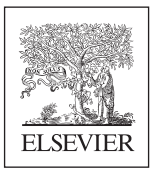
Data Science

Concepts and Practice

Second Edition

Vijay Kotu

Bala Deshpande



MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER

CHAPTER 1

Introduction

Data science is a collection of techniques used to extract value from data. It has become an essential tool for any organization that collects, stores, and processes data as part of its operations. Data science techniques rely on finding useful patterns, connections, and relationships within data. Being a buzzword, there is a wide variety of definitions and criteria for what constitutes data science. Data science is also commonly referred to as knowledge discovery, machine learning, predictive analytics, and data mining. However, each term has a slightly different connotation depending on the context. In this chapter, we attempt to provide a general overview of data science and point out its important features, purpose, taxonomy, and methods.

In spite of the present growth and popularity, the underlying methods of data science are decades if not centuries old. Engineers and scientists have been using predictive models since the beginning of nineteenth century. Humans have always been forward-looking creatures and predictive sciences are manifestations of this curiosity. So, who uses data science today? Almost every organization and business. Sure, we didn't call the methods that are now under data science as "*Data Science.*" The use of the term *science* in data science indicates that the methods are evidence based, and are built on empirical knowledge, more specifically historical observations.

As the ability to collect, store, and process data has increased, in line with Moore's Law - which implies that computing hardware capabilities double every two years, data science has found increasing applications in many diverse fields. Just decades ago, building a production quality regression model took about several dozen hours (Parr Rud, 2001). Technology has come a long way. Today, sophisticated machine learning models can be run, involving hundreds of predictors with millions of records in a matter of a few seconds on a laptop computer.

The process involved in data science, however, has not changed since those early days and is not likely to change much in the foreseeable future. To get meaningful results from any data, a major effort preparing, cleaning,

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00001-0

scrubbing, or standardizing the data is still required, before the learning algorithms can begin to crunch them. But what may change is the automation available to do this. While today this process is iterative and requires analysts' awareness of the best practices, soon smart automation may be deployed. This will allow the focus to be put on the most important aspect of data science: interpreting the results of the analysis in order to make decisions. This will also increase the reach of data science to a wider audience.

When it comes to the data science techniques, are there a core set of procedures and principles one must master? It turns out that a vast majority of data science practitioners today use a handful of very powerful techniques to accomplish their objectives: decision trees, regression models, deep learning, and clustering (Rexer, 2013). A majority of the data science activity can be accomplished using relatively few techniques. However, as with all 80/20 rules, the long tail, which is made up of a large number of specialized techniques, is where the value lies, and depending on what is needed, the best approach may be a relatively obscure technique or a combination of several not so commonly used procedures. Thus, it will pay off to learn data science and its methods in a systematic way, and that is what is covered in these chapters. But, first, how are the often-used terms artificial intelligence (AI), machine learning, and data science explained?

1.1 AI, MACHINE LEARNING, AND DATA SCIENCE

Artificial intelligence, Machine learning, and data science are all related to each other. Unsurprisingly, they are often used interchangeably and conflated with each other in popular media and business communication. However, all of these three fields are distinct depending on the context. Fig. 1.1 shows the relationship between artificial intelligence, machine learning, and data science.

Artificial intelligence is about giving machines the capability of mimicking human behavior, particularly cognitive functions. Examples would be: facial recognition, automated driving, sorting mail based on postal code. In some cases, machines have far exceeded human capabilities (sorting thousands of postal mails in seconds) and in other cases we have barely scratched the surface (search "artificial stupidity"). There are quite a range of techniques that fall under artificial intelligence: linguistics, natural language processing, decision science, bias, vision, robotics, planning, etc. Learning is an important part of human capability. In fact, many other living organisms can learn.

Machine learning can either be considered a sub-field or one of the tools of artificial intelligence, is providing machines with the capability of learning

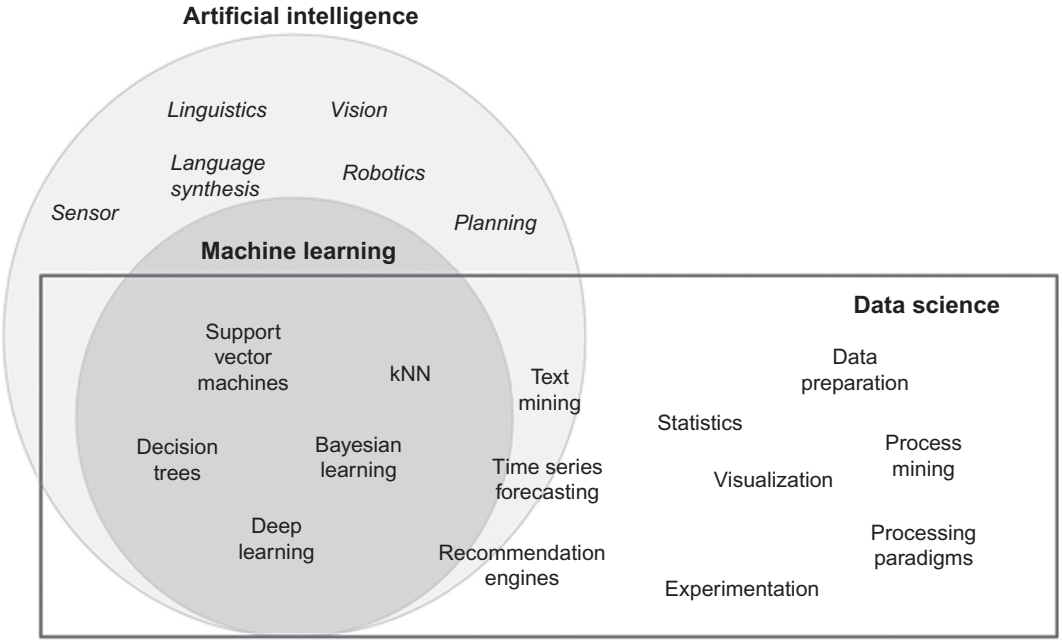


FIGURE 1.1
Artificial intelligence, machine learning, and data science.

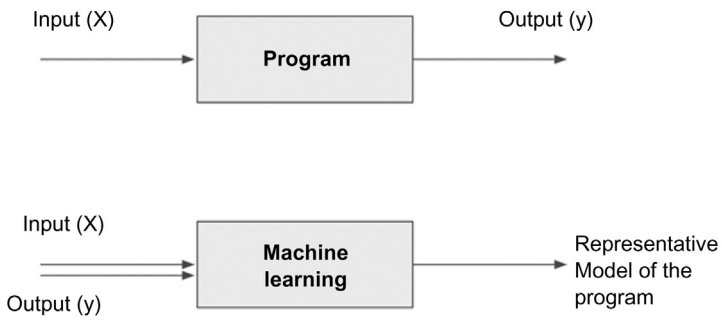


FIGURE 1.2
Traditional program and machine learning.

from experience. Experience for machines comes in the form of data. Data that is used to teach machines is called training data. Machine learning turns the traditional programing model upside down (Fig. 1.2). A program, a set of instructions to a computer, transforms input signals into output signals using predetermined rules and relationships. Machine learning algorithms,

also called "learners", take both the known input and output (training data) to figure out a model for the program which converts input to output. For example, many organizations like social media platforms, review sites, or forums are required to moderate posts and remove abusive content. How can machines be taught to automate the removal of abusive content? The machines need to be shown examples of both abusive and non-abusive posts with a clear indication of which one is abusive. The learners will generalize a pattern based on certain words or sequences of words in order to conclude whether the overall post is abusive or not. The model can take the form of a set of "if—-then" rules. Once the data science rules or model is developed, machines can start categorizing the disposition of any new posts.

Data science is the business application of machine learning, artificial intelligence, and other quantitative fields like statistics, visualization, and mathematics. It is an interdisciplinary field that extracts value from data. In the context of how data science is used today, it relies heavily on machine learning and is sometimes called data mining. Examples of data science user cases are: recommendation engines that can recommend movies for a particular user, a fraud alert model that detects fraudulent credit card transactions, find customers who will most likely churn next month, or predict revenue for the next quarter.

1.2 WHAT IS DATA SCIENCE?

Data science starts with *data*, which can range from a simple array of a few numeric observations to a complex matrix of millions of observations with thousands of variables. Data science utilizes certain specialized computational *methods* in order to discover meaningful and useful structures within a dataset. The discipline of data science coexists and is closely associated with a number of related areas such as database systems, data engineering, visualization, data analysis, experimentation, and business intelligence (BI). We can further define data science by investigating some of its key features and motivations.

1.2.1 Extracting Meaningful Patterns

Knowledge discovery in databases is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns or relationships within a dataset in order to make important decisions (Fayyad, Piatetsky-shapiro, & Smyth, 1996). Data science involves inference and iteration of many different hypotheses. One of the key aspects of data science is the process of *generalization* of patterns from a dataset. The generalization should be valid, not just for the dataset used to observe the pattern, but also for new unseen data. Data science is also a process with defined steps, each

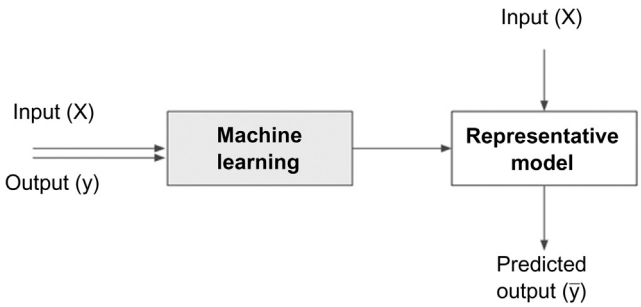


FIGURE 1.3
Data science models.

with a set of tasks. The term *novel* indicates that data science is usually involved in finding previously unknown patterns in data. The ultimate objective of data science is to find potentially useful conclusions that can be acted upon by the users of the analysis.

1.2.2 Building Representative Models

In statistics, a model is the representation of a relationship between variables in a dataset. It describes how one or more variables in the data are related to other variables. Modeling is a process in which a representative abstraction is built from the observed dataset. For example, based on credit score, income level, and requested loan amount, a model can be developed to determine the interest rate of a loan. For this task, previously known observational data including credit score, income level, loan amount, and interest rate are needed. Fig. 1.3 shows the process of generating a model. Once the representative model is created, it can be used to predict the value of the interest rate, based on all the input variables.

Data science is the process of building a representative model that fits the observational data. This model serves two purposes: on the one hand, it predicts the output (interest rate) based on the new and unseen set of input variables (credit score, income level, and loan amount), and on the other hand, the model can be used to understand the relationship between the output variable and all the input variables. For example, does income level really matter in determining the interest rate of a loan? Does income level matter more than credit score? What happens when income levels double or if credit score drops by 10 points? A Model can be used for both predictive and explanatory applications.

1.2.3 Combination of Statistics, Machine Learning, and Computing

In the pursuit of extracting useful and relevant information from large datasets, data science borrows computational techniques from the disciplines of statistics, machine learning, experimentation, and database theories. The algorithms used in data science originate from these disciplines but have since evolved to adopt more diverse techniques such as parallel computing, evolutionary computing, linguistics, and behavioral studies. One of the key ingredients of successful data science is substantial prior knowledge about the data and the business processes that generate the data, known as *subject matter expertise*. Like many quantitative frameworks, data science is an iterative process in which the practitioner gains more information about the patterns and relationships from data in each cycle. Data science also typically operates on large datasets that need to be stored, processed, and computed. This is where database techniques along with parallel and distributed computing techniques play an important role in data science.

1.2.4 Learning Algorithms

We can also define data science as a process of discovering previously unknown patterns in data using *automatic iterative methods*. The application of sophisticated learning algorithms for extracting useful patterns from data differentiates data science from traditional data analysis techniques. Many of these algorithms were developed in the past few decades and are a part of machine learning and artificial intelligence. Some algorithms are based on the foundations of Bayesian probabilistic theories and regression analysis, originating from hundreds of years ago. These iterative algorithms automate the process of searching for an optimal solution for a given data problem. Based on the problem, data science is classified into *tasks* such as classification, association analysis, clustering, and regression. Each data science task uses specific learning algorithms like decision trees, neural networks, *k*-nearest neighbors (*k*-NN), and *k*-means clustering, among others. With increased research on data science, such algorithms are increasing, but a few classic algorithms remain foundational to many data science applications.

1.2.5 Associated Fields

While data science covers a wide set of techniques, applications, and disciplines, there a few associated fields that data science heavily relies on. The techniques used in the steps of a data science process and in conjunction with the term "data science" are:

- *Descriptive statistics*: Computing mean, standard deviation, correlation, and other descriptive statistics, quantify the aggregate structure of a

dataset. This is essential information for understanding any dataset in order to understand the structure of the data and the relationships within the dataset. They are used in the exploration stage of the data science process.

- *Exploratory visualization*: The process of expressing data in visual coordinates enables users to find patterns and relationships in the data and to comprehend large datasets. Similar to descriptive statistics, they are integral in the pre- and post-processing steps in data science.
- *Dimensional slicing*: Online analytical processing (OLAP) applications, which are prevalent in organizations, mainly provide information on the data through dimensional slicing, filtering, and pivoting. OLAP analysis is enabled by a unique database schema design where the data are organized as dimensions (e.g., products, regions, dates) and quantitative facts or measures (e.g., revenue, quantity). With a well-defined database structure, it is easy to slice the yearly revenue by products or combination of region and products. These techniques are extremely useful and may unveil patterns in data (e.g., candy sales decline after Halloween in the United States).
- *Hypothesis testing*: In confirmatory data analysis, experimental data are collected to evaluate whether a hypothesis has enough evidence to be supported or not. There are many types of statistical testing and they have a wide variety of business applications (e.g., A/B testing in marketing). In general, data science is a process where many hypotheses are generated and tested based on observational data. Since the data science algorithms are iterative, solutions can be refined in each step.
- *Data engineering*: Data engineering is the process of sourcing, organizing, assembling, storing, and distributing data for effective analysis and usage. Database engineering, distributed storage, and computing frameworks (e.g., Apache Hadoop, Spark, Kafka), parallel computing, extraction transformation and loading processing, and data warehousing constitute data engineering techniques. Data engineering helps source and prepare for data science learning algorithms.
- *Business intelligence*: Business intelligence helps organizations consume data effectively. It helps query the ad hoc data without the need to write the technical query command or use dashboards or visualizations to communicate the facts and trends. Business intelligence specializes in the secure delivery of information to right roles and the distribution of information at scale. Historical trends are usually reported, but in combination with data science, both the past and the predicted future data can be combined. BI can hold and distribute the results of data science.

1.3 CASE FOR DATA SCIENCE

In the past few decades, a massive accumulation of data has been seen with the advancement of information technology, connected networks, and the businesses it enables. This trend is also coupled with a steep decline in data storage and data processing costs. The applications built on these advancements like online businesses, social networking, and mobile technologies unleash a large amount of complex, heterogeneous data that are waiting to be analyzed. Traditional analysis techniques like dimensional slicing, hypothesis testing, and descriptive statistics can only go so far in information discovery. A paradigm is needed to manage the massive volume of data, explore the inter-relationships of thousands of variables, and deploy machine learning algorithms to deduce optimal insights from datasets. A set of frameworks, tools, and techniques are needed to intelligently assist humans to process all these data and extract valuable information (Piatetsky-Shapiro, Brachman, Khabaza, Kloesgen, & Simoudis, 1996). Data science is one such paradigm that can handle large volumes with multiple attributes and deploy complex algorithms to search for patterns from data. Each key motivation for using data science techniques are explored here.

1.3.1 Volume

The sheer volume of data captured by organizations is exponentially increasing. The rapid decline in storage costs and advancements in capturing every transaction and event, combined with the business need to extract as much leverage as possible using data, creates a strong motivation to store more data than ever. As data become more granular, the need to use large volume data to extract information increases. A rapid increase in the volume of data exposes the limitations of current analysis methodologies. In a few implementations, the time to create generalization models is critical and data volume plays a major part in determining the time frame of development and deployment.

1.3.2 Dimensions

The three characteristics of the Big Data phenomenon are high volume, high velocity, and high variety. The variety of data relates to the multiple types of values (numerical, categorical), formats of data (audio files, video files), and the application of the data (location coordinates, graph data). Every single record or data point contains multiple attributes or variables to provide context for the record. For example, every user record of an ecommerce site can contain attributes such as products viewed, products purchased, user demographics, frequency of purchase, clickstream, etc. Determining the most effective offer for an ecommerce user can involve computing information across

these attributes. Each attribute can be thought of as a dimension in the data space. The user record has multiple attributes and can be visualized in multi-dimensional space. The addition of each dimension increases the complexity of analysis techniques.

A simple linear regression model that has one input dimension is relatively easy to build compared to multiple linear regression models with multiple dimensions. As the dimensional space of data increase, a scalable framework that can work well with multiple data types and multiple attributes is needed. In the case of text mining, a document or article becomes a data point with each unique word as a dimension. Text mining yields a dataset where the number of attributes can range from a few hundred to hundreds of thousands of attributes.

1.3.3 Complex Questions

As more complex data are available for analysis, the complexity of information that needs to get extracted from data is increasing as well. If the natural clusters in a dataset, with hundreds of dimensions, need to be found, then traditional analysis like hypothesis testing techniques cannot be used in a scalable fashion. The machine-learning algorithms need to be leveraged in order to automate searching in the vast search space.

Traditional statistical analysis approaches the data analysis problem by assuming a stochastic model, in order to predict a response variable based on a set of input variables. A linear regression is a classic example of this technique where the parameters of the model are estimated from the data. These hypothesis-driven techniques were highly successful in modeling simple relationships between response and input variables. However, there is a significant need to extract nuggets of information from large, complex datasets, where the use of traditional statistical data analysis techniques is limited (Breiman, 2001)

Machine learning approaches the problem of modeling by trying to find an algorithmic model that can better predict the output from input variables. The algorithms are usually recursive and, in each cycle, estimate the output and "learn" from the predictive errors of the previous steps. This route of modeling greatly assists in exploratory analysis since the approach here is not validating a hypothesis but generating a multitude of hypotheses for a given problem. In the context of the data problems faced today, both techniques need to be deployed. John Tuckey, in his article "We need both exploratory and confirmatory," stresses the importance of both exploratory and confirmatory analysis techniques (Tuckey, 1980). In this book, a range of data science techniques, from traditional statistical modeling techniques like regressions to the modern machine learning algorithms are discussed.

