steep downside to these classes of data sources is that they are often extremely limited in their geographic or temporal coverage. As such, they do not allow for broad generalizations or comparisons across place and time. Broader surveys, such as the US census, capture information about a large number of people, but usually only capture cursory descriptive information. Furthermore, this information is captured infrequently and in ways that are incomparable across borders. Narrower surveys, such as those fielded by researchers and firms, obviously are limited in their ability to support inferences about broad populations.

These data sources, despite their shortcomings in terms of coverage, are held in high regard due to their focused and authoritative nature. These sources are used due to the fear that bad input data will yield low-quality inferences, or "garbage in, garbage out," as the saying goes. But, what is garbage data? Should we consider social media data garbage because of its unfocused nature? Also, under what circumstances might we be willing to use social media data?

Our view is that focused, purpose-collected data is the best option when it is available. This statement may come as a surprise in a book on social media mining, but the linchpin of the statement is the phrase *when it is available*. For the vast majority of emerging questions related to business, politics, and social life, purpose-collected social science data-sets simply do not exist. As such, we take the pragmatic position that social data, due to its broad coverage and large volume, makes a nice fallback to targeted data. Social data is bad in the sense that much of it will be inapplicable to any particular question; however, limited applicability is certainly better than utter absence of data. The reality is that we live in an imperfect world, which will consequently yield imperfect data. Our job, as data analyst is to work with data in responsible ways.

This book does not cover how to handle poor, dirty, missing, or incorrect data in a comprehensive manner. However, we do wish to promote the use of social media data and its utility in cases where traditional social science data-sets do and do not exist and where there are low and high barriers to targeted collection.

Traditional social science modeling techniques tend to require data-sets in which observations are independent of one another. However, data gleaned from social media outlets, such as Twitter, is almost certainly not independent. That is, data is not randomly sampled from a larger population and thus each observation is likely to be related to observations that are nearby in some sense. For example, tweets about a large public event arise around the same time and from the same area. Also, many may express similar views. This nonindependence has implications for how you handle tweets given their degree of centrality, shared geography, and repetition through retweets. Although we do not often study sentiment polarity explicitly in terms of networks, doing so may prove useful for future researchers. We anticipate research in that direction will produce better measures and predictions, localized lexicons, and other advantages.

Understanding sentiments

Social media mining can and should have broad interpretation. It is not the intent of the authors to confine social media mining to sentiments or opinions, but rather we suggest that a sentiment or opinion is a useful tool for many research pursuits.

Until recently, sentiment was understood as a ubiquitous and constant part of the human experience, with variations in sentiments changing only slightly up or down. Klaus Scherer (2000) developed a working definition as follows:

> "*Emotions (sentiments) are episodes of coordinated changes in several components in response to external and internal events of major significance to the organism.*"

It is our intent to understand, measure, and interrelate these changes in a sentiment. Scherer's typology of emotions is a useful grounding point for the understanding of sentiments, and as a jumping-off point for a discussion of the difficulty in measuring sentiment-laden text.

Scherer's typology of emotions

Scherer's typology of emotions is briefly explained as follows:

- **Emotion**: This is a brief, organically synchronized evaluation of a major event, for example, being angry, sad, joyful, ashamed, proud, or elated
- **Mood**: This is a diffused, non-caused, low-intensity, long-duration change in subjective feeling, for example, being cheerful, gloomy, irritable, listless, depressed, or buoyant
- **Interpersonal stance**: This is an affective stance towards another person in a specific interaction, for example, being friendly, flirtatious, distant, cold, warm, supportive, or contemptuous
- **Attitude**: This is enduring, affectively colored beliefs or dispositions towards objects or persons, for example, being liking, loving, hating, valuing, or desiring
- **Personality traits**: These are stable personality dispositions and typical behavior tendencies, for example, being nervous, anxious, reckless, morose, hostile, or jealous

Generally, when we try to measure a sentiment, we talk about Scherer's emotions; though, in some situations, we might try to capture longer-term phenomena such as moods.

Anchoring the neo-social science approach using Twitter data versus other types of social media data is important as well because not all data is equal. Twitter data differs from data derived from sites such as Yelp and Google Reviews due to the simple fact that Twitter does not have ratings or explicit targets. If we want to know the sentiment of a given source or topic, whether it is the iPhone 5S or something more sensitive such as social policy, we have to discover that signal in a corpus of other signals. However, Yelp and Google Reviews (just two examples of many) have explicitly accounted for the source or topic by design and have ratings designed to measure sentiments.

A tweet is what Twitter users send to each other and to the Twittersphere. A tweet is sometimes a sentence and other times not, but it is restricted to 140 characters or approximately 11 words. Twitter therefore provides sentence-level sentiment analysis as opposed to reviews on Yelp or Google Review, which usually constitute the entire documents.

Sentiment polarity – data and classification

Social media mining primarily involves the following two steps:

1. Identifying and retrieving content related to the topic of interest.
2. Measuring the polarity of each datum.

The first step, message retrieval, requires some a priori insight into the topic of interest. The goal of message retrieval is to seek out only the messages or pieces of text that contain sentiment-laden content related to a particular topic. This topic could be almost anything of interest, subject to the constraint that information exists about it on public social media. For instance, in *Chapter 3*, *Mining Twitter with R*, we examined the topic Big Data, and in *Chapter 6*, *Social Media Mining – Case Studies*, we delve into social issues such as abortion and the economy. Atmospherics, that is, data gathered in an effort to track local sentiments with regards to economic, cultural, or political topics, can also be analyzed, as we do in *Chapter 6*, *Social Media Mining – Case Studies*. Lest readers think that social media is too diffuse to be useful, as of the writing of this book, at least one hedge fund uses atmospherics gleaned from Twitter to gauge stock prices.

To gather data, we generally collect content that contains a manually specified (set of) keyword(s). This is called the target. For example, the target for presidential approval would use the topic keyword **obama**. We may wish to add context to analyses done on particular keywords by adding additional opposing or specifying keywords. For example, in addition to **obama**, we could add **romney** to provide a counterpoint if we were studying the 2012 presidential election campaigns. Depending on the purpose of our analysis, we could jointly search for, say, **obama** and **economy** to target more specific subjects.

Topic models represent a second, more sophisticated, and potentially more thorough way of capturing bits of text that are relevant to a particular analysis. These models take very large sets of documents as their inputs and group them probabilistically into estimated topics. That is, each document is proclaimed to be a mixture of one or more topics that are themselves estimated from the data. This allows users to find texts that are related to a topic, though they may not explicitly use a particular keyword. The details of this class of statistical models are outside the scope of this text; however, in *Appendix*, *Conclusions and Next Steps*, we point readers to references on the theory and estimation of this exciting new class of tools.

Social data mining is the detection of attitudes, and the easiest way to understand it is through the following structure:

sentiment = {data source, source, target, sentiment, polarity}}

The parameters are explained in detail as follows:

- **Data source**: This relates to understanding the source of the data; that is, is the source a sentence or an entire document? Twitter or a blog?
- **Source or holder**: This is the one that expresses a sentiment or an opinion,
- **Target or aspect**: The target or aspect is what or to whom the sentiment is directed toward.
- **Type of sentiment**: This is the type(s) of emotion(s) expressed, that is, like, love, hate, value, desire, and so on.
- **Polarity**: These are juxtapositional sentiments on a dimension, that is, positive or negative.

The following examples highlight these components and also some of the challenges involved in sentiment analysis. We have parts of two reviews: one about *Steven Spielberg* and another about *John Carpenter*. In both examples, the data source is the **Internet Movie Database** (**IMDB**) that considers itself the world's most popular and authoritative source for movie, TV, and celebrity content. The holder is the one who wrote the review, and the targets are *Steven Spielberg* and *John Carpenter* respectively. However, the target is complicated by mentions of various movies over time. Also, complicating matters is the variety of sentiment types and polarities.

> *Steven Spielberg's second epic film on World War II is an unquestioned masterpiece. Spielberg, ever the student on film, has managed to resurrect the war genre by producing one of its grittiest and most powerful entries. He also managed to cast this era's greatest answer to Jimmy Stewart, Tom Hanks, who delivers a performance that is nothing short of an astonishing miracle for about 160 out of its 170 minutes; Saving Private Ryan is flawless, literally!*
>
> *There was a time when John Carpenter was a great horror director. Of course, his best film was 1978's masterpiece Halloween; however, he also made The Fog in the 1980s and 1987's underrated Prince of Darkness. Even, Heck made a good film, In the mouth of madness, in 1995. However, something terribly wrong happened to him in 1992 with the terrible comedy Memoirs of an Invisible Man.*

Supervised social media mining – lexicon-based sentiment

Lexicon-based sentiment classification is perhaps the most basic technique for measuring the polarity of the sentiment of a group of documents (that is, a corpus). Lexicon-based sentiment measurement requires a dictionary of words (a lexicon) and each word's associated polarity score. For example, a lexicon may contain the word excellent, which might have a score of positive two. Similarly, the word crummy may score negative one and a half. In the simplest implementation of lexicon-based sentiment analysis, all of the words in a document are compared to the words in the lexicon. Every time a word is used that is in the lexicon, the associated score is added to that text's overall sentiment score. For example, the sentence "I found the customer assistance to be excellent," would score a positive two.

A lexicon-based sentiment often entails merely counting the opinion words from a subset of data from a particular source. This approach certainly has errors, as does perhaps all natural language processing; however, in aggregate, the lexicon-based approach has proven to be fairly robust, even when only used on subsets. Additionally, there are many possible ways to aggregate the sentiment scores of each word, but most commonly, they are simply summed up to form an overall score for a document.

Despite lexicon-based sentiment classification being considered here as basic, it is still difficult. This is primarily due to the fact that when counting words with positive or negative valences, one must decide which words to count as each. Different dictionaries of positive and negative words can generate different sentiment scores for the same sentences. Some words with perceived sentiment are more neutral, while others have perceived neutrality, but are in fact more extreme. This challenge arises, in part, due to varied usages of words within and across contexts.

Preassembled lexicons are incredible resources and are applicable for a wide variety of problems—we use several in *Chapter 6*, *Social Media Mining – Case Studies*. Despite subtle differences, they are all good starting points, but they are just that, starting points and not end points. Rather than utilizing a preassembled lexicon indiscriminately, researchers should often develop lexicons that are sensitive to the domain they are analyzing. For instance, a lexicon that is useful for economic atmospherics (where moderate and stable are positive) may prove useless for examining political leanings. Preassembled domain-specific lexicons exist as well and two popular economic lexicons will be used later. There are many approaches to extending both generic preassembled lexicons as well as domain-specific preassembled lexicons, and we will describe two rather intuitive ones, dictionary-based lexicons, and corpus-based lexicons in addition to preassembled lexicons.