1.4 DATA SCIENCE CLASSIFICATION

Data science problems can be broadly categorized into *supervised* or *unsupervised* learning models. Supervised or directed data science tries to infer a function or relationship based on labeled training data and uses this function to map new unlabeled data. Supervised techniques predict the value of the output variables based on a set of input variables. To do this, a model is developed from a *training* dataset where the values of input and output are previously known. The model generalizes the relationship between the input and output variables and uses it to predict for a dataset where only input variables are known. The output variable that is being predicted is also called a class label or target variable. Supervised data science needs a sufficient number of labeled records to learn the model from the data. Unsupervised or undirected data science uncovers hidden patterns in unlabeled data. In unsupervised data science, there are no output variables to predict. The objective of this class of data science techniques, is to find patterns in data based on the relationship between data points themselves. An application can employ both supervised and unsupervised learners.

Data science problems can also be classified into tasks such as: classification, regression, association analysis, clustering, anomaly detection, recommendation engines, feature selection, time series forecasting, deep learning, and text mining (Fig. 1.4). This book is organized around these data science tasks. An overview is presented in this chapter and an in-depth discussion of the

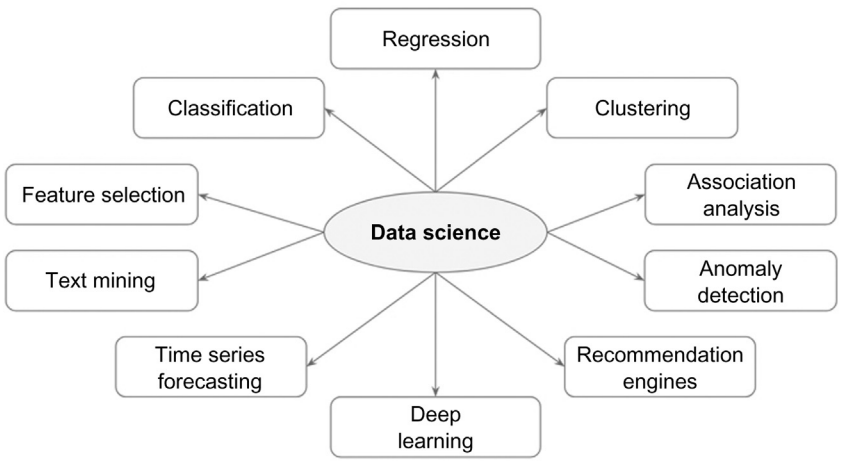


FIGURE 1.4
Data science tasks.

concepts and step-by-step implementations of many important techniques will be provided in the upcoming chapters.

Classification and *regression* techniques predict a target variable based on input variables. The prediction is based on a generalized model built from a previously known dataset. In regression tasks, the output variable is numeric (e.g., the mortgage interest rate on a loan). Classification tasks predict output variables, which are categorical or polynomial (e.g., the yes or no decision to approve a loan). *Deep learning* is a more sophisticated artificial neural network that is increasingly used for classification and regression problems. *Clustering* is the process of identifying the natural groupings in a dataset. For example, clustering is helpful in finding natural clusters in customer datasets, which can be used for market segmentation. Since this is unsupervised data science, it is up to the end user to investigate why these clusters are formed in the data and generalize the uniqueness of each cluster. In retail analytics, it is common to identify pairs of items that are purchased together, so that specific items can be bundled or placed next to each other. This task is called market basket analysis or *association analysis*, which is commonly used in cross selling. *Recommendation engines* are the systems that recommend items to the users based on individual user preference.

Anomaly or outlier detection identifies the data points that are significantly different from other data points in a dataset. Credit card transaction fraud detection is one of the most prolific applications of anomaly detection. *Time series forecasting* is the process of predicting the future value of a variable (e.g., temperature) based on past historical values that may exhibit a trend and seasonality. *Text mining* is a data science application where the input data is text, which can be in the form of documents, messages, emails, or web pages. To aid the data science on text data, the text files are first converted into document vectors where each unique word is an attribute. Once the text file is converted to document vectors, standard data science tasks such as classification, clustering, etc., can be applied. *Feature selection* is a process in which attributes in a dataset are reduced to a few attributes that really matter.

A complete data science application can contain elements of both supervised and unsupervised techniques (Tan et al., 2005). Unsupervised techniques provide an increased understanding of the dataset and hence, are sometimes called descriptive data science. As an example of how both unsupervised and supervised data science can be combined in an application, consider the following scenario. In marketing analytics, clustering can be used to find the natural clusters in customer records. Each customer is assigned a cluster label at the end of the clustering process. A labeled customer dataset can now be used to develop a model that assigns a cluster label for any new customer record with a supervised classification technique.

1.5 DATA SCIENCE ALGORITHMS

An algorithm is a logical step-by-step procedure for solving a problem. In data science, it is the blueprint for how a particular data problem is solved. Many of the learning algorithms are recursive, where a set of steps are repeated many times until a limiting condition is met. Some algorithms also contain a random variable as an input and are aptly called *randomized algorithms*. A classification task can be solved using many different learning algorithms such as decision trees, artificial neural networks, *k*-NN, and even some regression algorithms. The choice of which algorithm to use depends on the type of dataset, objective, structure of the data, presence of outliers, available computational power, number of records, number of attributes, and so on. It is up to the data science practitioner to decide which algorithm (s) to use by evaluating the performance of multiple algorithms. There have been hundreds of algorithms developed in the last few decades to solve data science problems.

Data science algorithms can be implemented by custom-developed computer programs in almost any computer language. This obviously is a time-consuming task. In order to focus the appropriate amount of time on data and algorithms, data science tools or statistical programing tools, like R, RapidMiner, Python, SAS Enterprise Miner, etc., which can implement these algorithms with ease, can be leveraged. These data science tools offer a library of algorithms as functions, which can be interfaced through programming code or configurated through graphical user interfaces. Table 1.1 provides a summary of data science tasks with commonly used algorithmic techniques and example cases.

1.6 ROADMAP FOR THIS BOOK

It's time to explore data science techniques in more detail. The main body of this book presents: the concepts behind each data science algorithm and a practical implementation (or two) for each. The chapters do not have to be read in a sequence. For each algorithm, a general overview is first provided, and then the concepts and logic of the learning algorithm and how it works in plain language are presented. Later, how the algorithm can be implemented using RapidMiner is shown. RapidMiner is a widely known and used software tool for data science (Piatetsky, 2018) and it has been chosen particularly for ease of implementation using GUI, and because it is available to use free of charge, as an open source data science tool. Each chapter is

CHAPTER 2

Data Science Process

The methodical discovery of useful relationships and patterns in data is enabled by a set of iterative activities collectively known as the data science process. The standard data science process involves (1) understanding the problem, (2) preparing the data samples, (3) developing the model, (4) applying the model on a dataset to see how the model may work in the real world, and (5) deploying and maintaining the models. Over the years of evolution of data science practices, different frameworks for the process have been put forward by various academic and commercial bodies. The framework put forward in this chapter is synthesized from a few data science frameworks and is explained using a simple example dataset. This chapter serves as a high-level roadmap to building deployable data science models, and discusses the challenges faced in each step and the pitfalls to avoid.

One of the most popular data science process frameworks is Cross Industry Standard Process for Data Mining (CRISP-DM), which is an acronym for Cross Industry Standard Process for Data Mining. This framework was developed by a consortium of companies involved in data mining (Chapman et al., 2000). The CRISP-DM process is the most widely adopted framework for developing data science solutions. Fig. 2.1 provides a visual overview of the CRISP-DM framework. Other data science frameworks are SEMMA, an acronym for Sample, Explore, Modify, Model, and Assess, developed by the SAS Institute (SAS Institute, 2013); DMAIC, is an acronym for Define, Measure, Analyze, Improve, and Control, used in Six Sigma practice (Kubiak & Benbow, 2005); and the Selection, Preprocessing, Transformation, Data Mining, Interpretation, and Evaluation framework used in the knowledge discovery in databases process (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). All these frameworks exhibit common characteristics, and hence, a generic framework closely resembling the CRISP process will be used. As with any process framework, a data science process recommends the execution of a certain set of tasks to achieve optimal output. However, the process of extracting information and knowledge from the data is *iterative*. The steps within the data science process are not linear and have to undergo many

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00002-2

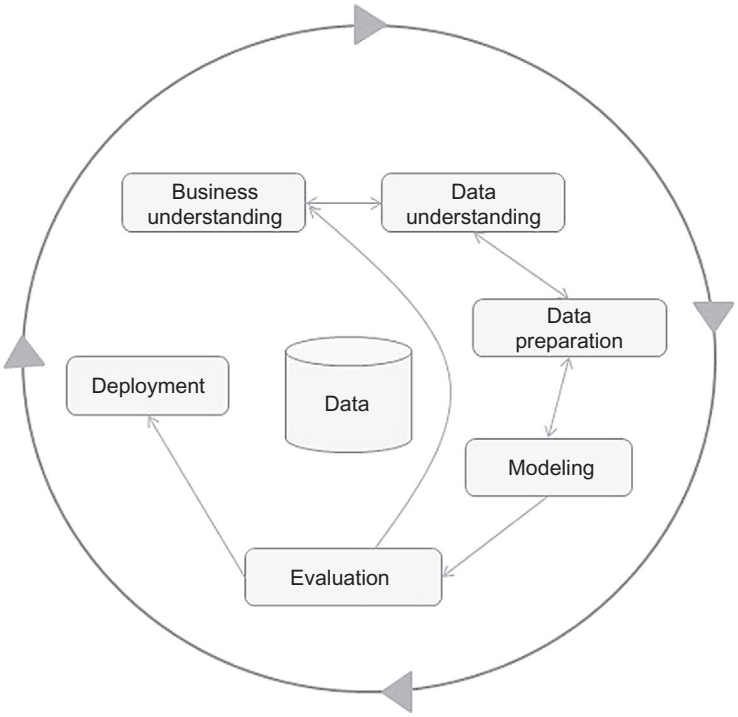


FIGURE 2.1
CRISP data mining framework.

loops, go back and forth between steps, and at times go back to the first step to redefine the data science problem statement.

The data science process presented in Fig. 2.2 is a generic set of steps that is problem, algorithm, and, data science tool agnostic. The fundamental objective of any process that involves data science is to address the analysis question. The problem at hand could be a segmentation of customers, a prediction of climate patterns, or a simple data exploration. The learning algorithm used to solve the business question could be a decision tree, an artificial neural network, or a scatterplot. The software tool to develop and implement the data science algorithm used could be custom coding, RapidMiner, R, Weka, SAS, Oracle Data Miner, Python, etc., (Piatetsky, 2018) to mention a few.

Data science, specifically in the context of big data, has gained importance in the last few years. Perhaps the most visible and discussed part of data science is the third step: *modeling*. It is the process of building representative models that can be inferred from the sample dataset which can be used for either predicting (*predictive modeling*) or describing the underlying pattern in the data (*descriptive or explanatory modeling*). Rightfully so, there is plenty of

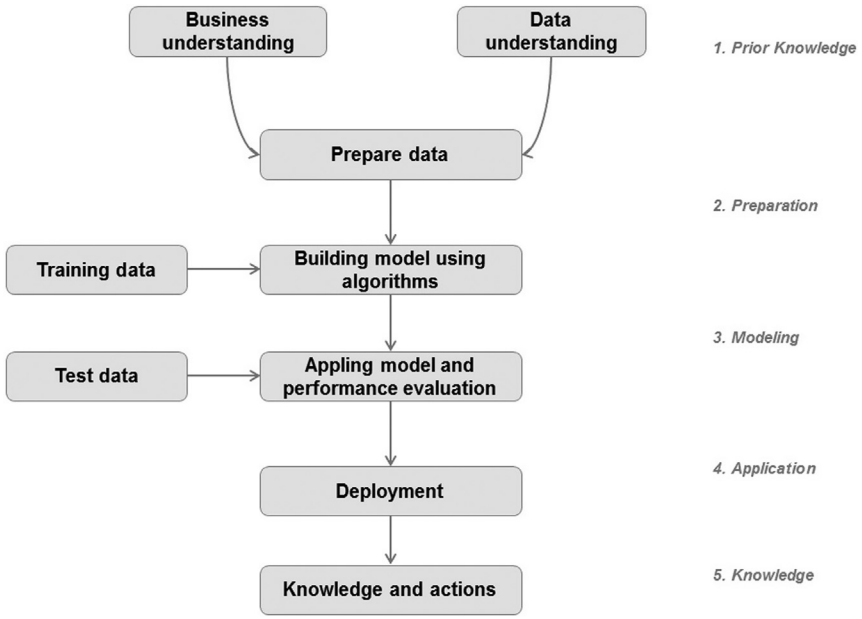


FIGURE 2.2
Data science process.

academic and business research in the modeling step. Most of this book has been dedicated to discussing various algorithms and the quantitative foundations that go with it. However, emphasis should be placed on considering data science as an end-to-end, multi-step, iterative process instead of just a model building step. Seasoned data science practitioners can attest to the fact that the most time-consuming part of the overall data science process is not the model building part, but the preparation of data, followed by data and business understanding. There are many data science tools, both open source and commercial, available on the market that can automate the model building. Asking the right business question, gaining in-depth business understanding, sourcing and preparing the data for the data science task, mitigating implementation considerations, integrating the model into the business process, and, most useful of all, gaining knowledge from the dataset, remain crucial to the success of the data science process. It's time to get started with Step 1: Framing the data science question and understanding the context.

2.1 PRIOR KNOWLEDGE

Prior knowledge refers to information that is already known about a subject. The data science problem doesn't emerge in isolation; it always develops on

top of existing subject matter and contextual information that is already known. The prior knowledge step in the data science process helps to define what problem is being solved, how it fits in the business context, and what data is needed in order to solve the problem.

2.1.1 Objective

The data science process starts with a need for analysis, a question, or a business objective. This is possibly the most important step in the data science process (Shearer, 2000). Without a well-defined statement of the problem, it is impossible to come up with the right dataset and pick the right data science algorithm. As an iterative process, it is common to go back to previous data science process steps, revise the assumptions, approach, and tactics. However, it is imperative to get the first step—the objective of the whole process—right.

The data science process is going to be explained using a hypothetical use case. Take the consumer loan business for example, where a loan is provisioned for individuals against the collateral of assets like a home or car, that is, a mortgage or an auto loan. As many homeowners know, an important component of the loan, for the borrower and the lender, is the interest rate at which the borrower repays the loan on top of the principal. The interest rate on a loan depends on a gamut of variables like the current federal funds rate as determined by the central bank, borrower's credit score, income level, home value, initial down payment amount, current assets and liabilities of the borrower, etc. The key factor here is whether the lender sees enough reward (interest on the loan) against the risk of losing the principal (borrower's default on the loan). In an individual case, the status of default of a loan is Boolean; either one defaults or not, during the period of the loan. But, in a group of tens of thousands of borrowers, the default rate can be found—a continuous numeric variable that indicates the percentage of borrowers who default on their loans. All the variables related to the borrower like credit score, income, current liabilities, etc., are used to assess the default risk in a related group; based on this, the interest rate is determined for a loan. The business objective of this hypothetical case is: *If the interest rate of past borrowers with a range of credit scores is known, can the interest rate for a new borrower be predicted?*

2.1.2 Subject Area

The process of data science uncovers hidden patterns in the dataset by exposing relationships between attributes. But the problem is that it uncovers a lot of patterns. The false or spurious signals are a major problem in the data science process. It is up to the practitioner to sift through the exposed patterns and accept the ones that are valid and relevant to the answer of the objective

question. Hence, it is essential to know the subject matter, the context, and the business process generating the data.

The lending business is one of the oldest, most prevalent, and complex of all the businesses. If the objective is to predict the lending interest rate, then it is important to know how the lending business works, why the prediction matters, what happens after the rate is predicted, what data points can be collected from borrowers, what data points cannot be collected because of the external regulations and the internal policies, what other external factors can affect the interest rate, how to verify the validity of the outcome, and so forth. Understanding current models and business practices lays the foundation and establishes known knowledge. Analysis and mining the data provides the new knowledge that can be built on top of the existing knowledge (Lidwell, Holden, & Butler, 2010).

2.1.3 Data

Similar to the prior knowledge in the subject area, prior knowledge in the data can also be gathered. Understanding how the data is collected, stored, transformed, reported, and used is essential to the data science process. This part of the step surveys all the data available to answer the business question and narrows down the new data that need to be sourced. There are quite a range of factors to consider: quality of the data, quantity of data, availability of data, gaps in data, does lack of data compel the practitioner to change the business question, etc. The objective of this step is to come up with a dataset to answer the business question through the data science process. It is critical to recognize that an inferred model is only as good as the data used to create it.

For the lending example, a sample dataset of ten data points with three attributes has been put together: identifier, credit score, and interest rate. First, some of the terminology used in the data science process are discussed.

- A *dataset* (*example set*) is a collection of data with a defined structure. Table 2.1 shows a dataset. It has a well-defined structure with 10 rows and 3 columns along with the column headers. This structure is also sometimes referred to as a "data frame".
- A *data point* (*record*, *object* or *example*) is a single instance in the dataset. Each row in Table 2.1 is a data point. Each instance contains the same structure as the dataset.
- An *attribute* (*feature*, *input*, *dimension*, *variable*, or *predictor*) is a single property of the dataset. Each column in Table 2.1 is an attribute. Attributes can be numeric, categorical, date-time, text, or Boolean *data types*. In this example, both the credit score and the interest rate are numeric attributes.

Table 2.1 Dataset

Borrower ID	Credit Score	Interest Rate (%)
01	500	7.31
02	600	6.70
03	700	5.95
04	700	6.40
05	800	5.40
06	800	5.70
07	750	5.90
08	550	7.00
09	650	6.50
10	825	5.70

Table 2.2 New Data With Unknown Interest Rate

Borrower ID	Credit Score	Interest Rate
11	625	?

- A *label* (*class label*, *output*, *prediction*, *target*, or *response*) is the special attribute to be predicted based on all the input attributes. In Table 2.1, the interest rate is the output variable.
- *Identifiers* are special attributes that are used for locating or providing context to individual records. For example, common attributes like names, account numbers, and employee ID numbers are identifier attributes. Identifiers are often used as lookup keys to join multiple datasets. They bear no information that is suitable for building data science models and should, thus, be excluded for the actual modeling step. In Table 2.1, the attribute ID is the identifier.