Both dictionary-based and corpus-based approaches augment preassembled lexicons in one of the following two ways:

- Using a dictionary (that is, synonyms and antonyms) to add keywords external to our corpus to enhance our preassembled lexicon(s)
- Using the corpus directly to add words already internal to our corpus that are keywords but are not accounted for by preassembled lexicons

Merging preassembled lexicons, dictionary-based lexicons, and corpus-based lexicons offers the best chance to successfully estimate sentiment.

The two approaches (dictionary and corpus) produce empirically constructed lexicons that seek to calibrate the underlying sentiment by adding to the preassembled lexicons. The intuition is words appearing in the complete collection of lexicons (preassembled, dictionary, and corpus) within our set of documents returned from a narrow topic (the search set) are more likely to objectively describe sentiment information. In the lexicon approach, it is sufficient to simply count the frequency of words from our lexicons with the set of documents returned from our topic of interest and sum the results over time, by space or by product. The next chapter outlines this process in detail and sums the results' set over time where the target is the US economy.

Supervised social media mining – Naive Bayes classifiers

Methods to extract sentiments from documents can be broadly classified into supervised and unsupervised approaches (semisupervised approaches are also available but are outside the scope of this text. Interested readers can consult Abney (2007)). Supervised methods are those that utilize data that has been tagged or labeled. In the parlance of statistics, these approaches utilize observations with both independent and dependent variables. For instance, the following Naive Bayes classifier approach involves a training dataset of documents that have already been scored as having positive or negative sentiment; a statistical model based on these forms the basis of scoring further documents. In contrast, unsupervised learning algorithms do not require a dependent variable to be provided. For instance, the IRT-based method described later in this chapter scales documents along a continuum of sentiments with no need to provide a labeled training set. Additionally, lexicon-based approaches mentioned earlier can also function without prelabeled observations.

The Naive Bayes classifier, in spite of its unfortunate name, turns out to be a highly useful tool for sentiment analysis. At the most general level, the Naive Bayes classifier is exactly that: a classifier. Classifiers are statistical tools that are used for, among other things, predicting which of two or more classes a new observation belongs to. In our case, we want to train our classifier to be able to distinguish documents featuring positive sentiment from those featuring negative sentiment (the two types or classes of interest). To do so, we feed the algorithm a large set of documents that are already coded as containing positive or negative sentiments about a particular topic. Then, if all goes as planned, we can pass new documents to the model and have it predict the direction of their sentiment, or valence, for us. The downside to this and other supervised techniques is having to handcode a sufficient set of initial training data.

So, where did the naive part of the name come from, and why is this method useful in spite of its self-assumed simplicity? The goal of any classifier is to determine which class or type a new observation belongs to based on its characteristics and previous examples from both types that we have seen before (that is, from existing data). Some types of classifiers can account for the fact that the characteristics we use for this prediction may be correlated. That is, if we are trying to predict whether an e-mail is spam or not by looking at what words and phrases the e-mail contains, the words easy and money are likely to co-occur (and are likely to be indicative of spam messages). The Naive Bayes classifier does not try to account for correlations between characteristics. It just uses each characteristic separately to try to determine each new observation's class membership.

The naive assumption that all of the characteristics of an observation are unrelated is always wrong. In predicting whether or not to extend a loan to an individual, a bank may look at their credit score, whether or not they own a home, their income, and their current debt level. Obviously, all of these things are likely to be correlated. However, ignoring the correlations between predictive characteristics allows us to do two things that would otherwise be problematic. First, it allows us to include a huge number of characteristics, which becomes important. This is because in text analysis, individual words often have predictive characteristics, and documents often contain thousands of unique words. Other models have a difficult time accommodating this number of predictors. Secondly, the Naive Bayes classifier is fast, thus allowing us to use large training sets to train a model and to generate results quickly.

Unsupervised social media mining – Item Response Theory for text scaling

The techniques set out earlier for scaling or classifying sentiments in texts are fairly robust; that is, they tend to work well under a wide variety of conditions such as heterogeneous text lengths and topic breadths. However, each of these methods requires substantial analyst input, such as labeling training data or creating a lexicon. **Item Response Theory** (**IRT**) is a theory, but will be used in this text to refer to a class of statistical models that rely on that theory, providing a way to scale texts according to sentiment in the absence of labeled training data. That is, IRT models are unsupervised learning models.

IRT models were developed by psychologists for scoring complex tests and were then picked up by political scientists who employ them for scaling legislators. We will briefly explain the legislative context as that will help readers build intuition over how models work when applied to scaling texts. Consider a set of V voters, such as US Senators, who, over the course of a year, vote on B bills. For simplicity, assume each voter can only vote **yes** or **no**. We could then put all of the data into a matrix, where each row represents a voter and each column a bill. Each cell then represents a particular voter's decision on a particular bill, that is, yes (1) or no (0). Now, we need to make two related assumptions. The first is that all or most of these voters can be described as lying along a single underlying continuum. The second more trivial assumption is that this position influences their votes, at least on some bills. With these assumptions in place, we can estimate a statistical model that describes the probability of each cell in our data matrix being a one or a zero. The model is a function of each bill's difficulty of being voted for (that is, how controversial it is), each voter's position on the underlying scale, and how strongly each bill is affected by voters' locations on the scale. Technically, we estimate a logistic regression as follows:

$$pr(y^{vb}=1) = logit(b1^{b}*x^{v} - b0^{b})$$

Here, x is the scaled position of each voter (v), $b0$ is the difficulty of voting yes for each bill (b), and $b1$ is the degree to which each voter's position affects their proclivity to vote in favor of each bill (b). Positive values of $b1$ mean that voters to the right are more likely to vote in favor of a bill, and negative values of $b1$ mean that senators to the left are more likely to vote for a bill.

As you will see, we apply the previous assumption to the analogous case of text scaling. To do so, we create a matrix with rows representing authors or documents (instead of voters) and columns representing words or phrases, (instead of bills). Each cell represents whether or not a particular author used a particular word or phrase. We modify the previous assumptions: authors lie along a sentiment continuum, and their placement affects their pattern of word use. The first part of this assumption is limiting. We can only apply the method to sets of documents that are sufficiently narrow to be usefully described by a single underlying continuum, and that continuum must essentially be the sentiment we are trying to measure. The results of this analysis are a continuous scaled measure of author (or document) location (x) as well as estimates of the weights for each word or phrase ($b1$). This scaled measure of location (x) represents the author's sentiment towards the topic under study if the previous assumptions are met.

The IRT-based method described here has mixed properties. It requires no training data, little subject matter expertise to employ, is language agnostic (that is, could function on any language), and generates a quantitative (instead of merely a binary) measure of a sentiment. However, this model can only be applied to documents that are all about the same topic, can only estimate a single underlying dimension, can be slow to estimate, and is not guaranteed to converge.

Summary

In this chapter, we learned key concepts related to sentiment analysis. Sentiment was defined and difficulties related to its mining were covered. We then walked through the theoretical underpinnings of three different models for sentiment analysis. We intentionally separated the details of implementation from theoretical concerns in hopes of giving readers an appreciation for the methods, including their strengths and weaknesses.

The next chapter delves into the details of implementing the classes of models described earlier.

6

Social Media Mining – Case Studies

The importance of examples cannot be downplayed as they help us to understand and enhanced understanding often leads to subsequent improvement of our skills. While this chapter represents a sizable minority of the overall book, it also represents the proportion of time spent during modeling, that is, only a sizable minority.
The previous chapters have established a solid groundwork of key concepts and foundational knowledge such that readers can now responsibly digest, comprehend, and execute the case studies discussed in this chapter. This pivotal chapter provides accessible material and tangible examples, including lexicon-based, supervised, and unsupervised approaches to sentiment analysis.

Introductory considerations

As promoted often throughout this book, social data mining can be about more than mere product reviews. This is not to suggest that these methods are used exclusively for marketing or business, nor that using these methods for such analyses is unimportant. It is merely an acknowledgment that our goals are largely about investigating socially critical issues, such as abortion, gun control, and immigration, or parts of issues broadly set within the health, economic, and political categories. These are perspectives that remain intrinsically important to the human condition and societal progress. To that end, we have intentionally chosen topics that are hard-hitting. Also, while we promote social media data, specifically Twitter, we also find utility in these methods on varied datasets, both big and small. Consequently, examples include data from the Web that is not from Twitter, and we simultaneously highlight nuances to consider when working with varied datasets. Furthermore, social data mining is often about Big Data, and R does a good job with large datasets, but size can become a consideration when working with real-world datasets.

When working with larger files, in general, you should consider the following:

- How large is your dataset?

  Generally, the number of rows and columns is a good estimation given the content in each cell, that is, numeric versus character.

- How much memory does your system have?

  You may want to avoid reading in data with file sizes greater than the memory available to you. As a rule of thumb, the overhead associated with reading data to memory, which is the default behavior of R, is about double. Therefore, if you estimate your data to be 3 GB, then the memory required is roughly 6 GB. Most computers now have 8 GB and even 16 GB of RAM, but if you do not have enough system memory, then some social media mining applications may be intractable.

- How many open applications do you have, and what are they?

  If you think you will be broaching the limits of your system, then you may want to consider closing applications or reading your data into memory at a later time.

- What is your operating system? Is it 32-bit or 64-bit?

  Some operating systems are more efficient, and having a 64-bit machine will allow increased access to memory.

We suggest you read the help page for `read.table` and `read.csv`; both offer simple mechanisms to gracefully handle larger datasets. `colClasses` is another option that should be considered. This option takes a vector whose length is equal to the number of columns in your table. By specifying this option instead of using the default we can tune R to load much faster since R will know in advance what the columns are and know their class. Also, by specifying the `nrows` argument we tune the internal memory usage. When R doesn't know how many rows it has to read it makes some rather crass estimations, and when it underestimates the memory demands, it allocates more memory. The constant allocations take time, and if R overestimates the amount of memory it needs, your computer will run out of memory. Even a mild overestimate for `nrows` is better than none at all.

Case study 1 – supervised social media mining – lexicon-based sentiment

The *Beige Book* (`http://www.federalreserve.gov/monetarypolicy/beigebook`), more formally called the *Summary of Commentary on Current Economic Conditions*, is a report published by the **United States** (**US**) **Federal Research Board** (**FRB**) eight times a year. The report is published by each of the Federal Reserve Bank districts ahead of the Federal Open Market Committee meeting and is designed to reflect economic conditions. Despite being a report published by the US FRB, the content is rather anecdotal. The report interviews key business contacts, economists, market experts, and others to get their opinion about the economy.