2.1.4 Causation Versus Correlation

Suppose the business question is inverted: *Based on the data in* Table 2.1, *can the credit score of the borrower be predicted based on interest rate?* The answer is yes—but it does not make business sense. From the existing domain expertise, it is known that credit score *influences* the loan interest rate. Predicting credit score based on interest rate inverses the direction of the causal relationship. This question also exposes one of the key aspects of model building. The correlation between the input and output attributes doesn't guarantee causation. Hence, it is important to frame the data science question correctly using the existing domain and data knowledge. In this data science example, the interest rate of the new borrower with an unknown interest rate will be predicted (Table 2.2) based on the pattern learned from known data in Table 2.1.

2.2 DATA PREPARATION

Preparing the dataset to suit a data science task is the most time-consuming part of the process. It is extremely rare that datasets are available in the form required by the data science algorithms. Most of the data science algorithms would require data to be structured in a tabular format with records in the rows and attributes in the columns. If the data is in any other format, the data would need to be transformed by applying pivot, type conversion, join, or transpose functions, etc., to condition the data into the required structure.

2.2.1 Data Exploration

Data preparation starts with an in-depth exploration of the data and gaining a better understanding of the dataset. Data exploration, also known as *exploratory data analysis*, provides a set of simple tools to achieve basic understanding of the data. Data exploration approaches involve computing descriptive statistics and visualization of data. They can expose the structure of the data, the distribution of the values, the presence of extreme values, and highlight the inter-relationships within the dataset. Descriptive statistics like mean, median, mode, standard deviation, and range for each attribute provide an easily readable summary of the key characteristics of the distribution of data. On the other hand, a visual plot of data points provides an instant grasp of all the data points condensed into one chart. Fig. 2.3 shows the scatterplot of credit score vs. loan interest rate and it can be observed that as credit score increases, interest rate decreases.

2.2.2 Data Quality

Data quality is an ongoing concern wherever data is collected, processed, and stored. In the interest rate dataset (Table 2.1), how does one know if the credit score and interest rate data are accurate? What if a credit score has a recorded value of 900 (beyond the theoretical limit) or if there was a data entry error? Errors in data will impact the representativeness of the model. Organizations use data alerts, cleansing, and transformation techniques to improve and manage the quality of the data and store them in companywide repositories called *data warehouses*. Data sourced from well-maintained data warehouses have higher quality, as there are proper controls in place to ensure a level of data accuracy for new and existing data. The data cleansing practices include elimination of duplicate records, quarantining outlier records that exceed the bounds, standardization of attribute values, substitution of missing values, etc. Regardless, it is critical to check the data using data exploration techniques in addition to using prior knowledge of the data and business before building models.

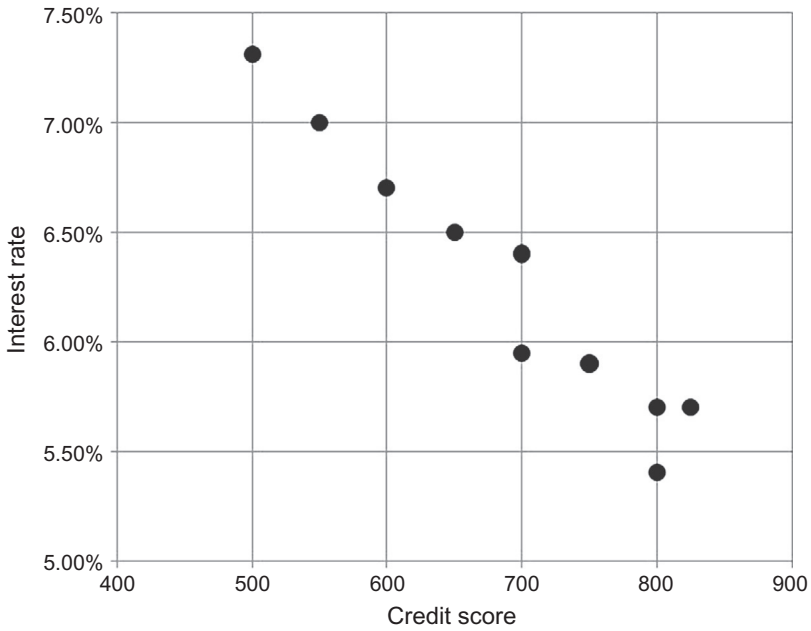


FIGURE 2.3
Scatterplot for interest rate dataset.

2.2.3 Missing Values

One of the most common data quality issues is that some records have missing attribute values. For example, a credit score may be missing in one of the records. There are several different mitigation methods to deal with this problem, but each method has pros and cons. The first step of managing missing values is to understand the reason behind why the values are missing. Tracking the data lineage (provenance) of the data source can lead to the identification of systemic issues during data capture or errors in data transformation. Knowing the source of a missing value will often guide which mitigation methodology to use. The missing value can be substituted with a range of artificial data so that the issue can be managed with marginal impact on the later steps in the data science process. Missing credit score values can be replaced with a credit score derived from the dataset (mean, minimum, or maximum value, depending on the characteristics of the attribute). This method is useful if the missing values occur randomly and the frequency of occurrence is quite rare. Alternatively, to build the representative model, all the data records with missing values or records with poor data quality can be ignored. This method reduces the size of the dataset. Some data science algorithms are good at handling records with missing values, while others expect the data preparation step to handle it before the model is

inferred. For example, k-nearest neighbor (k-NN) algorithm for classification tasks are often robust with missing values. Neural network models for classification tasks do not perform well with missing attributes, and thus, the data preparation step is essential for developing neural network models.

2.2.4 Data Types and Conversion

The attributes in a dataset can be of different types, such as continuous numeric (interest rate), integer numeric (credit score), or categorical. For example, the credit score can be expressed as categorical values (poor, good, excellent) or numeric score. Different data science algorithms impose different restrictions on the attribute data types. In case of linear regression models, the input attributes have to be numeric. If the available data are categorical, they must be converted to continuous numeric attribute. A specific numeric score can be encoded for each category value, such as poor = 400, good = 600, excellent = 700, etc. Similarly, numeric values can be converted to categorical data types by a technique called *binning*, where a range of values are specified for each category, for example, a score between 400 and 500 can be encoded as "low" and so on.

2.2.5 Transformation

In some data science algorithms like k-NN, the input attributes are expected to be numeric and *normalized*, because the algorithm compares the values of different attributes and calculates distance between the data points. Normalization prevents one attribute dominating the distance results because of large values. For example, consider income (expressed in USD, in thousands) and credit score (in hundreds). The distance calculation will always be dominated by slight variations in income. One solution is to convert the range of income and credit score to a more uniform scale from 0 to 1 by normalization. This way, a consistent comparison can be made between the two different attributes with different units.

2.2.6 Outliers

Outliers are anomalies in a given dataset. Outliers may occur because of correct data capture (few people with income in tens of millions) or erroneous data capture (human height as 1.73 cm instead of 1.73 m). Regardless, the presence of outliers needs to be understood and will require special treatments. The purpose of creating a representative model is to generalize a pattern or a relationship within a dataset and the presence of outliers skews the representativeness of the inferred model. Detecting outliers may be the primary purpose of some data science applications, like fraud or intrusion detection.

2.2.7 Feature Selection

The example dataset shown in Table 2.1 has one *attribute* or *feature*—the credit score—and one *label*—the interest rate. In practice, many data science problems involve a dataset with hundreds to thousands of attributes. In text mining applications, every distinct word in a document forms a distinct attribute in the dataset. Not all the attributes are equally important or useful in predicting the target. The presence of some attributes might be counterproductive. Some of the attributes may be highly correlated with each other, like annual income and taxes paid. A large number of attributes in the dataset significantly increases the complexity of a model and may degrade the performance of the model due to the *curse of dimensionality*. In general, the presence of more detailed information is desired in data science because discovering nuggets of a pattern in the data is one of the attractions of using data science techniques. But, as the number of dimensions in the data increase, data becomes sparse in high-dimensional space. This condition degrades the reliability of the models, especially in the case of clustering and classification (Tan, Steinbach, & Kumar, 2005).

Reducing the number of attributes, without significant loss in the performance of the model, is called feature selection. It leads to a more simplified model and helps to synthesize a more effective explanation of the model.

2.2.8 Data Sampling

Sampling is a process of selecting a subset of records as a representation of the original dataset for use in data analysis or modeling. The sample data serve as a representative of the original dataset with similar properties, such as a similar mean. Sampling reduces the amount of data that need to be processed and speeds up the build process of the modeling. In most cases, to gain insights, extract the information, and to build representative predictive models it is sufficient to work with samples. Theoretically, the error introduced by sampling impacts the relevancy of the model, but their benefits far outweigh the risks.

In the build process for data science applications, it is necessary to segment the datasets into training and test samples. The training dataset is sampled from the original dataset using simple sampling or class label specific sampling. Consider the example cases for predicting anomalies in a dataset (e.g., predicting fraudulent credit card transactions). The objective of anomaly detection is to classify the outliers in the data. These are rare events and often the dataset does not have enough examples of the outlier class. *Stratified sampling* is a process of sampling where each class is equally represented in the sample; this allows the model to focus on the difference between the patterns of each class that is, normal and outlier records. In classification applications,

sampling is used create multiple base models, each developed using a different set of sampled training datasets. These base models are used to build one meta model, called the *ensemble model*, where the error rate is improved when compared to that of the base models.

2.3 MODELING

A model is the abstract representation of the data and the relationships in a given dataset. A simple rule of thumb like "*mortgage interest rate reduces with increase in credit score*" is a model; although there is not enough quantitative information to use in a production scenario, it provides directional information by abstracting the relationship between credit score and interest rate.

There are a few hundred data science algorithms in use today, derived from statistics, machine learning, pattern recognition, and the body of knowledge related to computer science. Fortunately, there are many viable commercial and open source data science tools on the market to automate the execution of these learning algorithms. As a data science practitioner, it is sufficient to an overview of the learning algorithm, how it works, and determining what parameters need to be configured based on the understanding of the business and data. Classification and regression tasks are predictive techniques because they predict an outcome variable based on one or more input variables. Predictive algorithms require a prior known dataset to learn the model. Fig. 2.4 shows the steps in the modeling phase of predictive data science. Association analysis and clustering are descriptive data science techniques where there is no target variable to predict; hence, there is no test dataset. However, both predictive and descriptive models have an evaluation step.

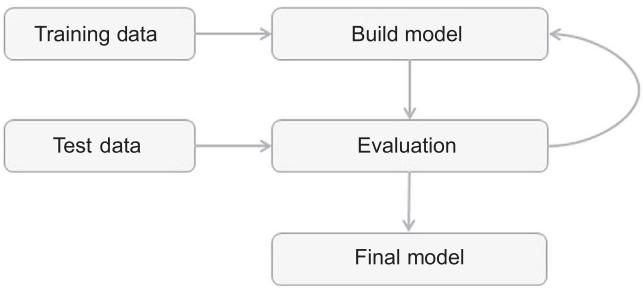


FIGURE 2.4
Modeling steps.

Table 2.3 Training Dataset

Borrower	Credit Score (*X*)	Interest Rate (*Y*) (%)
01	500	7.31
02	600	6.70
03	700	5.95
05	800	5.40
06	800	5.70
08	550	7.00
09	650	6.50

Table 2.4 Test Dataset

Borrower	Credit Score (*X*)	Interest Rate (*Y*)
04	700	6.40
07	750	5.90
10	825	5.70

2.3.1 Training and Testing Datasets

The modeling step creates a representative model inferred from the data. The dataset used to create the model, with known attributes and target, is called the *training dataset*. The validity of the created model will also need to be checked with another known dataset called the *test dataset* or *validation dataset*. To facilitate this process, the overall known dataset can be split into a training dataset and a test dataset. A standard rule of thumb is two-thirds of the data are to be used as training and one-third as a test dataset. Tables 2.3 and 2.4 show the random split of training and test data, based on the example dataset shown in Table 2.1. Fig. 2.5 shows the scatterplot of the entire example dataset with the training and test datasets marked.

2.3.2 Learning Algorithms

The business question and the availability of data will dictate what data science task (association, classification, regression, etc.,) can to be used. The practitioner determines the appropriate data science algorithm within the chosen category. For example, within a classification task many algorithms can be chosen from: decision trees, rule induction, neural networks, Bayesian models, k-NN, etc. Likewise, within decision tree techniques, there are quite a number of variations of learning algorithms like classification and regression tree (CART), CHi-squared Automatic Interaction Detector (CHAID) etc.

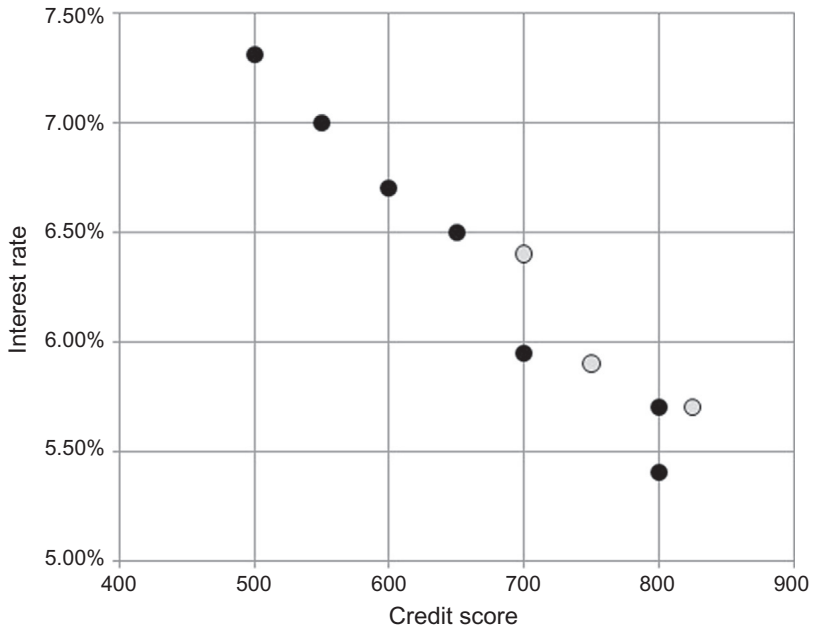


FIGURE 2.5
Scatterplot of training and test data.

These algorithms will be reviewed in detail in later chapters. It is not uncommon to use multiple data science tasks and algorithms to solve a business question.

Interest rate prediction is a regression problem. A simple linear regression technique will be used to model and generalize the relationship between credit score and interest rate. The training set of seven records is used to create the model and the test set of three records is used to evaluate the validity of the model.

The objective of simple linear regression can be visualized as fitting a straight line through the data points in a scatterplot (Fig. 2.6). The line has to be built in such a way that the sum of the squared distance from the data points to the line is minimal. The line can be expressed as:

$$y = a * x + b \tag{2.1}$$

where y is the output or dependent variable, x is the input or independent variable, b is the y-intercept, and a is the coefficient of x. The values of a and b can be found in such a way so as to minimize the sum of the squared residuals of the line.

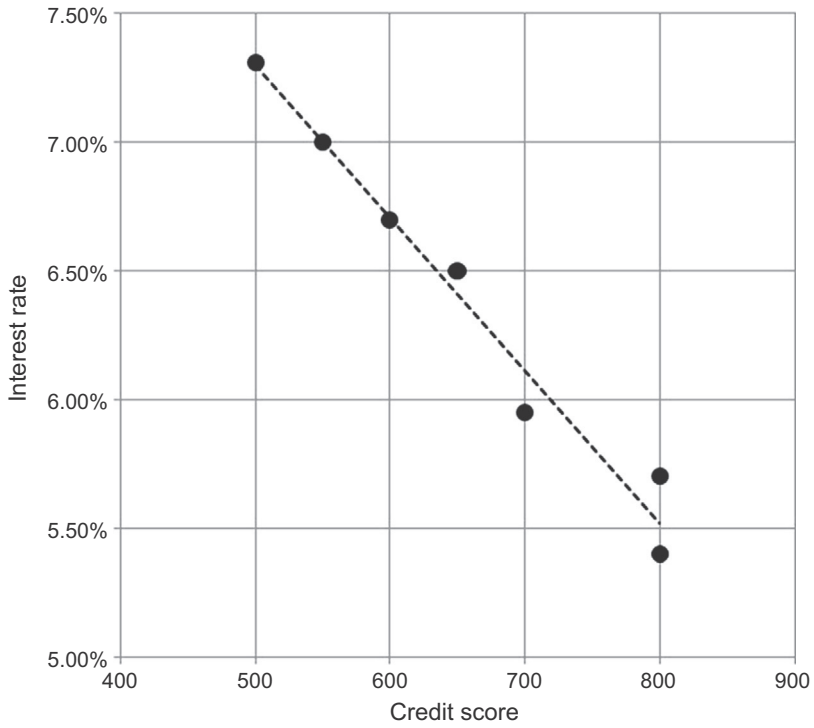


FIGURE 2.6
Regression model.

The line shown in Eq. (2.1) serves as a model to predict the outcome of new unlabeled datasets. For the interest rate dataset, the simple linear regression for the interest rate (y) has been calculated as (details in Chapter 5: Regression):

$$y = 0.1 + \frac{6}{100,000}x$$

$$\text{Interest rate} = 10 - \frac{6 \times \text{credit score}}{1000}$$

Using this model, the interest rate for a new borrower with a specific credit score can be calculated.

2.3.3 Evaluation of the Model

The model generated in the form of an equation is generalized and synthesized from seven training records. The credit score in the equation can be substituted to see if the model estimates the interest rate for each of the

Table 2.5 Evaluation of Test Dataset

Borrower	Credit Score (*X*)	Interest Rate (*Y*) (%)	Model Predicted (*Y*) (%)	Model Error (%)
04	700	6.40	6.11	− 0.29
07	750	5.90	5.81	− 0.09
10	825	5.70	5.37	− 0.33

seven training records. The estimation may not be exactly the same as the values in the training records. A model should not memorize and output the same values that are in the training records. The phenomenon of a model memorizing the training data is called *overfitting*. An overfitted model just memorizes the training records and will underperform on real unlabeled new data. The model should generalize or *learn* the relationship between credit score and interest rate. To evaluate this relationship, the validation or test dataset, which was not previously used in building the model, is used for evaluation, as shown in Table 2.5.

Table 2.5 provides the three testing records where the value of the interest rate is known; these records were not used to build the model. The actual value of the interest rate can be compared against the predicted value using the model, and thus, the *prediction error* can be calculated. As long as the error is acceptable, this model is ready for deployment. The error rate can be used to compare this model with other models developed using different algorithms like neural networks or Bayesian models, etc.