The *Beige Book* has been in publication since 1985 and is now published online. The data used in this book can be found on GitHub (`https://github.com/SocialMediaMininginR/beigebook`), as well as the Python code for all the scraping and parsing.

An example from the full report of the *Beige Book* (October 2013) is shown as follows, which will give you some idea about the nature of the content. The full report is an aggregated view from the 12 Federal Reserve Bank districts:

> *Consumer spending grew modestly in most Districts. Auto sales continued to be strong, particularly in the New York District where they were said to be increasingly robust. In contrast, Chicago, Kansas City, and Dallas indicated slower growth in auto sales in September.*

The *Beige Book* differs from Twitter in numerous ways: not everyone has the freedom to participate, the data points are not socially linked, and users cannot respond to one another directly. For our purposes, however, the most important difference is that the *Beige Book* contains paragraphs of information per document rather than being a collection of single sentences as is the case with Twitter.

For simplicity, the data has been collapsed over space. Other versions of the data include longer temporal ranges of the data and can be found on the authors' GitHub account (`https://github.com/SocialMediaMininginR/beigebook`). These datasets include disaggregated geographic views of reports by city and disaggregated views by topic. As the earlier quote (October 2013) indicates, and as intuition may suggest, the economic conditions in the US are nonstationary; that is, regional variation exists due to economic shocks affecting cities and regions that are similar either geographically or functionally. A more robust analysis of the data might include sentiment analysis by city and include neighborhood effects—we simplify the analysis here for expository purposes by omitting these complicating factors.

As we discussed in the previous chapter, a necessary requirement of lexicon-based approaches to measuring sentiment is to procure lexicons against which our data needs to be tested. There are many extant dictionaries that vary in terms of how they were generated (manually versus empirically) and their breadth (general or subject-specific). We use two preassembled lexicons and augment each with additions, one empirically induced from the corpus and the other dictionary-based. Our general opinion lexicon was created by *Hu* and *Liu* (`http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`), and our domain-specific lexicon was created by *Tim Loughran* and *Bill McDonald* (`http://www3.nd.edu/~mcdonald/Word_Lists.html`). While one is domain-specific (*Loughran* and *McDonald*) and the other is for more general use (*Hu* and *Lui*), both lexicons offer broad utility for textual analysis, natural language processing, information retrieval, and computational linguistics.

Let's start by loading our data, sentiment function, and lexicons. The `https_function`, located on the authors' GitHub account (`https://github.com/SocialMediaMininginR/https_function/blob/master/https_function.R`), will load R files directly into your R session over HTTPS. This allows us to centrally store code and data and to make it available in a simple, verbatim manner. The code present on GitHub is shown as follows:

```
# Run the following code - (Breyal)
https_function <- function(url, ...) {
  # load package
  require(RCurl)

  # parse and evaluate each .R script
  sapply(c(url, ...), function(u) {
    eval(parse(text = getURL(u, followlocation = TRUE,
    cainfo = system.file("CurlSSL", "cacert.pem",
    package = "RCurl"))), envir = .GlobalEnv)
  })
}
```

It should be noted that when performing this analysis, several tabs were used in RStudio. If you are using RStudio for the first time, you may want to consider how to make best use of the tabs for general organization. The following R code will appear to be all in one block, but in practice, the material was organized in three tabs in RStudio: one labeled **environment** that loaded packages and set working directories, a second tab that loaded the data named **load** (the labeling should be simple and sensible to you), and a third tab for **analysis**, which, as you may have guessed, is where most of the work was done. You can imagine extra tabs for data cleaning/munging code, another for visualization code, and maybe others. It is probably a bad idea to go beyond four or five tabs as the management of the tabs alone then becomes more of a task than what they are attempting to alleviate.

In the next section, we will pull in data from GitHub. In the event that you have trouble, we encourage you to go directly to GitHub and download the data (`https://github.com/SocialMediaMininginR`):

1. Load the `sentiment` function located at `https://github.com/SocialMediaMininginR`.
2. Load `https_function` located at `https://raw2.github.com/SocialMediaMininginR/sentiment_function/master/sentiment.R`.

The sentiment function is based on the approach of *Jeffrey Breen*. Some of his material is available on his blog at `http://jeffreybreen.wordpress.com/2011/07/04/twitter-text-mining-r-slides/`, and his complete code is available on GitHub at `https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107`.

The `qdap` package in R has a polarity function based on the work of *Jeffery Breen*. Its goal is quantitative discourse analysis of transcripts containing discourse. The analysis in this first case study is designed to capture discourse by matching and then counting opinionated words based on our lexicons. The counting occurs by whole numbers and does not represent a scale. The `qdap` package ranks sentiment from -1 to 1, akin to text scaling of sorts. Breen's original work is used in the first case study over the `qdap` package since we promote a non-dictionary-based, unsupervised text scaling method in the third case study.

In the first case study, we are interested in determining economic conditions over time. The example captures opinions of experts and local businesses eight times a year. Sentiment analysis will give us insight into the strength and direction of those opinions. Again, to measure the opinions, we need lexicons to match against. Exactly which lexicon you choose to employ may have a direct impact on the analysis of opinionated text, so choose wisely and understand the lexicon landscape. As a rule of thumb, start with general preassembled lexicons; proceed to domain-specific preassembled lexicons; and lastly, advance to empirically constructed lexicons.

Next, we load our lexicons directly from GitHub as follows:

```
# Download positive lexicons from the Social Media Mining Github
account
# note: you will substitute your directory for destination file
locations
# On Windows machines you may have to disregard the method argument
> download.file("https://raw2.github.com/SocialMediaMininginR/neg_
words/master/negative-words.txt", destfile = "/your-localdirectory/
neg_words.txt", method = "curl")
```

```
> download.file("https://raw.github.com/
SocialMediaMininginR/pos_words/master/      LoughranMcDonald_pos.csv",
destfile = "/
your-localdirectory/        LoughranMcDonald_pos.txt",
 method = "curl")

# import positive lexicons from your local directory defined in
earlier step
> pos<- scan(file.path("your-localdirectory",
 'pos_words.txt'), what = 'character',
  comment.char = ';')

# import financial positive lexicon from your local directory defined
in earlier step
> pos_finance<- scan(file.path("your-localdirectory",
'LoughranMcDonald_pos.txt'),
  what = 'character', comment.char = ';')

# combine both files into one
> pos_all<- c(pos, pos_finance)

# Download negative lexicons from Social Media Mining Github account
# note: you will substitute your directory for destination file
locations
# On Windows machines you may have to disregard the method argument >
download.file("https://raw2.github.com/SocialMediaMininginR/neg_words/
master/negative-words.txt", destfile = "/your-localdirectory/pos_
words.txt", method = "curl")

> download.file("https://raw.github.com/
SocialMediaMininginR/neg_words/master/LoughranMcDonald_neg.csv",
destfile = "/your-localdirectory/LoughranMcDonald_neg.txt", method =
"curl")

# import negative lexicons from your local directory defined in
earlier step
> neg<- scan(file.path("/your-localdirectory/
", 'neg_words.txt'),
what = 'character',       comment.char = ';')

# import financial negative lexicon from your local directory defined
in earlier step
> neg_finance<- scan(file.path("/your-localdirectory/",
```

```
'LoughranMcDonald_neg.txt'),
  what = 'character', comment.char = ';')
# combine both files into one
> neg_all<- c(neg, neg_finance)

# Import Beige Book data from Github and create a new data frame.
# *Important* You have three options when ingesting Beige Book data.
  # beigebook_summary.csv is three years of data (2011 - 2013)
  # bb_full.csv is sixteen years of data (1996 - 2011)
  # BB_96_2013.csv is eighteen years of data (1996 - 2013)
# The example below uses beigebook_summary and bb_full
   # Feel free to ingest what you wish or try all three
   # Outputs will look different depending on the file you chose
> download.file("https://raw.github.com/
SocialMediaMininginR/beigebook/master/beigebook_summary.csv", destfile
= "/
your-localdirectory/BB.csv", method = "curl")

> BB <- read.csv("/your-localdirectory/BB.csv")
```

We now have data (*Beige Book*) and both of the lexicons, general and domain-specific, loaded into our R session as well as our sentiment function. Thus, we can begin some exploratory analysis to better understand the data. By using `colnames` on our `data.frame (BB)`, we identify the column names of BB. Other operations too give us a more complete examination of the *Beige Book*, such as `class(BB)`, `str(BB)`, `dim(BB)`, and `head(BB)`. An example of using `colnames` is shown as follows:

```
> colnames(BB)
[1] "year""month""text"
```

We can also check for missing data (year ~ month) using *Hadley Wickham*'s `reshape` package. We can see that there seems to be some systematic missing data, notably that May (5) and December (12) are missing data in all three years of data collection as shown in the following example:

```
> cast(BB, year ~ month, length)

year   1 2 3 4 6 7 8 9 10 11
1 2011 1 0 1 1 1 1 0 1  1  1
2 2012 1 1 0 1 1 1 1 0  1  1
3 2013 1 0 1 1 1 1 0 0  0  0
```

In the real world, data analysis is dirty. Consequently, most of your time will be spent on cleaning the data. Identifying missing data is central to that pursuit. The `is.na()` function in R helps identify missing data. Regular expressions too are pretty handy and aid in pattern matching by finding and replacing data. If you are unfamiliar with regular expressions, we suggest that you learn more; RegexOne has a great regular expression tutorial (`www.regexone.com`), and Debuggex Beta has a helpful debugger (`https://www.debuggex.com/`). An example of regular expressions is given as follows:

```
> bad <- is.na(BB)
# create a new object "bad" that will hold missing data, in this case
from BB.
> BB[bad]
# return all missing elements

character(0)
# returns zero missing elements. Alternately, adding !before "bad"
# will return all good elements.

# regular expressions help us clean our data
# gsub is a function of the R package grep and replaces content that
matches our search
# gsub substitutes punctuation (must be surrounded by another set of
square brackets)
# when used in a regular expression with a space â€˜ â€˜
> BB$text<- gsub('[[:punct:]]', ' ', BB$text)
# gsub substitutes character classes that do not give an output such
as feed, backspace and tabspaces with a space ' '.
> BB$text<- gsub('[[:cntrl:]]', ' ', BB$text)
# gsub substitutes numerical values with digits of one or greater with
a space ' '.
> BB$text<- gsub('\\d+', ' ', BB$text)
# we are going to simplify our data frame and keep the clean text as
well as keep both
# year and a concatenated version of year/ month/day and will format
the latter.
> BB.text <- as.data.frame(BB$text)
> BB.text$year<- BB$year
> BB.text$Date <- as.Date( paste(BB$year, BB$month, BB$day, sep = "-"
)  , format =   "%Y-%m-%d" )
> BB.text$Date <- strptime(as.character(BB.text$Date), "%Y-%m-%d")
> colnames(BB.text) <- c("text", "year", "date")
> colnames(BB.text)
[1] "text" "year" "date"
```

To perform a more complete exploration and refine our analysis, we may also create a corpus via `VectorSource`, which is present in the `tm` package. It is quite useful and can create a corpus from a character vector. A corpus is a collection of text documents. Our goal here is to revisit our data frame to perform our sentiment analysis, but in the meantime we require a more comprehensive understanding of our data, especially to augment our lexicons. In order to do so, we will work with our data as a corpus and term-document matrix using the `tm` package.

For example, we can create a corpus by combining two character strings into the `example_docs` object and convert it using `VectorSource` into a corpus as follows:

```
example_docs<- c("this is an useful example", "augmented by another
useful example")
> example_docs
[1] "this is an useful example""augmented by another useful example"

> class(example_docs)
[1] "character"

> example_corpus<- Corpus(VectorSource(example_docs))
> example_corpus
A corpus with 2 text documents
```

We can perform much of the same cleaning of the data using the `tm` package, but our data then needs to be in a corpus, whereas regular expressions work on character vectors as shown in the following code:

```
> install.packages("tm")
> require(tm)
> bb_corpus<- Corpus(VectorSource(BB.text))
# tm_map allows transformation to a corpora.
# getTransformations() shows us what transformations are available via
the tm_map function
> getTransformations()
  "as.PlainTextDocument" "removeNumbers"        "removePunctuation"
"removeWords"          "stemDocument" "stripWhitespace"
> bb_corpus<- tm_map(bb_corpus, tolower)
> View(inspect(bb_corpus))


# before cleaning:
"The manufacturing sector continued to recover across all Districts."
(2011,1)
# after cleaning:
"the manufacturing sector continued to recover across all districts"
(2011,1)
```

Stemming is rather useful for reducing words down to their core element or stem, as we show in the Naive Bayes and IRT examples. An example of stemming for the words stemming and stems would be stem—effectively dropping the -ing and -s suffixes, as shown in the following code:

```
# stemming can be done easily
# we just need the SnowballC package
> install.packages("SnowballC")
> require(SnowballC)
> bb.text_stm<- tm_map(bb_corpus, stemDocument)
```

When exploring our corpus, it is often important to accentuate the signal and reduce noise. We can accomplish this by removing frequently used words (such as *the*) commonly known as stop words. These commonly used words often have information value at or very close to zero.

We will use a standard list of stop words and augment this further with words specific to our corpus. The list of stop words started with a simple list generated by reading a few of the reports, but was expanded based on some text mining explained later. Again, the goal is eliminating words that lack discriminatory power. The cause for eliminating city names is due to their frequency of use as shown in the following example:

```
# Standard stopwords such as the "SMART" list can be found in the tm
package.
> stnd.stopwords<- stopwords("SMART")
> head(stnd.stopwords)
> length(stnd.stopwords)
[1] 571

# the standard stopwords are useful starting points but we may want to
# add corpus-specific words
# the words below have been added as a consequence of exploring BB
# from subsequent steps
> bb.stopwords<- c(stnd.stopwords, "district", "districts",
"reported", "noted", "city", "cited",   "activity", "contacts",
"chicago", "dallas", "kansas", "san", "richmond", "francisco",
"cleveland", "atlanta", "sales", "boston", "york", "philadelphia",
"minneapolis", "louis",   "services","year", "levels", " louis")
```

The `bb.stopwords` list is a combination of `stnd.stopwords` and our custom list discussed earlier. You can certainly imagine another scenario where these city names are kept and words associated with city names are examined. For the following analysis, however, they were dropped:

```
> length(bb.stopwords)
[1] 596

# additional cleaning to eliminate words that lack discriminatory
power.
# bb.tf will be used as a control for the creation of our term-
document matrix.
> bb.tf <- list(weighting = weightTf, stopwords  = bb.stopwords,
  removePunctuation = TRUE,
  tolower = TRUE,
  minWordLength = 4,
  removeNumbers = TRUE)
```

A common approach in text mining is to create a term-document matrix from a corpus. In the `tm` package, the `TermDocumentMatrix` and `DocumentTermMatrix` classes (depending on whether you want terms as rows and documents as columns, or vice versa) employ sparse matrices for corpora as shown in the following code:

```
# create a term-document matrix
> bb_tdm<- TermDocumentMatrix(bb_corpus, control = bb.tf)

> dim(bb_tdm)
[1] 1515   21
> bb_tdm
  A term-document matrix (1515 terms, 21 documents)

  Non-/sparse entries: 5441/26374
  Sparsity           : 83%
  Maximal term length: 18
  Weighting: term frequency (tf)

> class(bb_tdm)
[1] "TermDocumentMatrix""simple_triplet_matrix"

# We can get all terms n = 1515
> Terms(bb_tdm)
```

A good exploratory step to get a handle on your dataset is sorting frequent words. This helps to first remove stop words that lack discriminatory power as a consequence of their repeated use.

```
> bb.frequent<- sort(rowSums(as.matrix(bb_tdm)), decreasing = TRUE)

# sum of frequent words
> sum(bb.frequent)
```

```
[1] 8948

# further exploratory data analysis
> bb.frequent[1:30]
  # BEFORE removing stopwords
  chicago        demand        dallaskansas           san
  248              245               244           236
220
  richmond francisco         sales     cleveland atlanta
  218              217                  210       201
198
  boston york philadelphia minneapolis louis
  186              185     173                154
140
  increased        growth      services    conditions        prices
  133                108            101            98
92
  mixed     continued        strong          home      manufacturing
  87            84                  71                68            68
  loan        steady         firms       construction      remained
  66            65              64            61
61

> bb.frequent[1:30]
  # AFTER removing stopwords
  demand     increased        growth    conditions        prices
mixed
  252            133                110           102
94               88
  continued        strong          loan        manufacturing
reports          home
  85                  72                 70            69
69                68
  steady     construction         firms        report      remained
consumer
  67            66                       65             64
61                 58
  increases        hiring      increase    production   residential
retail
  58                 57            57             57
56                  56
```

The word demand is a prominent noun, as are `growth`, `conditions`, and `prices`. Prominent adjectives include `mixed`, `strong`, and `steady`, while prominent verbs include `increased`, `continued`, `remained`, and `hiring`. It is too soon to determine the general direction of opinion relating to the economy based on this decontextualized information, but it does help us determine the nature of our corpus.

Finding the most frequent words (for example, *n*, where *n* is the minimum frequency) will help us build out our positive and negative library of words in order to refine our analyses and learn more about our corpus. This is the lexicon approach. A word such as `strong` would appear to be a good predictor of positive sentiment. Exploring the corpus itself can allow our lexicon to grow inductively, allowing us to augment our domain-specific dictionary or build one from a general purpose dictionary. The `any` function returns logical vectors if at least one of the values is true. Using the results from the frequent words, we can begin to test for the presence of words from within our corpus and our lexicons as follows:

```
# look at terms with a minimum frequency
> findFreqTerms(bb_tdm, lowfreq = 60)
  [1] "conditions""construction""continued""demand""firms"
  [6] "growth""home""increased""loan""manufacturing"
  [11] "mixed""prices""remained""report""reports"
  [16] "steady""strong"
```

Additionally, we could augment this even further by using a dictionary (for example, `http://www.merriam-webster.com`) to find words to add to our lexicon. The Thesaurus website is a good choice as it gives many relevant matches (`http://thesaurus.com/browse/increase`) and suggests hike, development, expansion, raise, and surge. These words may be useful. Also useful are their antonyms, which may be used in our negative lexicon. Words such as decrease, drop, shrinkage, and reduction may all prove to be helpful—none of which were included in the default lexicons nor in our manual additions to them. An example of using positive and negative words is shown as follows:

```
# Let us add some of these positive words:
> pos.words<- c(pos_all, "spend", "buy", "earn", "hike", "increase",
"increases",   "development", "expansion", "raise", "surge", "add",
"added", "advanced", "advances",   "boom", "boosted", "boosting",
"waxed",  "upbeat", "surge")

# And add the negative ones:
> neg.words = c(neg_all, "earn", "shortfall", "weak", "fell",
"decreases", "decreases",   "decreased", "contraction", "cutback",
"cuts", "drop", "shrinkage", "reduction",   "abated", "cautious",
"caution", "damped", "waned", "undermine", "unfavorable",   "soft",
"softening", "soften", "softer", "sluggish", "slowed", "slowdown",
"slower",   "recession")
> any(pos.words == "strong")
[1] TRUE
# TRUE is returned. Meaning, "strong" is already in our lexicon.
> any(pos.words == "increases")
[1] FALSE
# FALSE is returned.
# Meaning, "increases" is not already in our lexicon.
```

The word "increases" is not in our lexicon. The implications about the economy and its direction make this word potentially useful. Certainly, it could be associated with increases in unemployment; however, after reading a couple of the mentions of "increases", it seems a better predictor of positive sentiment. It seems the Federal Bank uses "decreases" to suggest negative direction.