2.3.4 Ensemble Modeling

Ensemble modeling is a process where multiple diverse base models are used to predict an outcome. The motivation for using ensemble models is to reduce the generalization error of the prediction. As long as the base models are diverse and *independent*, the prediction error decreases when the ensemble approach is used. The approach seeks the wisdom of crowds in making a prediction. Even though the ensemble model has multiple base models within the model, it acts and performs as a single model. Most of the practical data science applications utilize ensemble modeling techniques.

At the end of the modeling stage of the data science process, one has (1) analyzed the business question; (2) sourced the data relevant to answer the question; (3) selected a data science technique to answer the question; (4) picked a data science algorithm and prepared the data to suit the algorithm; (5) split the data into training and test datasets; (6) built a generalized model from the training dataset; and (7) validated the model against the test

dataset. This model can now be used to predict the interest rate of new borrowers by integrating it in the actual loan approval process.

2.4 APPLICATION

Deployment is the stage at which the model becomes production ready or *live*. In business applications, the results of the data science process have to be assimilated into the business process—usually in software applications. The model deployment stage has to deal with: assessing model readiness, technical integration, response time, model maintenance, and assimilation.

2.4.1 Production Readiness

The production readiness part of the deployment determines the critical qualities required for the deployment objective. Consider two business use cases: determining whether a consumer qualifies for a loan and determining the groupings of customers for an enterprise by marketing function.

The consumer credit approval process is a real-time endeavor. Either through a consumer-facing website or through a specialized application for frontline agents, the credit decisions and terms need to be provided in real-time as soon as prospective customers provide the relevant information. It is optimal to provide a quick decision while also proving accurate. The decision-making model has to collect data from the customer, integrate third-party data like credit history, and make a decision on the loan approval and terms in a matter of seconds. The critical quality of this model deployment is real-time prediction.

Segmenting customers based on their relationship with the company is a thoughtful process where signals from various customer interactions are collected. Based on the patterns, similar customers are put in cohorts and campaign strategies are devised to best engage the customer. For this application, batch processed, time lagged data would suffice. The critical quality in this application is the ability to find unique patterns amongst customers, not the response time of the model. The business application informs the choices that need to be made in the data preparation and modeling steps.

2.4.2 Technical Integration

Currently, it is quite common to use data science automation tools or coding using R or Python to develop models. Data science tools save time as they do not require the writing of custom codes to execute the algorithm. This allows the analyst to focus on the data, business logic, and exploring patterns

from the data. The models created by data science tools can be ported to production applications by utilizing the Predictive Model Markup Language (PMML) (Guazzelli, Zeller, Lin, & Williams, 2009) or by invoking data science tools in the production application. PMML provides a portable and consistent format of model description which can be read by most data science tools. This allows the flexibility for practitioners to develop the model with one tool (e.g., RapidMiner) and deploy it in another tool or application. Some models such as simple regression, decision trees, and induction rules for predictive analytics can be incorporated directly into business applications and business intelligence systems easily. These models are represented by simple equations and the "if-then" rule, hence, they can be ported easily to most programming languages.

2.4.3 Response Time

Data science algorithms, like k-NN, are easy to build, but quite slow at predicting the unlabeled records. Algorithms such as the decision tree take time to build but are fast at prediction. There are trade-offs to be made between production responsiveness and modeling build time. The quality of prediction, accessibility of input data, and the response time of the prediction remain the critical quality factors in business application.

2.4.4 Model Refresh

The key criterion for the ongoing relevance of the model is the representativeness of the dataset it is processing. It is quite normal that the conditions in which the model is built change after the model is sent to deployment. For example, the relationship between the credit score and interest rate change frequently based on the prevailing macroeconomic conditions. Hence, the model will have to be refreshed frequently. The validity of the model can be routinely tested by using the new known test dataset and calculating the prediction error rate. If the error rate exceeds a particular threshold, then the model has to be refreshed and redeployed. Creating a maintenance schedule is a key part of a deployment plan that will sustain a relevant model.

2.4.5 Assimilation

In the descriptive data science applications, deploying a model to live systems may not be the end objective. The objective may be to assimilate the knowledge gained from the data science analysis to the organization. For example, the objective may be finding logical clusters in the customer database so that separate marketing approaches can be developed for each customer cluster. Then the next step may be a classification task for new customers to bucket them in one of known clusters. The association analysis

provides a solution for the market basket problem, where the task is to find which two products are purchased together most often. The challenge for the data science practitioner is to articulate these findings, establish relevance to the original business question, quantify the risks in the model, and quantify the business impact. This is indeed a challenging task for data science practitioners. The business user community is an amalgamation of different points of view, different quantitative mindsets, and skill sets. Not everyone is aware about the process of data science and what it can and cannot do. Some aspects of this challenge can be addressed by focusing on the end result, the impact of knowing the discovered information, and the follow-up actions, instead of the technical process of extracting the information through data science.

2.5 KNOWLEDGE

The data science process provides a framework to extract nontrivial information from data. With the advent of massive storage, increased data collection, and advanced computing paradigms, the available datasets to be utilized are only increasing. To extract knowledge from these massive data assets, advanced approaches need to be employed, like data science algorithms, in addition to standard business intelligence reporting or statistical analysis. Though many of these algorithms can provide valuable knowledge, it is up to the practitioner to skillfully transform a business problem to a data problem and apply the right algorithm. Data science, like any other technology, provides various options in terms of algorithms and parameters within the algorithms. Using these options to extract the right information from data is a bit of an art and can be developed with practice.

The data science process starts with prior knowledge and ends with posterior knowledge, which is the incremental insight gained. As with any quantitative technique, the data science process can bring up spurious irrelevant patterns from the dataset. Not all discovered patterns lead to incremental knowledge. Again, it is up to the practitioner to invalidate the irrelevant patterns and identify the meaningful information. The impact of the information gained through data science can be measured in an application. It is the difference between gaining the information through the data science process and the insights from basic data analysis. Finally, the whole data science process is a framework to invoke the right questions (Chapman et al., 2000) and provide guidance, through the right approaches, to solve a problem. It is not meant to be used as a set of rigid rules, but as a set of iterative, distinct steps that aid in knowledge discovery.

CHAPTER 3

Data Exploration

The word "data" is derived from the Latin word *dare*, which means "something given"—an observation or a fact about a subject. (Interestingly, the Sanskrit word *dAta* also means "given"). Data science helps decipher the hidden useful relationships within data. Before venturing into any advanced analysis of data using statistical, machine learning, and algorithmic techniques, it is essential to perform basic data exploration to study the basic characteristics of a dataset. Data exploration helps with understanding data better, to prepare the data in a way that makes advanced analysis possible, and sometimes to get the necessary insights from the data faster than using advanced analytical techniques.

Simple pivot table functions, computing statistics like mean and deviation, and plotting data as a line, bar, and scatter charts are part of data exploration techniques that are used in everyday business settings. Data exploration, also known as exploratory data analysis, provides a set of tools to obtain fundamental understanding of a dataset. The results of data exploration can be extremely powerful in grasping the structure of the data, the distribution of the values, and the presence of extreme values and the interrelationships between the attributes in the dataset. Data exploration also provides guidance on applying the right kind of further statistical and data science treatment.

Data exploration can be broadly classified into two types—descriptive statistics and data visualization. Descriptive statistics is the process of condensing key characteristics of the dataset into simple numeric metrics. Some of the common quantitative metrics used are mean, standard deviation, and correlation. Visualization is the process of projecting the data, or parts of it, into multi-dimensional space or abstract images. All the useful (and adorable) charts fall under this category. Data exploration in the context of data science uses both descriptive statistics and visualization techniques.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00003-4

3.1 OBJECTIVES OF DATA EXPLORATION

In the data science process, data exploration is leveraged in many different steps including preprocessing or data preparation, modeling, and interpretation of the modeling results.

1. *Data understanding*: Data exploration provides a high-level overview of each attribute (also called variable) in the dataset and the interaction between the attributes. Data exploration helps answers the questions like what is the typical value of an attribute or how much do the data points differ from the typical value, or presence of extreme values.
2. *Data preparation*: Before applying the data science algorithm, the dataset has to be prepared for handling any of the anomalies that may be present in the data. These anomalies include outliers, missing values, or highly correlated attributes. Some data science algorithms do not work well when input attributes are correlated with each other. Thus, correlated attributes need to be identified and removed.
3. *Data science tasks*: Basic data exploration can sometimes substitute the entire data science process. For example, scatterplots can identify clusters in low-dimensional data or can help develop regression or classification models with simple visual rules.
4. *Interpreting the results*: Finally, data exploration is used in understanding the prediction, classification, and clustering of the results of the data science process. Histograms help to comprehend the distribution of the attribute and can also be useful for visualizing numeric prediction, error rate estimation, etc.

3.2 DATASETS

Throughout the rest of this chapter (and the book) a few classic datasets, which are simple to understand, easy to explain, and can be used commonly across many different data science techniques, will be introduced. The most popular datasets used to learn data science is probably the *Iris dataset*, introduced by Ronald Fisher, in his seminal work on discriminant analysis, "The use of multiple measurements in taxonomic problems" (Fisher, 1936). Iris is a flowering plant that is found widely, across the world. The genus of Iris contains more than 300 different species. Each species exhibits different physical characteristics like shape and size of the flowers and leaves. The *Iris dataset* contains 150 observations of three different species, *Iris setosa*, *Iris virginica*, and *I. versicolor*, with 50 observations each. Each observation consists of four attributes: sepal length, sepal width, petal length, and petal width. The fifth attribute, the label, is the name of the species observed, which takes the values *I. setosa*, *I. virginica*, and *I. versicolor*. The petals are the brightly

colored inner part of the flowers and the sepals form the outer part of the flower and are usually green in color. However, in an Iris flower, both sepals and petals are bright purple in color, but can be distinguished from each other by differences in the shape (Fig. 3.1).

All four attributes in the Iris dataset are numeric continuous values measured in centimeters. One of the species, *I. setosa*, can be easily distinguished from the other two using simple rules like the petal length is less than 2.5 cm. Separating the *virginica* and *versicolor* classes requires more complex rules that involve more attributes. The dataset is available in all standard data science tools, such as RapidMiner, or can be downloaded from public websites such as the University of California Irvine—Machine Learning repository[2] (Bache & Lichman, 2013). This dataset and other datasets used in this book can be accessed from the book companion website: www.IntroDataScience.com.

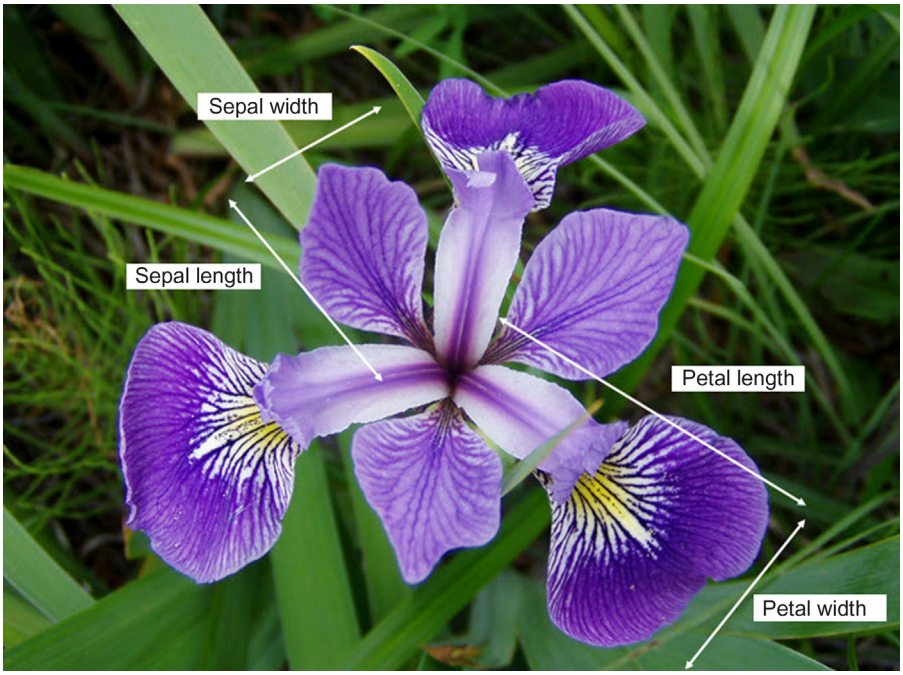


FIGURE 3.1

Iris versicolor. Source: *Photo by Danielle Langlois. July 2005 (Image modified from original by marking parts. "*Iris versicolor *3." Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons.*[1]*)*

[1] http://commons.wikimedia.org/wiki/File:Iris_versicolor_3.jpg#mediaviewer/File:Iris_versicolor_3.jpg.
[2] https://archive.ics.uci.edu/ml/datasets.html.

The Iris dataset is used for learning data science mainly because it is simple to understand, explore, and can be used to illustrate how different data science algorithms approach the problem on the same standard dataset. The dataset extends beyond two dimensions, with three class labels, of which one class is easily separable (*I. setosa*) just by visual exploration, while classifying the other two classes is slightly more challenging. It helps to reaffirm the classification results that can be derived based on visual rules, and at the same time sets the stage for data science to build new rules beyond the limits of visual exploration.

3.2.1 Types of Data

Data come in different formats and types. Understanding the properties of each attribute or feature provides information about what kind of operations can be performed on that attribute. For example, the temperature in weather data can be expressed as any of the following formats:

- Numeric centigrade (31°C, 33.3°C) or Fahrenheit (100°F, 101.45°F) or on the Kelvin scale
- Ordered labels as in hot, mild, or cold
- Number of days within a year below 0°C (10 days in a year below freezing)

All of these attributes indicate temperature in a region, but each have different data types. A few of these data types can be converted from one to another.

Numeric or Continuous

Temperature expressed in Centigrade or Fahrenheit is numeric and continuous because it can be denoted by numbers and take an infinite number of values between digits. Values are ordered and calculating the difference between the values makes sense. Hence, additive and subtractive mathematical operations and logical comparison operators like greater than, less than, and equal to, operations can be applied.

An integer is a special form of the numeric data type which does not have decimals in the value or more precisely does not have infinite values between consecutive numbers. Usually, they denote a count of something, number of days with temperature less than 0°C, number of orders, number of children in a family, etc.

If a zero point is defined, numeric data become a *ratio* or *real* data type. Examples include temperature in Kelvin scale, bank account balance, and income. Along with additive and logical operations, ratio operations can be performed with this data type. Both integer and ratio data types are categorized as a *numeric* data type in most data science tools.

Categorical or Nominal

Categorical data types are attributes treated as distinct symbols or just names. The color of the iris of the human eye is a categorical data type because it takes a value like black, green, blue, gray, etc. There is no direct relationship among the data values, and hence, mathematical operators except the logical or "is equal" operator cannot be applied. They are also called a nominal or polynominal data type, derived from the Latin word for *name*.

An ordered nominal data type is a special case of a categorical data type where there is some kind of order among the values. An example of an ordered data type is temperature expressed as hot, mild, cold.

Not all data science tasks can be performed on all data types. For example, the neural network algorithm does not work with categorical data. However, one data type can be converted to another using a type conversion process, but this may be accompanied with possible loss of information. For example, credit scores expressed in poor, average, good, and excellent categories can be converted to either 1, 2, 3, and 4 or average underlying numerical scores like 400, 500, 600, and 700 (scores here are just an example). In this type conversion, there is no loss of information. However, conversion from numeric credit score to categories (poor, average, good, and excellent) does incur loss of information.

3.3 DESCRIPTIVE STATISTICS

Descriptive statistics refers to the study of the aggregate quantities of a dataset. These measures are some of the commonly used notations in everyday life. Some examples of descriptive statistics include average annual income, median home price in a neighborhood, range of credit scores of a population, etc. In general, descriptive analysis covers the following characteristics of the sample or population dataset (Kubiak & Benbow, 2006):

Characteristics of the Dataset	Measurement Technique
Center of the dataset	Mean, median, and mode
Spread of the dataset	Range, variance, and standard deviation
Shape of the distribution of the dataset	Symmetry, skewness, and kurtosis

The definition of these metrics will be explored shortly. Descriptive statistics can be broadly classified into univariate and multivariate exploration depending on the number of attributes under analysis.

3.3.1 Univariate Exploration

Univariate data exploration denotes analysis of one attribute at a time. The example Iris dataset for one species, *I. setosa*, has 50 observations and 4 attributes, as shown in Table 3.1. Here some of the descriptive statistics for sepal length attribute are explored.

Measure of Central Tendency

The objective of finding the central location of an attribute is to quantify the dataset with one central or most common number.

- *Mean*: The mean is the arithmetic average of all observations in the dataset. It is calculated by summing all the data points and dividing by the number of data points. The mean for sepal length in centimeters is 5.0060.
- *Median*: The median is the value of the central point in the distribution. The median is calculated by sorting all the observations from small to large and selecting the mid-point observation in the sorted list. If the number of data points is even, then the average of the middle two data points is used as the median. The median for sepal length is in centimeters is 5.0000.
- *Mode*: The mode is the most frequently occurring observation. In the dataset, data points may be repetitive, and the most repetitive data point is the mode of the dataset. In this example, the mode in centimeters is 5.1000.

Table 3.1 Iris Dataset and Descriptive Statistics (Fisher, 1936)

Observation	Sepal Length	Sepal Width	Petal Length	Petal Width
1	5.1	3.5	1.4	0.2
2	4.9	3.1	1.5	0.1
...	...	...	...	...
49	5	3.4	1.5	0.2
50	4.4	2.9	1.4	0.2
Statistics	**Sepal Length**	**Sepal Width**	**Petal Length**	**Petal Width**
Mean	5.006	3.418	1.464	0.244
Median	5.000	3.400	1.500	0.200
Mode	5.100	3.400	1.500	0.200
Range	1.500	2.100	0.900	0.500
Standard deviation	0.352	0.381	0.174	0.107
Variance	0.124	0.145	0.030	0.011

In an attribute, the mean, median, and mode may be different numbers, and this indicates the shape of the distribution. If the dataset has outliers, the mean will get affected while in most cases the median will not. The mode of the distribution can be different from the mean or median, if the underlying dataset has more than one natural normal distribution.

Measure of Spread

In desert regions, it is common for the temperature to cross above 110°F during the day and drop below 30°F during the night while the average temperature for a 24-hour period is around 70°F. Obviously, the experience of living in the desert is not the same as living in a tropical region with the same average daily temperature around 70°F, where the temperature within the day is between 60°F and 80°F. What matters here is not just the central location of the temperature, but the *spread* of the temperature. There are two common metrics to quantify spread.