We may want to find associations (that is, terms with correlations greater than `0.5`) or correlations with various keywords, such as `demand`. This exploratory process can be used to augment our dictionary and also the contextualized local relationships of our data. We can get a general sense about the interaction between nouns (n) and verbs (v), such as the interaction between `demand` (n) and `hiring` (v) as well as `material` (n) and `building` (v) in the following example:

```
# interestingly, demand is associated with "weak"
> findAssocs(bb_tdm, "demand", 0.5)
  makers        season      products          weak          wood
years
  0.73            0.69          0.65               0.64
0.63          0.63
  livestock      category       pointed      december    electronic
feeding
  0.62             0.60            0.60           0.59
0.59              0.59
  november         power          snow    consistent      exceeded
manufacturers
  0.59                 0.59             0.59          0.57
0.57          0.57

# "increased" is associated with "materials", "hiring" and "building"
> findAssocs(bb_tdm, "increased", 0.5)
  availability    materials    corporate      finding    qualified
hiring
  0.75               0.75             0.68              0.65
0.65         0.63
  selective       yields    purchases     building      dealers
side
  0.58             0.56         0.55             0.54            0.54
0.51

# "growth" is associated with "slowdown" and "reductions"
> findAssocs(bb_tdm, "growth", 0.5)
  slowdownipo     reductions        capital         driven
  0.63                  0.59           0.59            0.55
0.55
```

```
  inputs semiconductors       supplier    contributed         months
  0.55           0.55                    0.55            0.54
0.54
  restrained            tax
  0.54                  0.52
```

Another potential step in our exploration of the data is to make a word cloud, which is a graphic that depicts common words in a corpus by displaying their relative frequencies as relative sizes. Word clouds give a sense of diction within our corpus. We utilize the `wordcloud` function from the `wordcloud` package to generate the useful graphic that is shown after the following code:

```
# Remove sparse terms from term document matrix with
# a numeric value of .95; representing the maximal allowed sparsity.
> BB.95 <- removeSparseTerms(bb_tdm, .95)

# Here we are sorting and counting the row sums of BB.95
> BB.rsums <- sort(rowSums(as.matrix(BB.95)), decreasing=TRUE)

# We will need to create a data frame with the words and their
frequencies.
> BBdf.rsums <- data.frame(word=names(BB.rsums), freq=BB.rsums)
> colnames(BBdf.rsums)
# [1] "word" "freq"

# Install RColorBrewer for coloring our wordcloud
> install.packages("RColorBrewer")
> require(RColorBrewer)

# RColorBrewer creates nice looking color palettes
# Create a palette, blue to green, and name it palette using brewer.
pal
> palette <- brewer.pal(9, "BuGn")
> palette <- palette[-(1:2)]
> install.packages("wordcloud")
> require(wordcloud)

# Create a png and define where it will be saved and named
> png(filename="your/file/location/name.png")
# Create a wordcloud and define the words and their frequencies as
well as how those word sizes will scale.
> bb_wordcloud <- wordcloud(BBdf.rsums$word, BBdf.rsums$freq,
> scale=c(7,.2), min.freq=4, max.words=200,
  random.order=FALSE, colors=palette)
```

```
# dev.off will complete the plot and save the png
> dev.off()
```

Word cloud

As you can see in the previous screenshot, **demand** and **prices** are central. Within the field of economics, demand has a nuanced and important meaning. Demand offers insight into the willingness to buy goods or a service.

You can imagine businesses and government alike spending a great deal of effort trying to understand the quantum of demand that exists within the public sector. Understanding this incorrectly or incompletely will result in incorrectly estimating the impact of government programs or, from a private sector perspective, will result in the loss of money or unrealized gains.

We are now in a position to run our data frame against the `score.sentiment` function. We will show results for the three and sixteen year datasets. Both datasets and the code for both analyses are located on GitHub:

```
# using our score.sentiment function on BB.text$text against pos.words
and neg.words
```

```
# progress = 'text' is useful for monitoring scoring of large
documents
# keep date and year since they are dropped in the score.sentiment
output
> BB.keeps <- BB.text[,c("date", "year")]
# run score.sentiment on our text field using pos.words and neg.words
> BB.score<- score.sentiment(BB.text$text, pos.words, neg.words,
.progress = 'text')
# add back BB.keeps to BB.score
> BB.sentiment <- cbind(BB.keeps, BB.score)
# colnames(BB.sentiment shows that we kept "text", "date", and "year"
field as well as the # new column "score"
> colnames(BB.sentiment)
[1] "date"   "year"  "score"    "text"
```

By examining `BB.sentiment$score` (the three year dataset), we discover a mean of 33. In other words, most scores are already above zero, suggesting that the sentiment is positive, but thereby making interpretability difficult. To improve interpretability, we mean-center our data and shift our midpoint value from 33 to zero. The new, empirically adjusted center may be interpreted as an empirically neutral midpoint. The histograms shown after the following code display both raw scores and centered scores:

```
# calculate mean from raw score
> BB.sentiment$mean <- mean(BB.sentiment$score)
# calculate sum and store it in BB.sum
> BB.sum <- BB.sentiment$score
# center the data by subtracting BB.sum from BB.sentiment$mean
> BB.sentiment$centered <- BB.sum - BB.sentiment$mean
# we can label observations above and below the centered values with 1
# and code N/A values with 0
> BB.sentiment$pos[BB.sentiment$centered>0] <- 1
> BB.sentiment$neg[BB.sentiment$centered<0] <- 1
> BB.sentiment[is.na(BB.sentiment)] <- 0
# we can then sum the values to get a sense of how balanced our data.
> sum(BB.sentiment$pos)

[1] 673
> sum(BB.sentiment$neg)
[1] 683
# we can create a histogram of raw score and centered score to see the
# impact of mean centering
```

```
> BB.hist <- hist(BB.sentiment$score, main="Sentiment Histogram",
xlab="Score", ylab="Frequency")
> BB.hist <- hist(BB.sentiment$centered, main="Sentiment Histogram",
xlab="Score", ylab="Frequency")
```

Histogram of BB.sentiment$score and BBsentiment$meancenter

Some of the upcoming plots will use a package named `ggplot2`, created by *Hadley Wickham*. Though difficult to master, `ggplot2` offers some rather elegant and powerful graphing. This package is different from the package used for plotting in *Chapter 2*, *Getting Started with R*, and is shown here to offer diversity in the ways in which you may plot graphics. There are some rather good resources available if you are interested in learning more about `ggplot2`, but *The Grammar of Graphics* by *Leland Wilkinson* may be the most comprehensive:

```
# install and load ggplot2
install.packages("ggplot2")
require(ggplot2)
# using the results from the function to score our documents we create
# a boxplot to examine the distribution of opinion relating to
# economic conditions the labeling assumes here that you imported
# summary file of three years
> BB.boxplot<- ggplot(BB.sentiment, aes(x = BB.sentiment$year,
  y = BB.sentiment$centered, group = BB.text$year))+
  > geom_boxplot(aes(fill = BB.sentiment$year),
  outlier.colour = "black", outlier.shape = 16, outlier.size = 2)
```

```
# add labels to our boxplot using xlab ("Year"),
ylab("Sentiment(Centered)"), and ggtitle            # ("Economic
Sentiment - Beige Book (2011-2013)")
> BB.boxplot<- BB.boxplot + xlab("Year") + ylab("Sentiment
(Centered)") +
  ggtitle("Economic Sentiment - Beige Book (2011-2013)")
# draw boxplot
BB.boxplot
```

BB.boxplot

This simple example exposes some interesting insight into the economic conditions in the United States as reflected by the *Beige Book*. The x axis shows year and the y axis reflects mean-centered sentiment. Immediately, we can see that both 2011 and 2013 were below the mean sentiment over the entire document space (with the mean of 2013 below that of 2011). The boxplot shows us that while 2012 was above the mean, it had quite a bit of variability as reflected by the large interquartile range. We can also use this visualization to make comparisons between years—note that 2011 has a higher median value than 2013 but also has lower values as shown by the lower hinge. We can also reduce the data to month-by-year instead of merely year to potentially expose further patterns and increased variability. The following boxplot uses a larger portion of the data and reflects the ups and downs of sentiment over time:

BB.boxplot

The screenshot after the following example shows economic sentiment with recession bars highlighted (2001-2002, 2007-2009):

```
# this code can be used to add the recession bars shown below where
xmin and xmax    # are used to add vertical columns to our plot.
```

```
> rect2001 <- data.frame (
  xmin=2001, xmax=2002, ymin=-Inf, ymax=Inf)
> rect2007 <- data.frame (
  xmin=2007, xmax=2009, ymin=-Inf, ymax=Inf)

# ggplot is an R package used for advanced plotting.
> BB.boxplot <- ggplot(BB.sentiment, aes(x=BB.sentiment$year, y=BB.
sentiment$centered,    group=BB.sentiment$year))
> BB.boxplot <- BB.boxplot + geom_boxplot(outlier.colour = "black",
     outlier.shape = 16, outlier.size = 2)
> BB.boxplot <- BB.boxplot + geom_rect(data=rect2001, aes(xmin=xmin,
     xmax=xmax, ymin=-Inf, ymax=+Inf), fill='pink', alpha=0.2,
inherit.aes = FALSE)
> BB.boxplot <- BB.boxplot + geom_rect(data=rect2007, aes(xmin=xmin,
     xmax=xmax, ymin=-Inf, ymax=+Inf), fill='pink', alpha=0.2,
inherit.aes = FALSE)
> BB.boxplot <- BB.boxplot + xlab("Date") + ylab("Sentiment
     (Centered)") + ggtitle("Economic Sentiment - Beige Book (1996-
2010)")
> BB.boxplot
```

Boxplots with recession bars

Case study 2 – Naive Bayes classifier

In the previous chapter, we described how Naive Bayes is a type of classifier, that is, a statistical model designed to estimate group membership of observations. If we have a sufficient amount of training data, we can use it to train or learn a statistical model that we can subsequently use to estimate the sentiment of other, unlabeled observations. The key assumption underlying this technique is that at least some words are used with different frequencies by those with positive and negative sentiments towards a particular target. This section walks through the implementation of Naive Bayes for sentiment classification.

For demonstrative purposes, we have scraped about 4,000 tweets using the methods set out in *Chapter 3*, *Mining Twitter with R*. About half include the hashtag `#prolife`, and the other half include the hashtag `#prochoice`. It seems likely that these tweets satisfy the earlier assumption; that is, tweets using each of these opposing hashtags likely use different words and phrases to a different extent. The following code sets up a data frame of the tweets and appends the variable "hash" (as in hashtag) to the data frame, where 1 denotes the use of `#prochoice` and 0 denotes the use of `#prolife`. As this is a supervised method, we use the hashtags as labels for the supervised learning process. The results of the model will allow us to categorize unlabeled tweets in future.

Suppose that all the tweets are in a list called `abortion_tweets`, as would be output by the `searchTwitter` function introduced in *Chapter 3*, *Mining Twitter with R*. Also, suppose that we've created a vector named `hash`, which is a set of 1's repeating for the number of prochoice tweets concatenated with a vector of zeroes as long as the number of prolife tweets, as follows:

```
# generate a data frame from the list of tweets
> require(twitteR)
> twtsdf<- twListToDF(abortion_tweets)

> twtsdf$hash<- hash

# Drop unneeded variables from the data frame
> keeps <- c("text", "id", "retweetCount", "isRetweet", "screenName",
"hash")
> twtsdf<- twtsdf[,keeps]
```

The following loop generates a list of the tweets, where each tweet is broken into a vector of separate words rather than a sentence. It also compiles a list of the usernames, called names. Lastly, we separate and keep the vector of hashtags and name it `outcome` as shown in the following code snippet:

```
> list.vector.words<- list()
> allwords<- NULL

> names<- NULL
  for (i in 1:dim(twtsdf)[1]){
    each.vector<- strsplit(twtsdf$text[i], split="")
    names<- c(names, twtsdf$screenName[i])
    allwords<- c(allwords, each.vector)
    list.vector.words[[i]] <- each.vector
}

> outcome<- twtsdf$hash
```

Next, we use the `tm` package to create a corpus like the one shown in *Chapter 3, Mining Twitter with R*. Also, we'll remove the hashtags, as we are going to treat that information as labels for the supervised learning process as follows:

```
>require(tm)
# make a corpus
> dat.tm <- Corpus(VectorSource(list.vector.words))
# convert all words to lowercase
> dat.tm <- tm_map(dat.tm, tolower)
# remove punctuation
> dat.tm <- tm_map(dat.tm, removePunctuation)
# remove the hashtags
> dat.tm <- tm_map(dat.tm, removeWords, words=c("prochoice"))
> dat.tm <- tm_map(dat.tm, removeWords, words=c("prolife"))
# remove extra white space
> dat.tm <- tm_map(dat.tm, stripWhitespace)
# stem all words
> dat.tm <- tm_map(dat.tm, stemDocument)
```

Next, we create a document-term matrix, with one twist. Instead of using single words, we use bigrams (ordered two-word pairs). Thus, instead of breaking the sentence "See spot run" into three single words, we make two bigrams, "see spot" and "spot run." This helps us capture more nuance than single words, which helps deal with extreme sentiments and negation as follows:

```
# create a bigram tokenizer using the RWeka package
> require(RWeka)
> BigramTokenizer<- function(x) NGramTokenizer(x, Weka_control(min =
2, max = 2))
# create the document-term matrix
> datmat<- DocumentTermMatrix(dat.tm, control = list(tokenize =
BigramTokenizer))
> dat<- as.matrix(datmat)
# Add user names as rownames to matrix
> rownames(dat) <- names
```

Only bigrams that are used by a sufficient number of authors are useful. Thus, we need a bit of code to remove bigrams that are uncommon. The following lines create a vector of the column sums (that is, the number of times each bigram is used) and then creates a table of those column sums:

```
> word.usage<- colSums(dat)
> table(word.usage)
```

There is no hard-and-fast rule about the lower limit on the number or proportion of uses of a bigram necessary to keep it. We recommend starting out with a low number, such as 9, and working upwards from there as shown in the following code:

```
# first, set all values in the matrix that are greater than 1 to 1
> dat[dat>1] <- 1
> threshold <- 9  # set a threshold
> tokeep <- which(word.usage>threshold)
# find which column sums are above the threshold
# keep all rows, and only columns with sums greater than the threshold
> dat.out<- dat[,tokeep]
```

The last processing step is to drop users who use a very small number of bigrams. The logic is the same as when we dropped the least common bigrams: users who use different words from all the other users are hard to model. Again, there is a bit of art here. We recommend only keeping users who used at least two bigrams, though you may want to increase this if the documents you are using are larger than tweets, as shown in the following code:

```
# Drop users with few words....
# find how many zeroes are in each row
> num.zero <- rowSums(dat.out==0)

# explore data by making a table; can inform choice of cutoff
> table(num.zero)
# the number of columns of the document bigram matrix
> num_cols <- dim(dat.out)[2]
# users must have used this many bigrams to scale
> cutoff <- 2
# create a list of authors to keep
> authors_tokeep <- which(num.zero <(num_cols-cutoff))
# keep only users with 2 bigrams
> dat.drop <- dat.out[authors_tokeep,]
# similarly, drop those users from the vector of hashtags
> outcome <- outcome[authors_tokeep]
```

Finally, we are ready to implement the model. To do so, we'll load the `e1071` package, a general-purpose data-mining package. Then, we set up the data so that it is back in the data frame format, with the `outcome` variable, `hash`, turned into a factor. Factors are data storage types that are good for holding integer-valued variables, as shown in the following code:

```
> require(e1071)
# append the outcome to the first column of dat.drop
> myDat <- cbind(outcome, dat.drop)
# turn the doc-term matrix into a data frame
> myDat <- as.data.frame(myDat)
# turn the outcome variable (first column) into a factor
> myDat[,1] <- as.factor(myDat[,1])
```

Finally, a single line of code implements the model. We will save the model as an object called `NBmod`. The first argument to the `NaiveBayes` function lists the predictors, while the second argument gives the `outcome` variable. We use a trick to capture all of the columns of `myDat` except the first one; using a negative number means "include all but this item", as shown in the following code:

```
# run the model; save the results to an object
> NBmod<- naiveBayes(myDat[,-1], myDat[,1])
```

We should expect our model to perform well on the data on which it was trained, or "in sample". To get a sense of how our model performed, we can make a confusion matrix that compares actual values of the `outcome` variable to predicted values as follows:

```
# generate a vector of predictions
# arguments: estimated model, predictors, outcome to predict
> NBpredictions <- predict(NBmod, myDat[,-1])
# pull out the actual outcomes
> actual<- myDat[,1]
# make the confusion matrix
> table(NBpredictions, actual, dnn=list("predicted", "actual"))

actual
predicted        0        1
        0          339     24
        1          617    825
```

Elements in this table on the main diagonal (the top-left and bottom-right cells) are correctly predicted. The table shows that, in the sample, our **percent correctly predicted** (**PCP**) is about 65 percent. The degree to which our model is accurate will be a function of several parameters. First, the more training data we use, the more accurate our model will be (in sample). Second, the model's accuracy will increase with the divergence in the word-use patterns between our two sentiment groups. Lastly, the larger the documents included in this type of analysis, the better the accuracy of the model. Thus, this model is a bit tenuous for Twitter data and only achieves a modest in-sample accuracy.

The point of this type of model is not to check its accuracy on existing sentiment data. Rather, we want to use this model to predict the sentiment of unlabeled observations. One simple way to accomplish this is to preprocess the unlabeled data along with the labeled data. Then, estimate the model on only the labeled data. Finally, use the trained model to predict the values of the unlabeled observations. To simulate this, suppose we had preprocessed some data as we did earlier but with 500 `#prolife` and 500 `#prochoice` tweets and additionally 100 tweets with the hashtag `#abortion`. Then, we could estimate the model and predict the values of the 100 unlabeled tweets with the following code:

```
# run the model on the 1000 labelled instances
> NBmod<- naiveBayes(myDat[1:1000,-1], myDat[1:1000,1])
# predict outcomes for the last 100 unlabeled instances
> NBpredictions<- predict(NBmod, myDat[1001:1100,-1])
# make a table of the predictions
> table(NBpredictions)

actual
predicted       0         1
        0          24        0
        1          27       49
```

Interestingly, this model predicts better on the test set than on the training set (73 percent accuracy). This is a bit of an anomaly; generally, we should not expect test results to be stronger than training accuracies, unless, by some chance, the test data is more well behaved than the training data. Overall, the Naive Bayes classifier is a useful tool for estimating sentiment valence. It is quick to estimate and has reasonable accuracy. However, as we saw in this example, it requires training data with binary scores already assigned.

Case study 3 – IRT models for unsupervised sentiment scaling

The theoretical underpinnings of IRT models were set out in the previous chapter. Here, we briefly review them before demonstrating how to implement this class of models. However, readers should note that this class of model is cutting-edge, to the point of being considered experimental.

IRT models for text analysis start with the strong assumption that texts (or authors thereof) lie along a continuum, and that this continuum directly affects their word choices in a monotonic way such that if word use is likely at one end of the spectrum, it is unlikely at the other. These assumptions are somewhat restrictive; we can only scale texts that deal with moderately narrow topics and that are subject to word choice differences. Furthermore, it is important that the sentiment continuum be that underlying continuum; else, the model will estimate whatever continuum underlies the data. A good example would be the debate about the Affordable Care Act, wherein liberals occasionally refer to it by name, while conservatives are more likely to refer to it as Obamacare. Note that for IRT methods to function, these differences only have to be probabilistic (that is, some liberals certainly call it Obamacare some of the time), and these word choice differences have to be consistent across a number of terms, not just one.

As an example, we can use the same Twitter data set out in the section on Naive Bayes. We should suspect that these tweets meet the assumptions made earlier, namely, there is a single underlying continuum of sentiment about a single moderately narrow topic, and that that continuum likely affects the word choice of each author.

The preprocessing steps are very similar to those of the Naive Bayes application discussed in the previous section. To begin, take the same initial steps: collect tweets with one (or more) `searchTwitter` call(s); turn them into a data frame called `tweetsdf`; and append an `outcome` variable called `hash` to this data frame. Again, these steps are exactly the same as the ones discussed earlier, and are thus not repeated. One difference is that, before continuing, we will drop retweets from the data frame. The reason is that including a large number of retweets affects scaling negatively. It is a best practice to drop the retweets and then just give all tweets with the same text the same scale after running the IRT model. The following line of code in English reads: put into `twtsdf` all of the rows in `twtsdf` where the variable `isRetweet` is `FALSE`.

```
> twtsdf<- twtsdf[twtsdf$isRetweet == FALSE,]
```

Next, we continue following the Naive Bayes preprocessing steps by dropping the unnecessary variables from the data frame. After that, we execute the `for` loop that generates our list of vectors of words, create a corpus, and turn it into a document-term matrix, just as done earlier. We normalize all cells to one, and then drop infrequent bigrams, as in the Naïve Bayes example.