- *Range:* The range is the difference between the maximum value and the minimum value of the attribute. The range is simple to calculate and articulate but has shortcomings as it is severely impacted by the presence of outliers and fails to consider the distribution of all other data points in the attributes. In the example, the range for the temperature in the desert is 80°F and the range for the tropics is 20°F. The desert region experiences larger temperature swings as indicated by the range.
- *Deviation:* The variance and standard deviation measures the spread, by considering all the values of the attribute. Deviation is simply measured as the difference between any given value (x_i) and the mean of the sample (μ). The variance is the sum of the squared deviations of all data points divided by the number of data points. For a dataset with N observations, the variance is given by the following equation:

$$\text{Variance} = s^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{3.1}$$

Standard deviation is the square root of the variance. Since the standard deviation is measured in the same units as the attribute, it is easy to understand the magnitude of the metric. High standard deviation means the data points are spread widely around the central point. Low standard deviation means data points are closer to the central point. If the distribution of the data aligns with the *normal distribution*, then 68% of the data points lie within one standard deviation from the mean. Fig. 3.2 provides the univariate summary of the Iris dataset with all 150 observations, for each of the four numeric attributes.

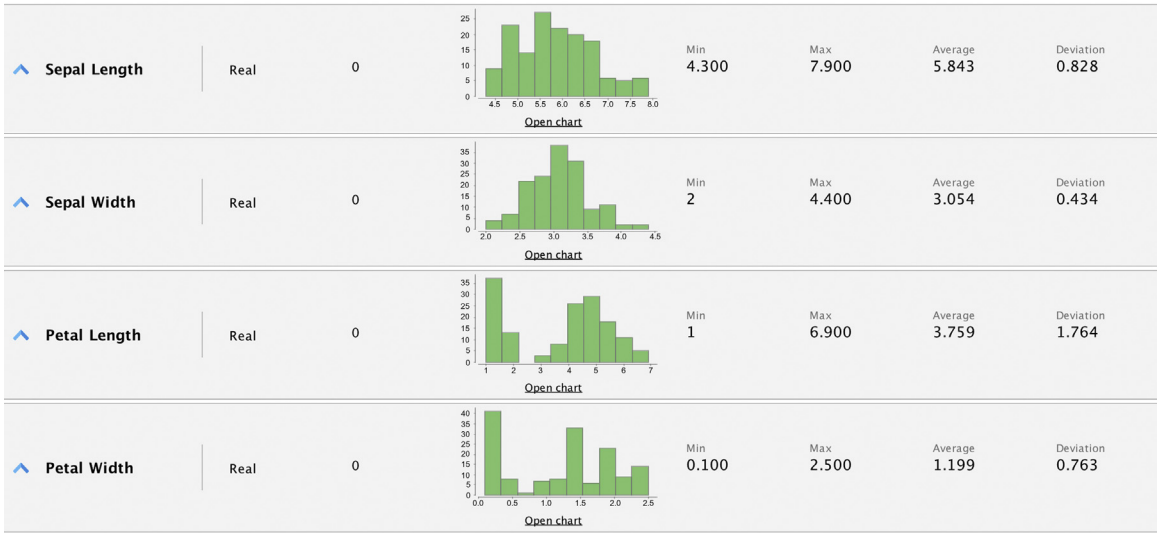


FIGURE 3.2
Descriptive statistics for the Iris dataset.

3.3.2 Multivariate Exploration

Multivariate exploration is the study of more than one attribute in the dataset simultaneously. This technique is critical to understanding the relationship between the attributes, which is central to data science methods. Similar to univariate explorations, the measure of central tendency and variance in the data will be discussed.

Central Data Point

In the Iris dataset, each data point as a set of all the four attributes can be expressed:

observation *i*: {sepal length, sepal width, petal length, petal width}

For example, observation one: {5.1, 3.5, 1.4, 0.2}. This observation point can also be expressed in four-dimensional Cartesian coordinates and can be plotted in a graph (although plotting more than three dimensions in a visual graph can be challenging). In this way, all 150 observations can be expressed in Cartesian coordinates. If the objective is to find the most "typical" observation point, it would be a data point made up of the mean of each attribute in the dataset independently. For the Iris dataset shown in Table 3.1, the central mean point is {5.006, 3.418, 1.464, 0.244}. This data point may not be an actual observation. It will be a hypothetical data point with the most typical attribute values.

Correlation

Correlation measures the statistical relationship between two attributes, particularly dependence of one attribute on another attribute. When two attributes are highly correlated with each other, they both vary at the same rate with each other either in the same or in opposite directions. For example, consider average temperature of the day and ice cream sales. Statistically, the two attributes that are correlated are dependent on each other and one may be used to predict the other. If there are sufficient data, future sales of ice cream can be predicted if the temperature forecast is known. However, correlation between two attributes does not imply causation, that is, one doesn't necessarily cause the other. The ice cream sales and the shark attacks are correlated, however there is no causation. Both ice cream sales and shark attacks are influenced by the third attribute—the summer season. Generally, ice cream sales spikes as temperatures rise. As more people go to beaches during summer, encounters with sharks become more probable.

Correlation between two attributes is commonly measured by the Pearson correlation coefficient (r), which measures the strength of *linear* dependence (Fig. 3.3). Correlation coefficients take a value from $-1 \leq r \leq 1$. A value closer to 1 or -1 indicates the two attributes are highly correlated, with perfect correlation at 1 or -1. Perfect correlation also exists when the attributes are governed by formulas and laws. For example, observations of the values of gravitational force and the mass of the object (Newton's second law) or the quantity of the products sold and total revenue (price* volume = revenue). A correlation value of 0 means there is no linear relationship between two attributes.

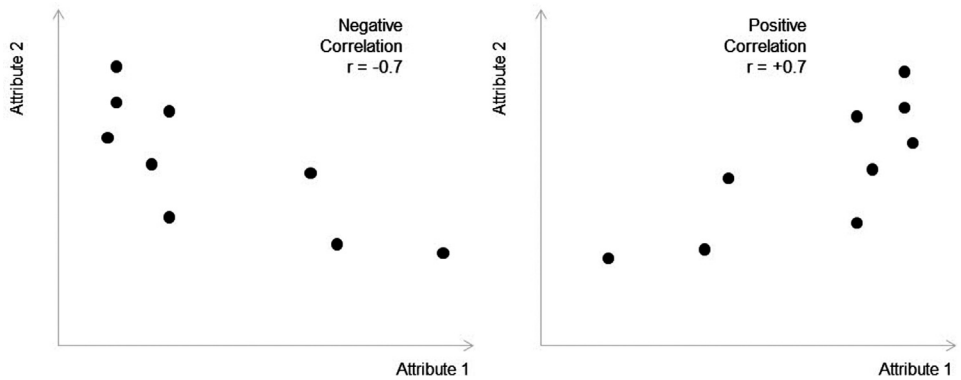


FIGURE 3.3
Correlation of attributes.

The Pearson correlation coefficient between two attributes x and y is calculated with the formula:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}} = \frac{\sum_{i=1}^{N}(x_i-\bar{x})(y_i-\bar{y})}{N\times s_x\times s_y} \tag{3.2}$$

where s_x and s_y are the standard deviations of random variables x and y, respectively. The Pearson correlation coefficient has some limitations in quantifying the strength of correlation. When datasets have more complex nonlinear relationships like quadratic functions, only the effects on linear relationships are considered and quantified using correlation coefficient. The presence of outliers can also skew the measure of correlation. Visually, correlation can be observed using scatterplots with the attributes in each Cartesian coordinate (Fig. 3.3). In fact, visualization should be the first step in understanding correlation because it can identify nonlinear relationships and show any outliers in the dataset. Anscombe's quartet (Anscombe, 1973) clearly illustrates the limitations of relying only on the correlation coefficient to understand the data (Fig. 3.4). The quartet consists of four different datasets, with two attributes (x, y). All four datasets have the same mean, the same variance for x and y, and the same correlation coefficient between x and y, but look drastically different when plotted on a chart. This evidence illustrates the necessity of visualizing the attributes instead of just calculating statistical metrics.

3.4 DATA VISUALIZATION

Visualizing data is one of the most important techniques of data discovery and exploration. Though visualization is not considered a data science technique, terms like visual mining or pattern discovery based on visuals are increasingly used in the context of data science, particularly in the business world. The discipline of data visualization encompasses the methods of expressing data in an abstract visual form. The visual representation of data provides easy comprehension of complex data with multiple attributes and their underlying relationships. The motivation for using data visualization includes:

- *Comprehension of dense information*: A simple visual chart can easily include thousands of data points. By using visuals, the user can see the big picture, as well as longer term trends that are extremely difficult to interpret purely by expressing data in numbers.

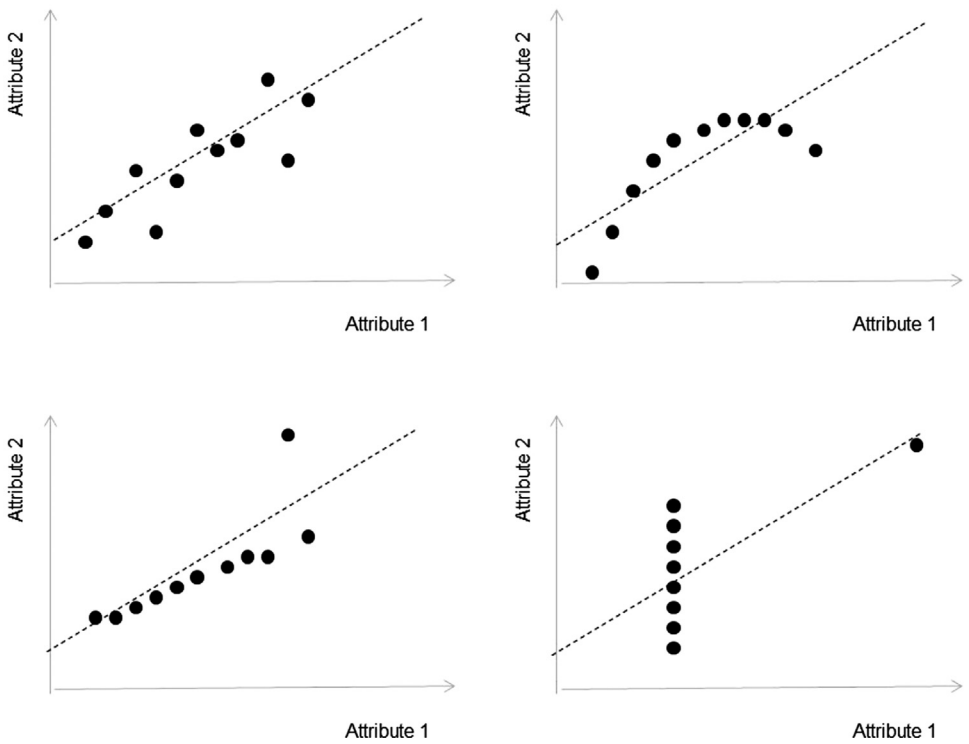


FIGURE 3.4

Anscombe's Quartet: descriptive statistics versus visualization. Source*: Adapted from: Anscombe, F. J., 1973. Graphs in statistical analysis*, American Statistician, 27*(1), pp. 19–20.*

- *Relationships*: Visualizing data in Cartesian coordinates enables exploration of the relationships between the attributes. Although representing more than three attributes on the *x*, *y*, and *z*-axes is not feasible in Cartesian coordinates, there are a few creative solutions available by changing properties like the size, color, and shape of data markers or using flow maps (Tufte, 2001), where more than two attributes are used in a two-dimensional medium.

Vision is one of the most powerful senses in the human body. As such, it is intimately connected with cognitive thinking (Few, 2006). Human vision is trained to discover patterns and anomalies even in the presence of a large volume of data. However, the effectiveness of the pattern detection depends on how effectively the information is visually presented. Hence, selecting suitable visuals to explore data is critically important in discovering and comprehending hidden patterns in the data (Ware, 2004). As with descriptive statistics, visualization techniques are also categorized into: univariate visualization, multivariate visualization and visualization of a large number of attributes using parallel dimensions.

Some of the common data visualization techniques used to analyze data will be reviewed. Most of these visualization techniques are available in commercial spreadsheet software like MS Excel. RapidMiner, like any other data science tool, offers a wide range of visualization tools. To maintain consistency with rest of the book, all further visualizations are output from RapidMiner using the Iris dataset. Please review Chapter 15, Getting Started With RapidMiner, to become familiar with RapidMiner.

3.4.1 Univariate Visualization

Visual exploration starts with investigating one attribute at a time using univariate charts. The techniques discussed in this section give an idea of how the attribute values are distributed and the shape of the distribution.

Histogram

A histogram is one of the most basic visualization techniques to understand the frequency of the occurrence of values. It shows the distribution of the data by plotting the frequency of occurrence in a range. In a histogram, the attribute under inquiry is shown on the horizontal axis and the frequency of occurrence is on the vertical axis. For a continuous numeric data type, the range or *binning* value to group a range of values need to be specified. For example, in the case of human height in centimeters, all the occurrences between 152.00 and 152.99 are grouped under 152. There is no optimal number of bins or bin width that works for all the distributions. If the bin width is too small, the distribution becomes more precise but reveals the noise due to sampling. A general rule of thumb is to have a number of bins equal to the square root or cube root of the number of data points.

Histograms are used to find the central location, range, and shape of distribution. In the case of the petal length attribute in the Iris dataset, the data is multimodal (Fig. 3.5), where the distribution does not follow the bell curve pattern. Instead, there are two peaks in the distribution. This is due to the fact that there are 150 observations of three *different* species (hence, distributions) in the dataset. A histogram can be *stratified* to include different classes in order to gain more insight. The enhanced histogram with class labels shows the dataset is made of three different distributions (Fig. 3.6). *I. setosa's* distribution stands out with a mean around 1.25 cm and ranges from 1−2 cm. *I. versicolor* and *I. virginica's* distributions overlap *I. setosa's* have separate means.

Quartile

A *box whisker* plot is a simple visual way of showing the distribution of a continuous variable with information such as quartiles, median, and outliers,

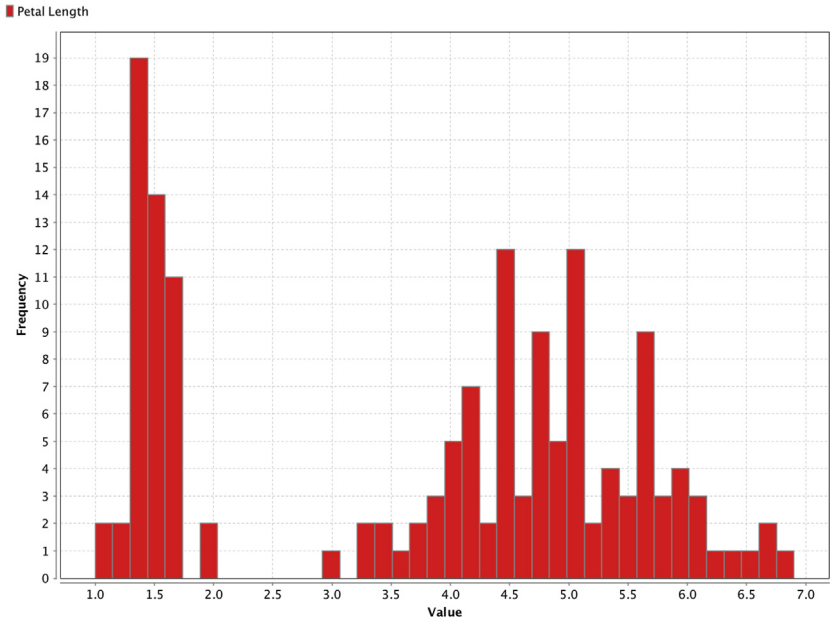


FIGURE 3.5
Histogram of petal length in Iris dataset.

overlaid by mean and standard deviation. The main attraction of box whisker or quartile charts is that distributions of multiple attributes can be compared side by side and the overlap between them can be deduced. The quartiles are denoted by Q1, Q2, and Q3 points, which indicate the data points with a 25% bin size. In a distribution, 25% of the data points will be below Q1, 50% will be below Q2, and 75% will be below Q3.

The Q1 and Q3 points in a box whisker plot are denoted by the edges of the box. The Q2 point, the median of the distribution, is indicated by a cross line within the box. The outliers are denoted by circles at the end of the whisker line. In some cases, the mean point is denoted by a solid dot overlay followed by standard deviation as a line overlay.

Fig. 3.7 shows that the quartile charts for all four attributes of the Iris dataset are plotted side by side. Petal length can be observed as having the broadest range and the sepal width has a narrow range, out of all of the four attributes.

One attribute can also be selected—petal length—and explored further using quartile charts by introducing a class label. In the plot in Fig. 3.8, we can see the distribution of three species for the petal length measurement. Similar to the previous comparison, the distribution of multiple species can be compared.

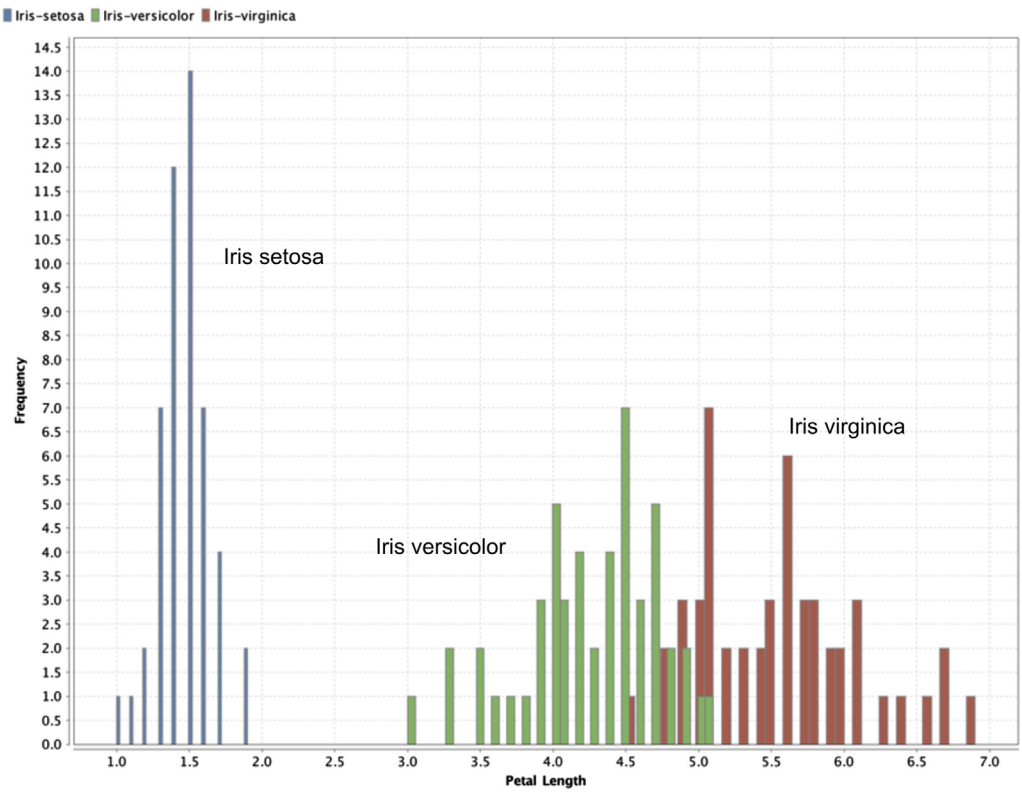


FIGURE 3.6
Class-stratified histogram of petal length in Iris dataset.