The second difference in the preprocessing steps for IRT is that we need to aggregate all of the tweets by user, that is, collapse the document-term matrix such that there is only one row per user instead of one row per tweet. This step follows from the fact that IRT models assume that all of an author's word choices are affected by his or her position on the underlying continuum. Thus, we aggregate all of their tweets together as shown in the following code snippet:

```
# pull out list of words
> words<- colnames(dat.out)
# aggregate by rowname (i.e. twitter user name), and sum rows with the
same user name
> dat.agg<- aggregate(dat.out, list(rownames(dat.out)), sum)
# aggregating makes a variable called Group.1; turn this back into the
matrix rowname
> names<- dat.agg$Group.1
> dat.agg<- as.matrix(dat.agg[,2:dim(dat.agg)[2]])
# set cells greater than 1 back to 1
> dat.agg[dat.agg>1] <- 1
> rownames(dat.agg) <- names
```

This bit of code similarly aggregates our hashtag vector, which we will use for model checking later:

```
> outcomes<- as.matrix(twtsdf$hash)
> rownames(outcomes) <- rownames(dat.out)
> outcomes.agg <- aggregate(outcomes, list(rownames(outcomes)), mean)
> hashscores<- round(outcomes.agg$V1)
```

The final preprocessing step mimics that of Naive Bayes: we drop users who use a very small number of bigrams. In Twitter examples, we find that only keeping users who employ at least four key bigrams makes a good choice, though there is no hard-and-fast rule about this:

```
> num.zero <- rowSums(dat.agg==0)

# explore data by making a table; can inform choice of cutoff
> table(num.zero)
```

```
# the number of columns of the document bigram matrix
> num_cols <- dim(dat.out)[2]
# users must have used this many bigrams to scale
> cutoff <- 4
# create a list of authors to keep
> authors_tokeep <- which(num.zero <(num_cols-cutoff))
# keep only users with 2 bigrams
> dat.drop <- dat.out[authors_tokeep,]
# similarly, drop those users from the vector of hashtags
> outcome <- outcome[authors_tokeep]
```

After all that processing, we are ready to implement the model. The package we employ, `pscl`, was designed for political science uses, hence the mentions of legislators, votes, and roll-calls. The `rc` function sets up the data matrix for estimation, while the `ideal` function estimates the logistic model described in the previous chapter as follows:

```
> require(pscl)
> rc <- rollcall(data=dat.drop)     # sets up the data
# executes the model (this may take several minutes)
> jmod <- ideal(rc,store.item=T)
```

Now we have a new object named `jmod`. The `xbar` variable captures the scale positions of every author for whom we had enough data to scale. One thing we may want is a list of all of the scaled positions of all users as shown in the following code snippet:

```
> scaled.positions<- data.frame(jmod$xbar)     # make a new data frame
of the scale
> rownames(scaled.positions) <- rownames(dat.drop)
# make the rownames sensible
> colnames(scaled.positions) <- "scale"
# give the single variable a good name
> head(scaled.positions)
# list the first few scaled positions
# note: these are not real user names
scale
exUser1          0.739675498
exUser2          -0.099127391
exUser3          0.466915634
```

Another way to look at this data is to draw a histogram of all of the scaled positions:

```
> hist(jmod$xbar, main= "Scaled Positions of Twitter Users", xlab=
"relative position")
```

The histogram shows the overall distribution of positions in the data. Some care needs to be taken in interpreting these values. Foremost, these values are relative to one another. Put in a different way, these numbers describe distances between authors but not compared to any true values. This means that zero, rather than meaning neutral, probably means something closer to central. Similarly, the score only captures the fact that an author at zero is to the left of authors at 0.1, and to the right of authors at -0.1. This means that, had we *only* scaled the prochoice tweets, we still would have had a distribution centered at zero, even though all of the tweets being scaled would be to the right of center.

A second caution concerns authors scaled at or near zero. These may be centrist authors; however, authors
that are difficult to scale also end up in the center. This may be because they use few bigrams or because they use conflicting bigrams (some to the left and some to the right). The point is to note that zero may not mean center; it may also mean "I'm not sure".

Additionally, the model has no sense of left or right; it just assigns numbers to authors. To figure out whether authors with scores of +1 are liberals or conservatives, we need to examine the tweets of a couple of authors from each end. First, identify a handful of authors with large, positive scale values, and then go back to the original data frame and read their tweets. This should give you a sense of which end of the scale is which. Additionally, it should give you an idea of whether the underlying scale estimated corresponds to the latent dimension you had hoped to capture, that is, sentiment. The following is a list of positive authors:

```
> scaled.positions[scaled.positions$scale> 0.9,]
# generates a list of large, positive authors
# these are just examples, not real users
> MrTweeter             0.9137
> AnotherTweeter         0.9442
```

Then, go back to the original data frame and pull up the tweet(s) from those users. This should tell you if the people on the positive end of this scale are on the left or right. You should check several of these from both ends. People on the same end of the scale should have the same views, generally. If not, the model may have failed or pulled out an underlying continuum other than the one you were looking for!

```
> twtsdf[twtsdf$id== "MrTweeter",]
1   This would be MrTweeter's original tweet and hashtag.
```

Also, we may be interested in the `difficulty` and `discrimination` parameters of each bigram. We can get a sense of both by plotting one against the other. The `jmod$betabar[,2]` parameter is the difficulty parameter, and `jmod$betabar[,1]` is the discrimination parameter.

```
> plot(jmod$betabar[,2], jmod$betabar[,1],
    xlab="difficulty", ylab="discrimination", main="Bigrams")
```

On this graph, each point represents a bigram. The x axis represents how hard it is for a user to use a bigram; essentially, this is a measure of the bigram's rarity. The y axis plots each bigram's discrimination, that is, the extent to which it is more likely to be used by those on one side of the scale or the other. For instance, bigrams with large, positive discrimination parameters are likely to be used by those on the right-hand side of the scale and unlikely to be used by those on the left-hand side. The sign determines left/right, and the magnitude represents how strong the effect is. Bigrams with near-zero discrimination parameters are equally used by authors on all parts of the scale. Examining this graph, we see that most bigrams are not discriminating between sides of the scale. Additionally, there is a strong correlation between difficulty and discrimination; bigrams that are used frequently do not discriminate much, whereas more infrequently used bigrams discriminate better. This is a classic pattern in scaling applications; absence of this type of *flying V* pattern is evidence that scaling has failed or that the model has picked up an underlying continuum that is bizarre or nonsensical.

If you want to get a sense of the most discriminating bigrams, you can generate a list with the following code, which uses the `plyr` package for its convenient `arrange()` function:

```
# identify which words have large discrimination parameters
# abs() returns the absolute value
> t <- which(abs(jmod$betabar[,1]) >1)
> twords<- colnames(dat.drop)[t]
> tnums<- jmod$betabar[,1][t]
# make a data frame of the discriminating words
> bigwords<- data.frame(twords, tnums)
> bigwords<- arrange(bigwords,desc(bigwords$tnums))
```

Finally, since we know what side of the abortion issue all of our users were on, we can plot them and color code the hashtag they used (this is why we saved the hashtags during data processing). In real, unsupervised sentiment analysis settings, you would not have this information, but since we do, we can use it to check the accuracy of the model as follows:

```
# make a vector, o, of the order of the scale positions
> o <- order(jmod$xbar)
# plot the scale against an arbitrary y value; zero to number of users
# color code each point according to its hashtag with the 'col'
parameter
> plot(jmod$xbar[o], seq(1:length(jmod$xbar)), col=-outcome[o]+3)
```

We've put the results of this code in the following graph, and also added a text callout of a couple of the tweets and a legend for readability. The code for the extras is simple and can be found on this book's web page on the Packt Publishing website:

The x axis here represents the scale position, while the y axis is arbitrary. The green points are those tweets that used `#prolife`, whereas the red ones used `#prochoice`. We can see, generally, that most of the red points are to the left of zero, while most of the green points are to the right of zero. Thus, the model seems to have scaled most of these tweets accurately. However, the model has a tough time discriminating authors near the middle of the scale, which is common in scaling applications. Additionally, note the group of mis-characterized tweets in the upper-left corner. On further examination, these are users who tweeted prochoice-related phrases, but used them ironically. The model, not recognizing irony, incorrectly put these tweets with the prochoice tweets. This is a common hurdle in text analysis and represents a fruitful area for future research.

Summary

Social media mining is technical, and is occasionally as much an art as a science. We hope to have given readers a leg up on the subject by walking them through these extended case studies, each involving a different style of analysis. However, gaining a mastery of these techniques will certainly entail readers scraping their own text-based data, and applying the tools demonstrated here. As you do so, we hope you remain careful about the pitfalls listed in *Chapter 4*, *Potentials and Pitfalls of Social Media Data*, and proceed with due thought and care.

Conclusions and Next Steps

Social media has become ubiquitous, as has the understanding that harnessing it is crucial to measuring the sentiments of an increasingly plugged-in population. Ignoring this information while acknowledging its presence, whether for businesses or civic purpose, constitutes an informal logical fallacy. Those businesses, politicians, social movements, and researchers who choose to ignore this data do so at their own peril and to their own detriment.

Final thoughts

People are highly opinionated and compelled to share with others. The advent of the social web has given them a tremendous new venue to do so, and explains, in part, the explosive growth in text data. These personal opinions are valuable; rather than being fleeting or trivial, they are both predictive of and caused by individual intentions. Over the last decade, scholars and practitioners of social media mining have developed techniques to measure and thus glean insights from textual opinion data. These tools are crucial, especially since much text data does not come with easily quantifiable opinions such as the availability of stars, likes, or thumbs-ups that can be easily counted.

With the expanding availability of data and the increasing sophistication and usability of text data mining tools, social media mining is more accessible to a wide array of practitioners. An increasing number of social scientists, businesses, politicians, and media outlets put themselves at a stark disadvantage by ignoring this source of insight. Social scientists are now able to tap the sentiments of larger and harder-to-reach populations. Industries can now obtain granular reactions to products and adjust their offerings accordingly based on large samples, rather than a few poignant complaints. Politicians can gauge the desires of their constituents, the polarity of issues, and the effectiveness of their campaigns. Meanwhile, media outlets can not only track the interest in their stories, but also more easily take the pulse of the population on which they report.

The opinionated and plugged-in nature of people has driven the growth of text data; however, it is techniques like the ones outlined in this book that help add measurable value to that text. Techniques like the ones outlined in this book help explain the value of that text. Up until the last 40 years, these opinions were a part of small networks and shared mainly with first order neighbors. The data was hard if not impossible to collect, and few methods existed to examine the data. Nowadays, however, the data is available and the methods are maturing. We hope this book gives practitioners and scholars a quality entry point to the study and use of opinion mining techniques, and that it invests them with the knowledge necessary to explore textual data in a systematic and rigorous manner.

An expanding field

The field of sentiment analysis is growing quickly. Google Scholar reports nearly 70,000 articles including the words sentiment analysis published from 2012 to 2013; three and a half times as many as were published in the preceding annum. Assuredly, this growth is due in part to the wide array of purposes to which people apply sentiment analysis and text mining. Beyond the number of applications is the power of social media data; one study (Brynjolfsson, 2011) found that technology investments of 179 large publicly-traded firms that adopted data-driven decision making have output and productivity five to six percent higher than what would otherwise be expected; in an era when data provably matters for high-stakes decisions, being able to make new quantities measurable will certainly be a benefit. What would be the impact on your organization if you could improve output and productivity by a mere five to six percent? Or, what would be the social impact of a policy that was five to six percent more effective?