Distribution Chart

For continuous numeric attributes like petal length, instead of visualizing the actual data in the sample, its normal distribution function can be visualized instead. The normal distribution function of a continuous random variable is given by the formula:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{(x-\mu)^2/2\sigma^2} \tag{3.3}$$

where μ is the mean of the distribution and σ is the standard deviation of the distribution. Here an inherent assumption is being made that the measurements of petal length (or any continuous variable) follow the normal distribution, and hence, its distribution can be visualized instead of the actual values. The normal distribution is also called the *Gaussian distribution* or "bell curve" due to its bell shape. The normal distribution function

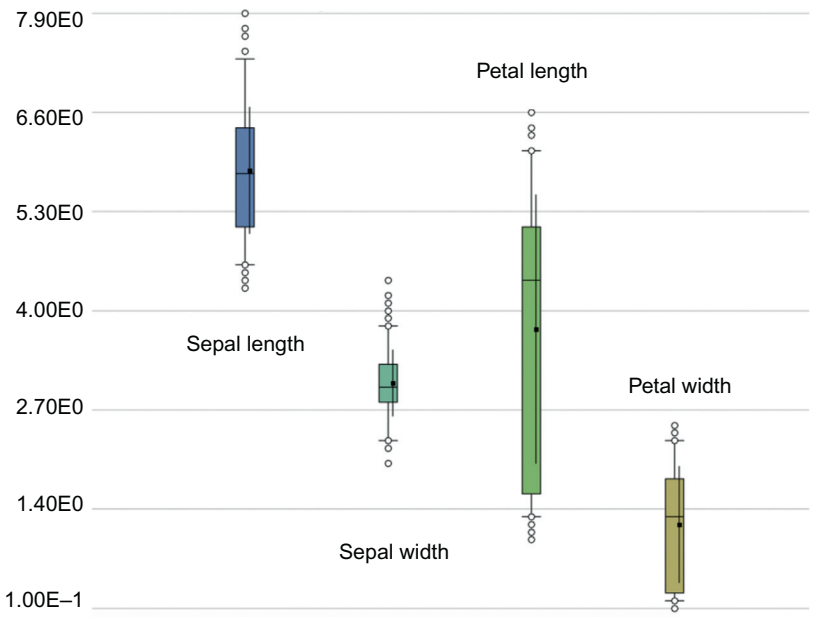


FIGURE 3.7
Quartile plot of Iris dataset.

shows the probability of occurrence of a data point within a range of values. If a dataset exhibits normal distribution, then 68.2% of data points will fall within one standard deviation from the mean; 95.4% of the points will fall within 2σ and 99.7% within 3σ of the mean. When the normal distribution curves are stratified by class type, more insight into the data can be gained. Fig. 3.9 shows the normal distribution curves for petal length measurement for each Iris species type. From the distribution chart, it can be inferred that the petal length for the *I. setosa* sample is more distinct and cohesive than *I. versicolor* and *I. virginica*. If there is an unlabeled measurement with a petal length of 1.5 cm, it can be predicted that the species is *I. setosa*. However, if the petal length measurement is 5.0 cm, there is no clear prediction, as the species could be either *Iris versicolor* and *I. virginica*.

3.4.2 Multivariate Visualization

The multivariate visual exploration considers more than one attribute in the same visual. The techniques discussed in this section focus on the relationship of one attribute with another attribute. These visualizations examine two to four attributes simultaneously.

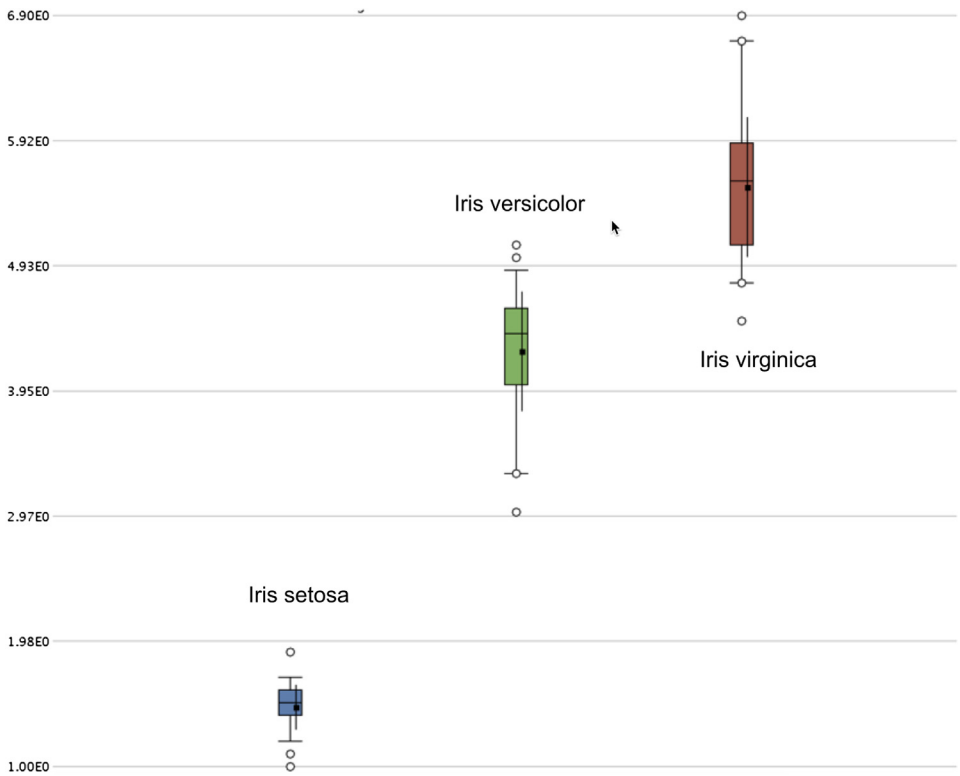


FIGURE 3.8
Class-stratified quartile plot of petal length in Iris dataset.

Scatterplot

A scatterplot is one of the most powerful yet simple visual plots available. In a scatterplot, the data points are marked in Cartesian space with attributes of the dataset aligned with the coordinates. The attributes are usually of continuous data type. One of the key observations that can be concluded from a scatterplot is the existence of a relationship between two attributes under inquiry. If the attributes are linearly correlated, then the data points align closer to an imaginary straight line; if they are not correlated, the data points are scattered. Apart from basic correlation, scatterplots can also indicate the existence of patterns or groups of clusters in the data and identify outliers in the data. This is particularly useful for low-dimensional datasets. Chapter 13: Anomaly detection, provides techniques for finding outliers in high-dimensional space.

Fig. 3.10 shows the scatterplot between petal length (x-axis) and petal width (y-axis). These two attributes are slightly correlated, because this is a measurement of the same part of the flower. When the data markers are colored to

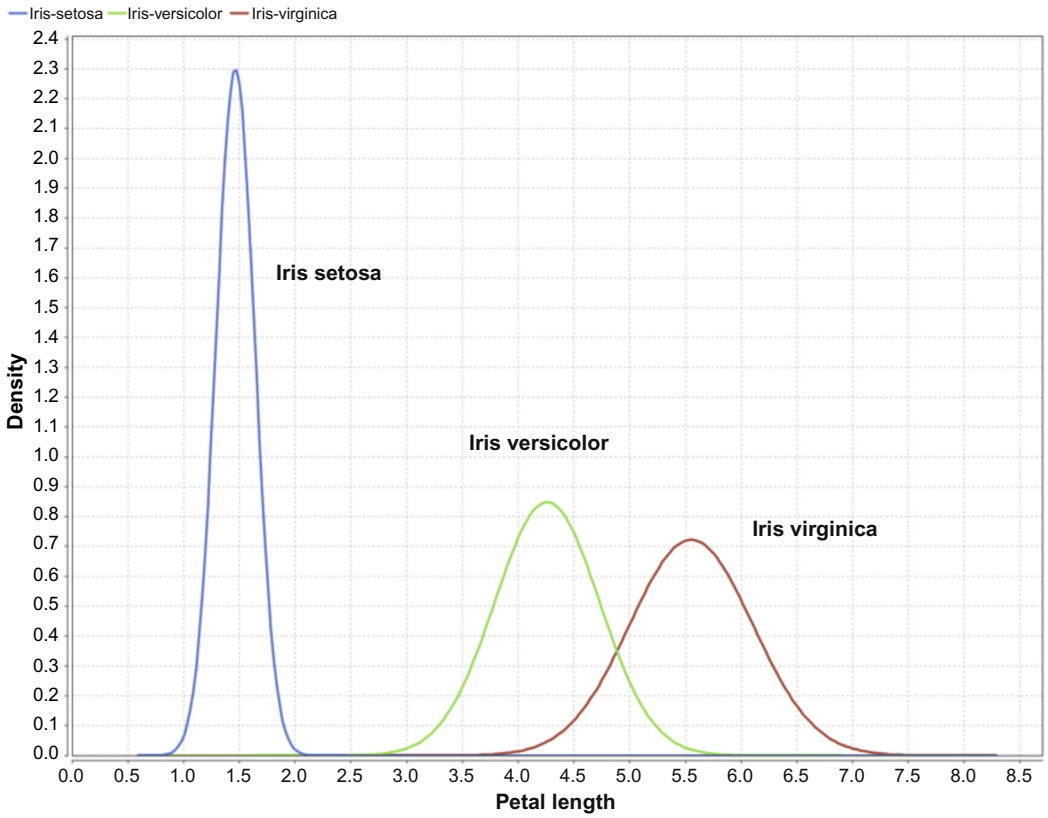


FIGURE 3.9
Distribution of petal length in Iris dataset.

indicate different species using class labels, more patterns can be observed. There is a cluster of data points, all belonging to species *I. setosa*, on the lower left side of the plot. *I. setosa* has much smaller petals. This feature can be used as a rule to predict the species of unlabeled observations. One of the limitations of scatterplots is that only two attributes can be used at a time, with an additional attribute possibly shown in the color of the data marker. However, the colors are usually reserved for class labels.

Scatter Multiple

A *scatter multiple* is an enhanced form of a simple scatterplot where more than two dimensions can be included in the chart and studied simultaneously. The primary attribute is used for the *x*-axis coordinate. The secondary axis is shared with more attributes or dimensions. In this example (Fig. 3.11), the values on the *y*-axis are shared between sepal length, sepal

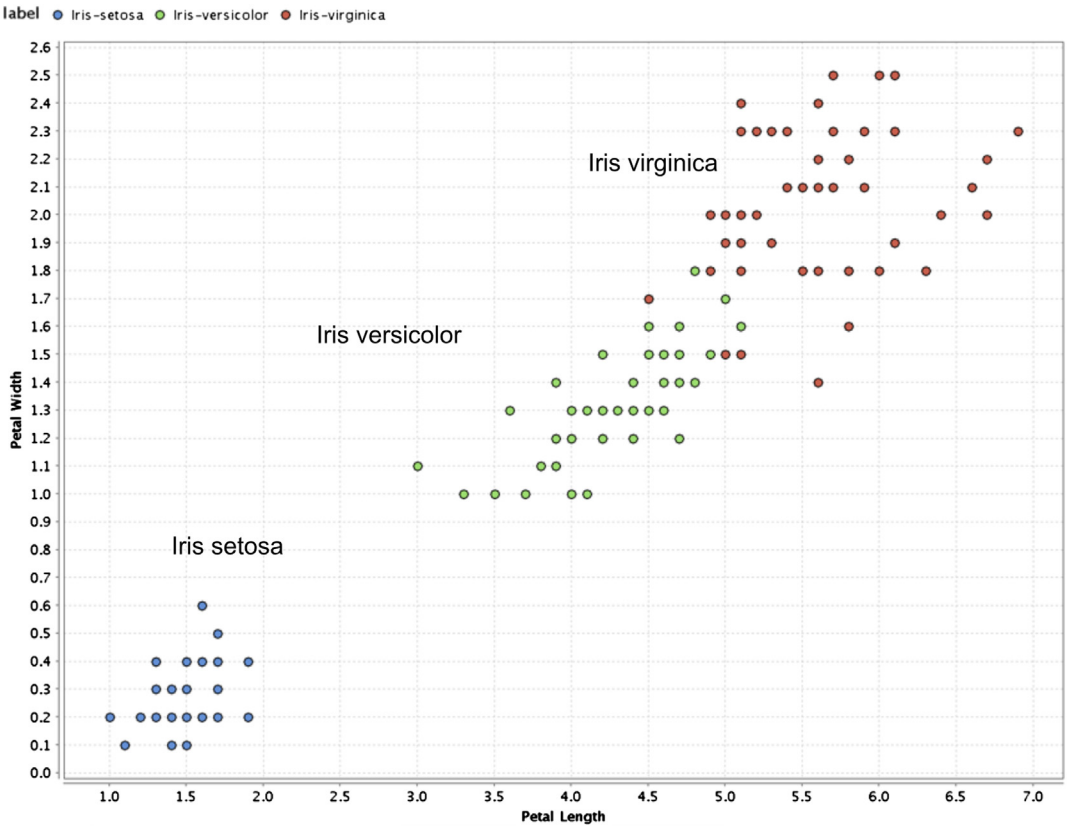


FIGURE 3.10
Scatterplot of Iris dataset.

width, and petal width. The name of the attribute is conveyed by colors used in data markers. Here, sepal length is represented by data points occupying the topmost part of the chart, sepal width occupies the middle portion, and petal width is in the bottom portion. Note that the data points are *duplicated for each attribute in the y-axis*. Data points are color-coded for each dimension in *y*-axis while the *x*-axis is anchored with one attribute—petal length. All the attributes sharing the y-axis should be of the same unit or normalized.

Scatter Matrix

If the dataset has more than two attributes, it is important to look at combinations of all the attributes through a scatterplot. A *scatter matrix* solves this need by comparing all combinations of attributes with individual scatterplots and arranging these plots in a matrix.

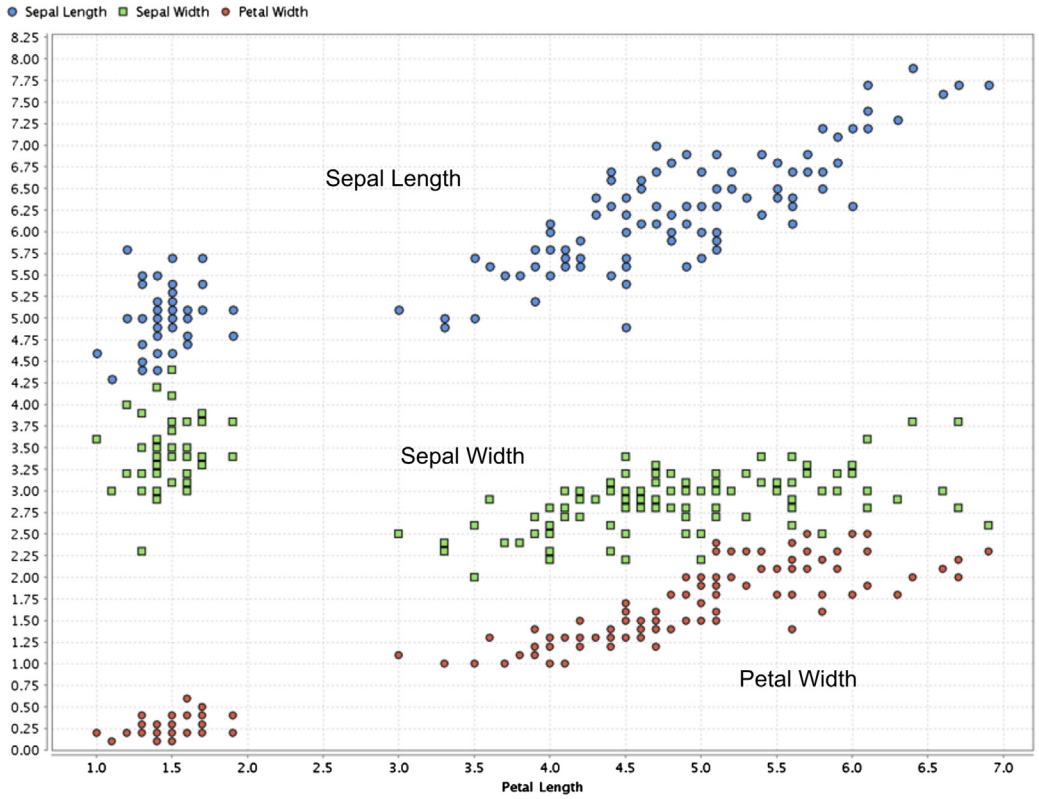


FIGURE 3.11
Scatter multiple plot of Iris dataset.

A scatter matrix for all four attributes in the Iris dataset is shown in Fig. 3.12. The color of the data point is used to indicate the species of the flower. Since there are four attributes, there are four rows and four columns, for a total of 16 scatter charts. Charts in the diagonal are a comparison of the attribute with itself; hence, they are eliminated. Also, the charts below the diagonal are mirror images of the charts above the diagonal. In effect, there are six distinct comparisons in scatter multiples of four attributes. Scatter matrices provide an effective visualization of comparative, multivariate, and high-density data displayed in small multiples of the similar scatterplots (Tufte, 2001).

Bubble Chart

A *bubble chart* is a variation of a simple scatterplot with the addition of one more attribute, which is used to determine the size of the data point. In the Iris dataset, petal length and petal width are used for x and y-axis, respectively

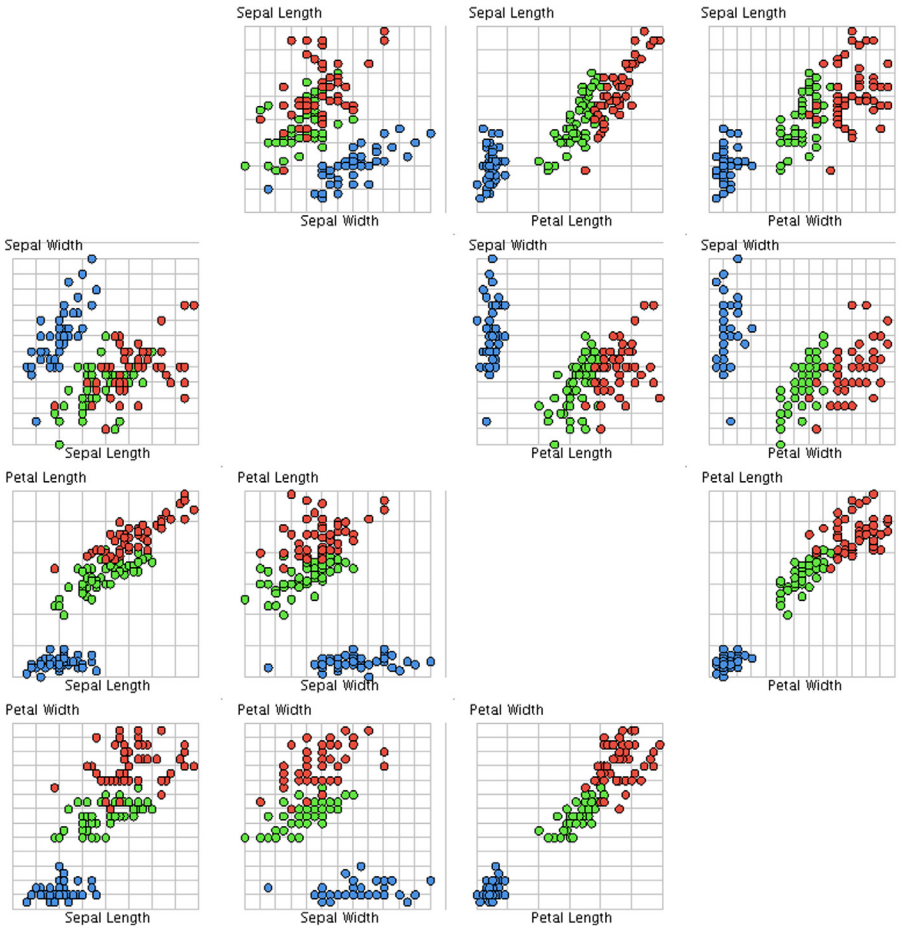


FIGURE 3.12
Scatter matrix plot of Iris dataset.

and sepal width is used for the size of the data point. The color of the data point represents a species class label (Fig. 3.13).

Density Chart

Density charts are similar to the scatterplots, with one more dimension included as a background color. The data point can also be colored to visualize one dimension, and hence, a total of four dimensions can be visualized in a density chart. In the example in Fig. 3.14, petal length is used for the *x*-axis, sepal length for the *y*-axis, sepal width for the background color, and class label for the data point color.

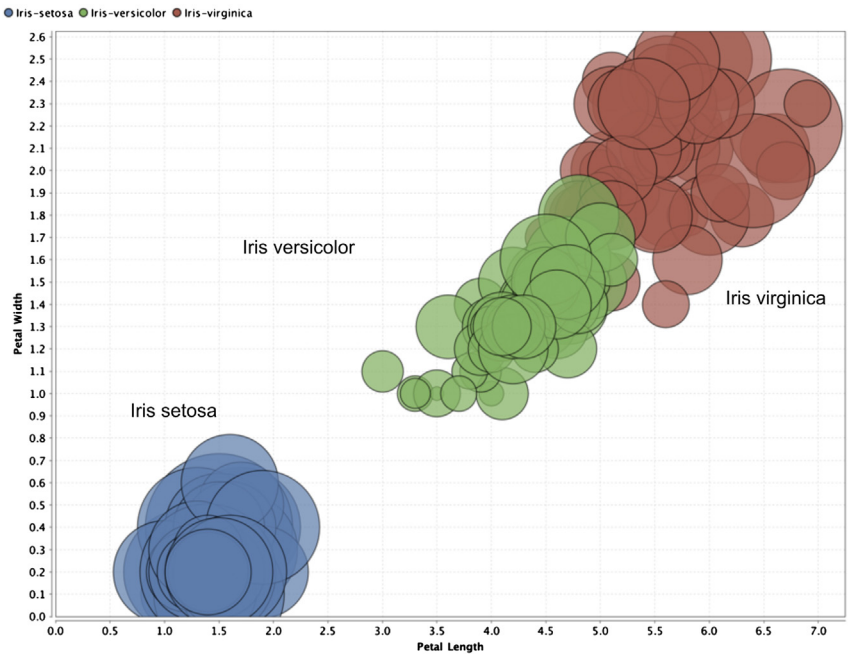


FIGURE 3.13
Bubble chart of Iris dataset.

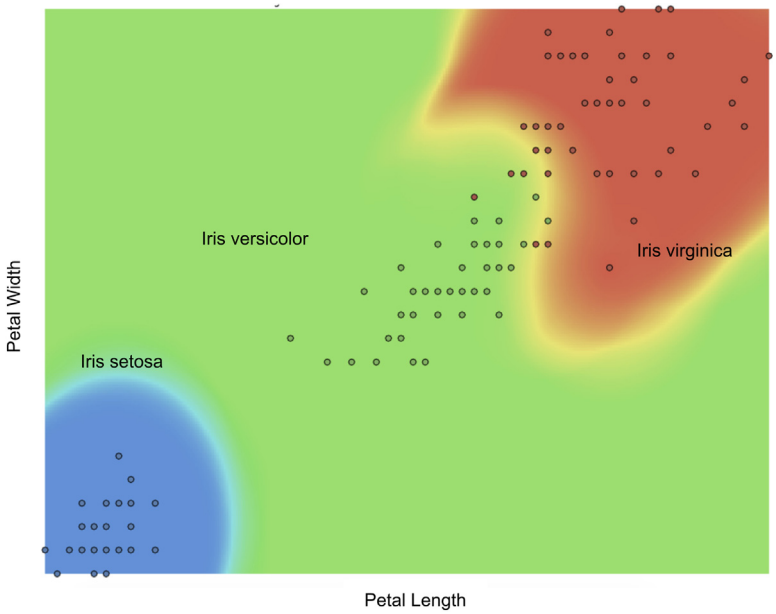


FIGURE 3.14
Density chart of a few attributes in the Iris dataset.

3.4.3 Visualizing High-Dimensional Data

Visualizing more than three attributes on a two-dimensional medium (like a paper or screen) is challenging. This limitation can be overcome by using transformation techniques to project the high-dimensional data points into parallel axis space. In this approach, a Cartesian axis is shared by more than one attribute.

Parallel Chart

A *parallel chart* visualizes a data point quite innovatively by transforming or projecting multi-dimensional data into a two-dimensional chart medium. In this chart, every attribute or dimension is linearly arranged in one coordinate (x-axis) and all the measures are arranged in the other coordinate (y-axis). Since the x-axis is multivariate, each data point is represented as a *line* in a parallel space.

In the case of the Iris dataset, all four attributes are arranged along the x-axis. The y-axis represents a generic distance and it is "shared" by all these attributes on the x-axis. Hence, parallel charts work only when attributes share a common unit of numerical measure or when the attributes are normalized. This visualization is called a *parallel axis* because all four attributes are represented in four parallel axes parallel to the y-axis.

In a parallel chart, a class label is used to color each data *line* so that one more dimension is introduced into the picture. By observing this parallel chart in Fig. 3.15, it can be noted that there is overlap between the three species on the sepal width attribute. So, sepal width cannot be the metric used to differentiate these three species. However, there is clear separation of species in petal length. No observation of *I. setosa* species has a petal length above 2.5 cm and there is little overlap between the *I. virginica* and *I. versicolor* species. Visually, just by knowing the petal length of an unlabeled observation, the species of Iris flower can be predicted. The relevance of this rule as a predictor will be discussed in the later chapter on Classification.

Deviation Chart

A *deviation chart* is very similar to a *parallel chart*, as it has parallel axes for all the attributes on the x-axis. Data points are extended across the dimensions as lines and there is one common y-axis. Instead of plotting all data lines, deviation charts only show the mean and standard deviation statistics. For each class, deviation charts show the mean line connecting the mean of each attribute; the standard deviation is shown as the band above and below the mean line. The mean line does not have to correspond to a data point (line). With this method, information is elegantly displayed, and the essence of a parallel chart is maintained.

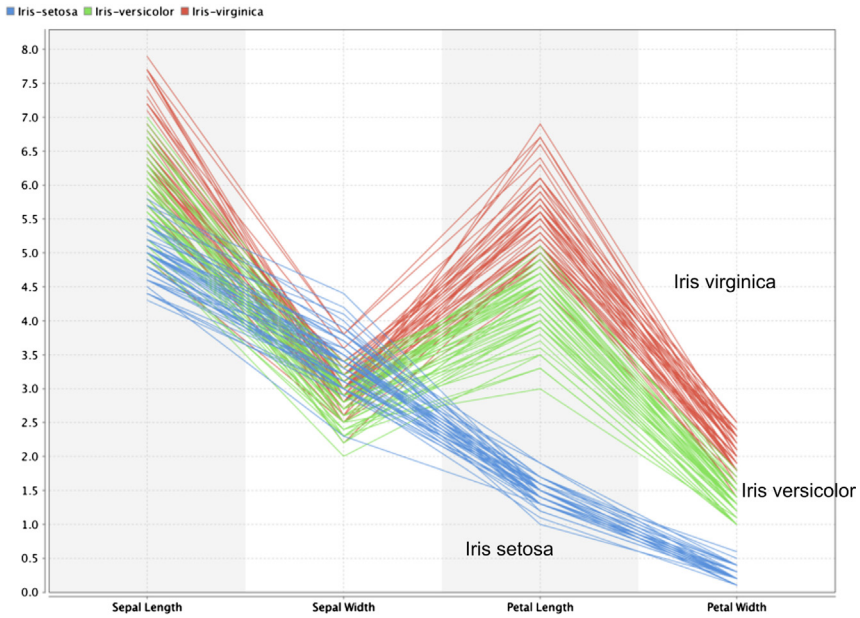


FIGURE 3.15
Parallel chart of Iris dataset.

In Fig. 3.16, a deviation chart for the Iris dataset stratified by species is shown. It can be observed that the petal length is a good predictor to classify the species because the mean line and the standard deviation bands for the species are well separated.

Andrews Curves

An *Andrews plot* belongs to a family of visualization techniques where the high-dimensional data are projected into a vector space so that each data point takes the form of a line or curve. In an Andrews plot, each data point X with d dimensions, $X = (x_1, x_2, x_3, \ldots, x_d)$, takes the form of a Fourier series:

$$f_x(t) = \frac{x_1}{\sqrt{2}} + x_2\sin(t) + x_3\cos(t) + x_4\sin(2t) + x_5\cos(2t) + \cdots \tag{3.4}$$

This function is plotted for $-\pi < t < \pi$ for each data point. Andrews plots are useful to determine if there are any outliers in the data and to identify potential patterns within the data points (Fig. 3.17). If two data points are similar, then the curves for the data points are closer to each other. If curves are far apart and belong to different classes, then this information can be used classify the data (Garcia-Osorio & Fyfe, 2005).

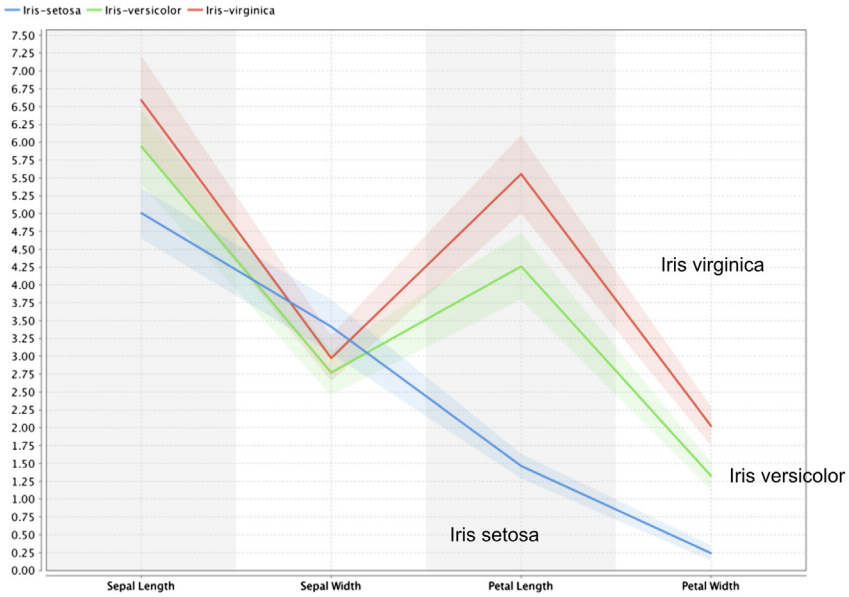


FIGURE 3.16
Deviation chart of Iris dataset.

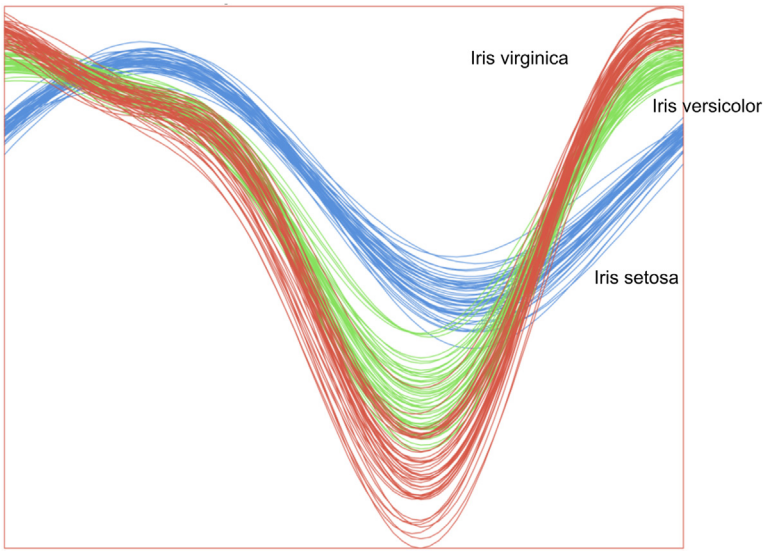


FIGURE 3.17
Andrews curves of Iris dataset.

Many of the charts and visuals discussed in this chapter explore the multivariate relationships within the dataset. They form the set of classic data visualizations used for data exploration, postprocessing, and understanding data science models. Some new developments in the area of visualization deals with networks and connections within the data objects (Lima, 2011). To better analyze data extracted from graph data, social networks, and integrated applications, connectivity charts are often used. Interactive exploration of data using visualization software provides an essential tool to observe multiple attributes at the same time but has limitations on the number of attributes used in visualizations. Hence, dimensional reduction using techniques discussed in Chapter 14, Feature Selection, can help in visualizing higher-dimensional data by reducing the dimensions to a critical few.

3.5 ROADMAP FOR DATA EXPLORATION

If there is a new dataset that has not been investigated before, having a structured way to explore and analyze the data will be helpful. Here is a roadmap to inquire a new dataset. Not all steps may be relevant for every dataset and the order may need to be adjusted for some sets, so this roadmap is intended as a guideline.

1. *Organize the dataset*: Structure the dataset with standard rows and columns. Organizing the dataset to have objects or instances in rows and dimensions or attributes in columns will be helpful for many data analysis tools. Identify the target or "class label" attribute, if applicable.
2. *Find the central point for each attribute*: Calculate *mean*, *median*, and *mode* for each attribute and the class label. If all three values are very different, it may indicate the presence of an outlier, or a multimodal or nonnormal distribution for an attribute.
3. *Understand the spread of each attribute*: Calculate the *standard deviation* and *range* for an attribute. Compare the standard deviation with the mean to understand the spread of the data, along with the max and min data points.
4. *Visualize the distribution of each attribute*: Develop the *histogram* and *distribution* plots for each attribute. Repeat the same for class-stratified histograms and distribution plots, where the plots are either repeated or color-coded for each class.
5. *Pivot the data*: Sometimes called dimensional slicing, a pivot is helpful to comprehend different values of the attributes. This technique can stratify by class and drill down to the details of any of the attributes. Microsoft Excel and Business Intelligence tools popularized this technique of data analysis for a wider audience.

CHAPTER 4

Classification

Enter the realm of data science—the process in which historical records are used to make a prediction about an uncertain future. At a fundamental level, most data science problems can be categorized into either class or numeric prediction problems. In classification or class prediction, one should try to use the information from the predictors or independent variables to sort the data samples into two or more distinct *classes* or *buckets*. In the case of numeric prediction, one would try to predict the numeric value of a dependent variable using the values assumed by the independent variables.

Here the classification process will be described with a simple example. Most golfers enjoy playing if the weather and outlook conditions meet certain requirements: too hot or too humid conditions, even if the outlook is sunny, are not preferred. On the other hand, overcast skies are no problem for playing even if the temperatures are somewhat cool. Based on the historic fictional records of these conditions and preferences, and information about a day's temperature, humidity level, and outlook, classification will allow one to predict if someone prefers to play golf or not. The outcome of classification is to categorize the weather conditions when golf is likely to be played or not, quite simply: *Play* or *Not Play* (two classes). The predictors can be continuous (temperature, humidity) or categorical (sunny, cloudy, windy, etc.). Those beginning to explore data science are confused by the dozens of techniques that are available to address these types of classification problems. In this chapter, several commonly used data science techniques will be described where the idea is to develop rules, relationships, and models based on predictor information that can be applied to classify outcomes from new and unseen data.

To begin with, fairly simple schemes will be used with a progression to more sophisticated techniques. Each section contains essential algorithmic details about the technique, describes how it is developed using simple examples, and finally closes with implementation details.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00004-6

4.1 DECISION TREES

Decision trees (also known as classification trees) are probably one of the most intuitive and frequently used data science techniques. From an analyst's point of view, they are easy to set up and from a business user's point of view they are easy to interpret. Classification trees, as the name implies, are used to separate a dataset into classes belonging to the response variable. Usually the response variable has two classes: Yes or No (1 or 0). If the response variable has *more* than two categories, then variants of the decision tree algorithm have been developed that may be applied (Quinlan, 1986). In either case, classification trees are used when the response or target variable is categorical in nature.