Before setting you off to conduct your own research, however, we again implore you to take care with your analyses, for they often come with consequences. Numbers, and especially measurements, tend to get reified in unhealthy ways such as IQ and BMI have been in the last two decades. Knowing this, it is especially important that you think hard about your measurements and deliver them with appropriate caveats. Consider carefully the population of people to which you can extrapolate; the users of social media are often young and urban. Furthermore, beware of contexts in which people have incentives to provide or promote biased opinions such as when writing about rivals. Lastly, carefully consider a new trend in paid opinion writing, wherein companies or advertisers hire tech-savvy authors to spam social media with favorable information about them. Detecting these strategic actors may in fact be an interesting research area in the field over the next decade. However, if you chose to analyze social data, the need will remain to be vigilant of its pitfalls. That said, the promise of social media data is great, if it can be leveraged with care.

Further reading

For more information, you can refer to the following books:

- *Taming Text: How to Find, Organize, and Manipulate It*, *Ingersoll*, *Morton*, and *Farris*: This is a pragmatic volume that covers exactly what its title promises. As data scientists, it has been noted that we spend around 80 percent of our time arranging our data for analysis. This book can help to trim that figure, allowing you to spend more time analyzing and interpreting, rather than data munging.
- *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*, *Matthew Russell*: This is an excellent resource for practitioners wanting to learn how to get their hands on a wide array of social and web data. The book also features a companion piece that deals with analyzing social data. Readers should note that this pair of books utilizes Python rather than R.
- *Speech and Language Processing*, *Jurafski* and *Martin*: This book provides a slightly more advanced and fairly wide review of many techniques and technologies applied to language. This book is slightly more technical than *Social Media Mining in R*; it is likely inappropriate for those unfamiliar with optimization.
- *The Handbook of Computational Linguistics and Natural Language Processing*, *Clark*, *Fox*, and *Lappin*: This is another fairly broad book that covers many NLP topics, but with more of a focus on machine learning than theoretical statistics. It is suitable for graduate-level students and researchers.
- *Machine Learning with R*, *Brett Lanz*: This is a nice volume that covers a wide array of machine-learning algorithms. While not aimed specifically at social media, readers will find several of the techniques applicable.
- *Finding Groups in Data: An Introduction to Cluster Analysis*, *Kaufman* and *Rousseeuw*: This is a well-crafted book that makes an excellent first read on clustering techniques, which we mentioned in *Chapter 3*, *Mining Twitter with R*. The book covers several clustering techniques in an intuitive and non-technical manner.
- *A First Course in Statistical Programming with R*, *Braun* and *Murdoch*: This is an excellent resource for the world's most popular and fastest-growing statistical programming language. This book goes into great detail about programming constructs and graphical parameters, all with an eye towards building a student's competencies in applied statistics.
- *Designing Social Research*, *Normal Blaikie*: This is an excellent first book on research design. It carefully walks readers through knowledge areas such as theory testing, measurement, and inference.

Bibliography

- *Semisupervised Learning for Computational Linguistics*, *Steven Abney*, *Chapman and Hall*, 2007
- *Sentiment Analysis and Opinion Mining*, *Morgan & Claypool Publishers*, *Bing Liu*, May 2012
- *Designing Social Research (2nd Edition)*, *Polity*, November 2009
- *A First Course in Statistical Programming with R*, *Braun.W.* and *Murdoch.D.*, *Cambridge University Press*, 2007
- *Strength in Numbers: How Does Data-Driven Decision Making Affect Firm Performance?*, *Erik Brynjolfsson*, *ICIS 2011 Proceedings, Paper 13*, 2011
- *The Handbook of Computational Linguistics and Natural Language Processing*, *Clark.A.*, *Fox.C.*, and *Lappin.S.*, *Wiley-Blackwell*, October 2012
- *The Statistical Analysis of Roll Call Data*, *Clinton.J.*, *Jackman.S.*, and *Rivers.D.*, *American Political Science Review*, May 2004
- *Taming Text: How to Find, Organize, and Manipulate It*, *Ingersoll.G.*, *Morton.T.*, and *Farris.A.*, *Manning Publications*, January 2013
- *Speech and Language Processing (2nd Edition)*, *Jurafsky.D.* and *Martin.J.*, *Pearson Prentice Hall*, May 2008
- *Finding Groups in Data: An Introduction to Cluster Analysis*, *Kaufman.L.* and *Rousseeuw.P.*, *Wiley-Interscience*, March 2005
- *Machine Learning with R*, *Lantz* and *Brett*, *Packt Publishing*, October 2013
- *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More (2nd Edition)*, *Matthew Russell*, *O'Reilly Media*, 2013
- *Consilience: The Unity of Knowledge*, *E. O. Wilson*, *Vintage*, March 1999
- About Me The Long Tail, n.p, n.d, Web, 10 Jan. 2014
- what's new on reddit: Reflections on the Recent Boston Crisis, blog.reddit, n.p, n.d, Web. 10 Jan. 2014
- *Fake Reviews: Amazon's Rotten Core*, *Forbes*, n.p, n.d, Web. 10 Jan. 2014
- *Gartner Blog Network*, n.p, n.d, Web, 10 Jan. 2014
- *Google Politics: Four Screens To Victory*, *Independent Voter Network IVN.us*, n.p, n.d, Web, 10 Jan. 2014
- *IBM big data platform Bringing big data to the Enterprise*, n.p, Web, 10 Jan. 2014
- *Key Facts-Facebook's latest news, announcements and media resources*, n.p, n.d, Web, 10 Jan. 2014

- *Open Innovation: Open Innovation Community*, n.p, n.d, Web, 10 Jan. 2014
- *Taboada et al. Lexicon-based Methods for Sentiment Analysis. Computational Linguistics*, Vol. 37, No 2, 2011
- *Twitter turns six*, Twitter blogs, n.p, n.d, Web, 10 Jan. 2014
- U.S. Distrust in Media Hits New High, Gallup.Com-Daily News, Polls, Public Opinion on Politics, Economy, Wellbeing, and World, n.p, n.d, Web, 10 Jan. 2014
- Welcome to the United Nations: It's Your World, n.p, n.d, Web, 10 Jan. 2014
- `https://vimeo.com/11742135`
- Social web, Wikipedia, the free encyclopedia, 16 Nov. 2013, Web, 10 Jan. 2014
- 50 Cent Party, Wikipedia, the free encyclopedia, 28 Nov. 2013, Web, 10 Jan. 2014
- You and Your Research Computer Science, U.Va. Engineering, graduate and undergraduate computer science programs, Master's degree, PhD, Virginia, n.p, n.d, Web, 10 Jan. 2014

Bibliography

This course is packaged keeping your journey in mind. It includes content from the following Packt products:

- *Learning Data mining with R by Bater Makhabel*
- *R Data Mining Blueprints by Pradeepta Mishra*
- *Social Media Mining with R by Nathan Danneman and Richard Heimann*

Index

A

Amazon Web Services (AWS.tools) 16
Anscombe's quartet 23
application programming interface (API) 28
arguments 17
assignment operator 18

B

bccmpls 35
benefits, R 13
Big Data
  about 4, 5
  scope 6

C

carrot (>) operator 17
case studies, social media mining
  considerations 59, 60
  IRT models 85
  lexicon based sentiment 61
  Naive Bayes classifier 80
corpus 32

D

data frames
  creating 20-22
data mining 2
dendrogram 35
Distributed Storage and List (dsl) 16
dist variable 22
document-term matrix
  building 33
Domain Specific Languages. *See* **DSLs**
Dropbox 36

F

Facebook
  about 3
FAQs, R 14
files
  importing 20-22
function 17

G

Git 36
GitHub 36
GNU (GNU's Not Unix) 14
GoogleDrive 36
Google Reviews 51
graphical user interface (GUI) 16

H

HadoopInteractiVE (hive) 16
Hadoop Steaming (HadoopSteaming) 16
help function 18
hierarchical agglomerative clustering 34
honest signals 6-8
human sensors 6-8

I

information bounce 4
installation, R 16
integrated development environment (IDE) 16
Internet Movie Database (IMDB) 53
IRT models 57

IRT models case study 85-92
Item Response Theory (IRT) 56, 57

L

lexicon based sentiment case study 61-78
lexicon-based sentiment classification 53-55
lisarosie 35
logical operators 18

M

modifiable areal unit problem (MAUP) 43

N

naive Bayes classifier 55, 56
Naive Bayes classifier case study 80-84
natural language processing (NLP) 1
nontraditional social data
  versus traditional social data 40, 41

O

OAuthFactory function 29
operators, R
  arithmetic 17
  assignment 18
  logical 18
opinion mining. *See* **also sentiment analysis 37**
ordinary least squares (OLS) 21

P

plot() function 23
preliminary analyses 32-35
ProjectTemplate
  URL 36

Q

qualitative approaches
  using 10

R

R
  additional resources 24
  benefits 13
  community 15
  consequences 15
  FAQs 14
  installing 16
  URL, for FAQs 16
R code
  URL, for info 24
  writing, tips 24
references 95
registerTwitterOAuth function 29
RStudio
  about 14, 16
  URL 16
ruleflow. *See* **Drools Flow**

S

Scherers typology of emotions
  about 50
  attitudes 50
  emotion 50
  interpersonal stance 50
  mood 50
  personality traits 50
sentiment 50
sentiment analysis 94
  about 1
sentiment polarity
  measuring 51, 53
seq() function 19
sequences
  about 19
  example 19
social media
  sentiments, measuring 38, 39
social media data
  inferential challenges 41-44
  measurements 41-43
  overview 40
  pitfalls 37, 38
  potentials 37
social media mining 93
  about 1, 47
  case studies 59
  concepts 47, 48
  content, identifying 51

content, retrieving 51
lexicon-based sentiment 53
naive Bayes classifiers 55
URL, for code 2
with sentiment analysis 1, 2
Stack Overflow 15
state of communication section 2-4
stop words 32

T

Text Mining Distributed Corpus Plug-In (tm.plug.dc) 16
traditional social data
versus nontraditional social data 40, 41
traditional social science data
versus, social media data 48, 49
tweet 51
tweets 32
Twitter
about 4, 27
URL, for developer account 28
Twitter data 51
about 37
need for 27
obtaining 28-31
twitteR package 28
Twittersphere 51

U

Universal Description and Discovery Information. *See* **UDDI**

V

vectors
about 19
example 19
visualization 22, 23

W

weak ties 27
weblog. *See* **blog**
WordCloud package 32
World Wide Web (WWW) 3

Y

Yelp 51

Packt

Thank you for buying

R: Mining Spatial, Text, Web, and Social Media Data

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at `www.packtpub.com`.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

Bibliography

This course is packaged keeping your journey in mind. It includes content from the following Packt products:

- *Learning Data mining with R by Bater Makhabel*
- *R Data Mining Blueprints by Pradeepta Mishra*
- *Social Media Mining with R by Nathan Danneman and Richard Heimann*