Regression trees (Brieman, 1984) are similar in function to classification trees and are used for numeric prediction problems, when the response variable is numeric or continuous: for example, predicting the price of a consumer good based on several input factors. Keep in mind that in either case, the predictors or independent variables may be either categorical or numeric. It is the *target variable* that determines the type of decision tree needed.

4.1.1 How It Works

A decision tree model takes a form of decision flowchart (or an inverted tree) where an attribute is tested in each node. At end of the decision tree path is a leaf node where a prediction is made about the target variable based on conditions set forth by the decision path. The nodes split the dataset into subsets. In a decision tree, the idea is to *split* the dataset based on the homogeneity of data. Say for example, there are two variables, age and weight, that predict if a person is likely to sign up for a gym membership or not. In the training data if it was seen that 90% of the people who are older than 40 signed up, the data can be split into two parts: one part consisting of people older than 40 and the other part consisting of people under 40. The first part is now "90% pure" from the standpoint of which class they belong to. However, a rigorous measure of impurity is needed, which meets certain criteria, based on computing a proportion of the data that belong to a class. These criteria are simple:

1. The measure of impurity of a dataset must be at a maximum when all possible classes are equally represented. In the gym membership example, in the initial dataset, if 50% of samples belonged to "not signed up" and 50% of the samples belonged to "signed up," then this non-partitioned raw data would have maximum impurity.
2. The measure of impurity of a dataset must be zero when only one class is represented. For example, if a group is formed of only those people who signed up for the membership (only one class = members), then this subset has "100% purity" or "0% impurity."

Measures such as *entropy* or *Gini index* easily meet these criteria and are used to build decision trees as described in the following sections. Different criteria will build different trees through different biases, for example, *information gain* favors tree splits that contain many cases, while *information gain ratio* attempts to balance this.

HOW DATA SCIENCE CAN REDUCE UNCERTAINTY

Imagine a box that can contain one of three colored balls inside—red, yellow, and blue, see Fig. 4.1. Without opening the box, if one had to "predict" which colored ball is inside, then they are basically dealing with a lack of information or uncertainty. Now what is the *highest* number of "yes/no" questions that can be asked to reduce this uncertainty and, thus, increase our information?

1. Is it red? No.
2. Is it yellow? No.

Then it must be blue.

That is *two* questions. If there were a fourth color, green, then the highest number of yes/no questions is *three*. By extending this reasoning, it can be mathematically shown that the maximum number of *binary* questions needed to reduce uncertainty is essentially $\log(T)$, where the log is taken to base 2 and T is the number of possible outcomes (Meagher, 2005) [e.g., if there was only one color, that is, one outcome, then $\log(1) = 0$, which means there is no uncertainty!].

Many real-world business problems can be thought of as extensions to this "uncertainty reduction" example. For example, knowing only a handful of characteristics such as the length of a loan, borrower's occupation, annual income, and previous credit behavior, several available data science techniques can be used to rank the riskiness of a potential loan, and by extension, the interest rate of the loan. This is nothing but a more sophisticated uncertainty reduction exercise, similar in spirit to the ball-in-a-box problem. Decision trees embody this problem-solving technique by systematically examining the available attributes and their impact on the eventual class or category of a sample. Later in this section, how to predict the credit ratings of a bank's customers using their demographic and other behavioral data will be examined in detail using a decision tree, which is a practical implementation of the entropy principle for decision-making under uncertainty.

FIGURE 4.1
Playing 20 questions with entropy.

Continuing with the example in the box, if there are T events with equal probability of occurrence P, then $T = 1/P$. Claude Shannon, who developed the mathematical underpinnings for information theory (Shannon, 1948), defined entropy as $\log_2(1/p)$ or $-\log_2 p$ where p is the probability of an event occurring. If the probability for all events is not identical, a weighted expression is needed and, thus, entropy, H, is adjusted as follows:

$$H = -\sum_{k=1}^{m} p_k \log_2(p_k) \tag{4.1}$$

where $k = 1, 2, 3, \ldots, m$ represents the m classes of the target variable. p_k represents the proportion of samples that belong to class k. For the gym membership example from earlier, there are two classes: member or nonmember. If the dataset had 100 samples with 50% of each, then the entropy of the dataset is given by $H = -[(0.5 \log_2 0.5) + (0.5 \log_2 0.5)] = -\log_2 0.5 = -(-1) = 1$. On the other hand, if the data can be partitioned into two sets of 50 samples each that exclusively contain all members and all nonmembers, the entropy of either of these two partitioned sets is given by $H = -1 \log_2 1 = 0$. Any other proportion of samples within a dataset will yield entropy values between 0 and 1 (which is the maximum). The Gini index (G) is similar to the entropy measure in its characteristics and is defined as

$$G = 1 - \sum_{k=1}^{m} p_k^2 \tag{4.2}$$

The value of G ranges between 0 and a maximum value of 0.5, but otherwise has properties identical to H, and either of these formulations can be used to create partitions in the data (Cover, 1991).

the golf example, introduced earlier, is used in this section to explain the application of entropy concepts for creating a decision tree. This was the same dataset used by J. Ross Quinlan to introduce one of the original decision tree algorithms, the *Iterative Dichotomizer* 3, or ID3 (Quinlan, 1986). The full data are shown in Table 4.1.

There are essentially two questions that need to be answered at each step of the tree building process: *where to split the data* and *when to stop splitting*.

Step 1: Where to Split Data?

There are 14 examples, with four attributes—Outlook, Temperature, Humidity, and Wind. The target attribute that needs to be predicted is Play with two classes: Yes and No. It's important to understand how to build a decision tree using this simple dataset.

Table 4.1 The Classic Golf Dataset

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	false	no
Sunny	80	90	true	no
Overcast	83	78	false	yes
Rain	70	96	false	yes
Rain	68	80	false	yes
Rain	65	70	true	no
Overcast	64	65	true	yes
Sunny	72	95	false	no
Sunny	69	70	false	yes
Rain	75	80	false	yes
Sunny	75	70	true	yes
Overcast	72	90	true	yes
Overcast	81	75	false	yes
Rain	71	80	true	no

Start by partitioning the data on each of the four regular attributes. Let us start with Outlook. There are three categories for this variable: sunny, overcast, and rain. We see that when it is overcast, there are four examples where the outcome was Play = yes for all four cases (see Fig. 4.2) and so the proportion of examples in this case is 100% or 1.0. Thus, if we split the dataset here, the resulting four sample partition will be 100% pure for Play = yes. Mathematically for this partition, the entropy can be calculated using Eq. (4.1) as:

$$H_{\text{outlook:overcast}} = -(0/4)\log_2(0/4) - (4/4)\log_2(4/4) = 0.0$$

Similarly, the entropy in the other two situations for Outlook can be calculated:

$$H_{\text{outlook:sunny}} = -(2/5)\log_2(2/5) - (3/5)\log_2(3/5) = 0.971$$
$$H_{\text{outlook:rain}} = -(3/5)\log_2(3/5) - (2/5)\log_2(2/5) = 0.971$$

For the attribute on the whole, the total *information I* is calculated as the weighted sum of these component entropies. There are four instances of Outlook = overcast, thus, the proportion for overcast is given by $p_{\text{outlook:overcast}} = 4/14$. The other proportions (for Outlook = sunny and rain) are 5/14 each:

$$I_{\text{outlook}} = P_{\text{outlook:overcast}} \times H_{\text{outlook:overcast}} + P_{\text{outlook:sunny}} \times H_{\text{outlook:sunny}} + P_{\text{outlook:rain}} \times H_{\text{outlook:rain}}$$
$$I_{\text{outlook}} = (4/14) \times 0 + (5/14) \times 0.971 + (5/14) \times 0.971 = 0.693$$

Row No.	Play	Outlook
1	no	sunny
2	no	sunny
3	yes	overcast
4	yes	rain
5	yes	rain
6	no	rain
7	yes	overcast
8	no	sunny
9	yes	sunny
10	yes	rain
11	yes	sunny
12	yes	overcast
13	yes	overcast
14	no	rain

FIGURE 4.2
Splitting the data on the Outlook attribute.

Had the data *not* been partitioned along the three values for Outlook, the total information would have been simply the weighted average of the respective entropies for the two classes whose overall proportions were 5/14 (Play = no) and 9/14 (Play = yes):

$$I_{\text{outlook,no partition}} = -(5/14)\log_2(5/14) - (9/14)\log_2(9/14) = 0.940$$

By creating these splits or partitions, some entropy has been reduced (and, thus, some information has been gained). This is called, aptly enough, *information gain*. In the case of Outlook, this is given simply by:

$$I_{\text{outlook, no partition}} - I_{\text{outlook}} = 0.940 - 0.693 = 0.247$$

Similar information gain values for the other three attributes can now be computed, as shown in Table 4.2.

For numeric variables, possible split points to examine are essentially averages of available values. For example, the first potential split point for Humidity could be Average [65,70], which is 67.5, the next potential split point could be Average [70,75], which is 72.5, and so on. Similar logic can

Table 4.2 Computing the Information Gain for All Attributes

Attribute	Information Gain
Temperature	0.029
Humidity	0.102
Wind	0.048
Outlook	0.247

be used for the other numeric attribute, Temperature. The algorithm computes the information gain at each of these potential split points and chooses the one which maximizes it. Another way to approach this would be to discretize the numerical ranges, for example, Temperature $>= 80$ could be considered "Hot," between 70 and 79 "Mild," and less than 70 "Cool."

From Table 4.2, it is clear that if the dataset is partitioned into three sets along the three values of Outlook, the largest information gain would be experienced. This gives the first node of the decision tree as shown in Fig. 4.3. As noted earlier, the terminal node for the Outlook = overcast branch consists of four samples, all of which belong to the class Play = yes. The other two branches contain a mix of classes. The Outlook = rain branch has three yes results and the Outlook = sunny branch has three no results.

Thus, not all the final partitions are 100% homogenous. This means that the same process can be applied for each of these subsets till purer results are obtained. So, back to the first question once again—where to split the data? Fortunately, this was already answered for when the information gain for all attributes was computed. The other attributes, that yielded the highest gains, are used. Following that logic, the Outlook = sunny branch can be split along Humidity (which yielded the second highest information gain) and the Outlook = rain branch can be split along Wind (which yielded the third highest gain). The fully grown tree shown in Fig. 4.4 does precisely that.

Step 2: When to Stop Splitting Data?

In real-world datasets, it is very unlikely that to get terminal nodes that are 100% homogeneous as was just seen in the golf dataset. In this case, the algorithm would need to be instructed when to stop. There are several situations where the process can be terminated:

1. No attribute satisfies a minimum information gain threshold (such as the one computed in Table 4.2).
2. A maximal depth is reached: as the tree grows larger, not only does interpretation get harder, but a situation called "overfitting" is induced.
3. There are less than a certain number of examples in the current subtree: again, a mechanism to prevent *overfitting*.

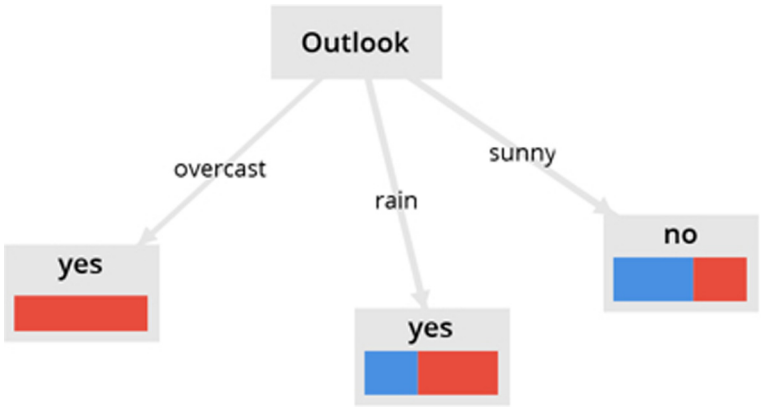


FIGURE 4.3
Splitting the Golf dataset on the Outlook attribute yields three subsets or branches. The middle and right branches may be split further.

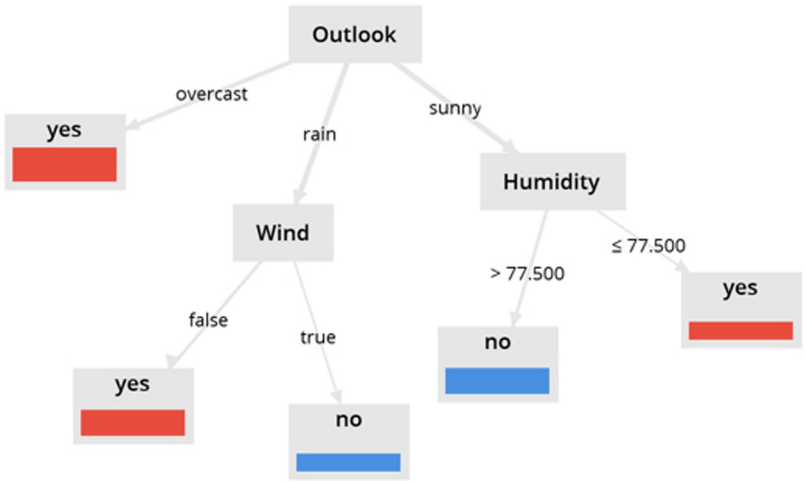


FIGURE 4.4
Decision Tree for the Golf data.

So, what exactly is overfitting? Overfitting occurs when a model tries to memorize the training data instead of generalizing the relationship between inputs and output variables. Overfitting often has the effect of performing well on the training dataset but performing poorly on any new data previously unseen by the model. As mentioned, overfitting by a decision tree results not only in difficulty interpreting the model, but also provides quite a useless model for unseen data. To prevent overfitting, tree growth may need

to be restricted or reduced, using a process called *pruning*. All three stopping techniques mentioned constitute what is known of as *pre-pruning* the decision tree, because the pruning occurs before or during the growth of the tree. There are also methods that will not restrict the number of branches and allow the tree to grow as deep as the data will allow, and *then* trim or prune those branches that do not effectively change the classification error rates. This is called *post-pruning*. Post-pruning may sometimes be a better option because one will not miss any small but potentially significant relationships between attribute values and classes if the tree is allowed to reach its maximum depth. However, one drawback with post-pruning is that it requires additional computations, which may be wasted when the tree needs to be trimmed back.

Now the application of the decision tree algorithm can be summarized with this simple five-step process:

1. Using Shannon entropy, sort the dataset into homogenous (by class) and non-homogeneous variables. Homogeneous variables have low information entropy and non-homogeneous variables have high information entropy. This was done in the calculation of $I_{\text{outlook, no partition}}$.
2. Weight the influence of each independent variable on the target variable using the entropy weighted averages (sometimes called joint entropy). This was done during the calculation of I_{outlook} in the example.
3. Compute the information gain, which is essentially the reduction in the entropy of the target variable due to its relationship with each independent variable. This is simply the difference between the information entropy found in step 1 minus the joint entropy calculated in step 2. This was done during the calculation of $I_{\text{outlook, no partition}} - I_{\text{outlook}}$.
4. The independent variable with the highest information gain will become the root or the first node on which the dataset is divided. This was done using the calculation of the information gain table.
5. Repeat this process for each variable for which the Shannon entropy is nonzero. If the entropy of a variable is zero, then that variable becomes a "leaf" node.

4.1.2 How to Implement

Before jumping into a business use case of decision trees, a simple decision tree model will be implemented using the concepts discussed in the earlier section. The first implementation gives an idea about key building blocks in a data science implementation process. The second implementation provides a deep-dive into a business application. This is the first implementation of a

data science technique, so some extra effort will be spent going into detail on many of the preliminary steps and also on introducing several additional tools and concepts that will be required throughout the rest of this chapter and other chapters that focus on supervised learning methods. These are the concepts of splitting data into testing and training samples and applying the trained model on testing. It may also be useful to first review Section 15.1 (Introduction to the GUI) and Section 15.2 (Data Import and Export) from Chapter 15, Getting started with RapidMiner before working through the rest of this implementation. As a final note, the ways and means to improve the performance of a classification model using RapidMiner will not be discussed in this section, but this very important part of data science will be revisited in several later chapters, particularly in the section on using optimization.

Implementation 1: To Play Golf or Not?

The complete RapidMiner process for implementing the decision tree model discussed in the earlier section is shown in Fig. 4.5. The key building blocks for this process are: training dataset, test dataset, model building, predicting using the model, predicted dataset, model representation, and performance vector.

The decision tree process has two input datasets. The *training dataset*, shown in Table 4.1, is used to build the decision tree with default parameter options. Fig. 4.6 shows the *test dataset*. The test dataset shares the same structure as the training dataset but with different records. These two operators constitute the inputs to the data science process.

The *modeling* block builds the decision tree using the training dataset. The *Apply model* block predicts the class label of the test dataset using the developed model and appends the predicted label to the dataset. The predicted dataset is one of the three outputs of the process and is shown in Fig. 4.7.

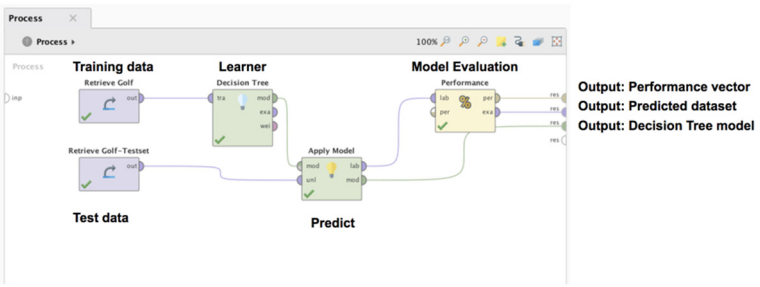


FIGURE 4.5
Building blocks of the Decision Tree process.

Row No.	Play	Outlook	Temperature	Humidity	Wind
1	yes	sunny	85	85	false
2	no	overcast	80	90	true
3	yes	overcast	83	78	false
4	yes	rain	70	96	false
5	yes	rain	68	80	true
6	no	rain	65	70	true
7	yes	overcast	64	65	true
8	no	sunny	72	95	false
9	yes	sunny	69	70	false
10	no	sunny	75	80	false
11	yes	sunny	68	70	true
12	yes	overcast	72	90	true
13	no	overcast	81	75	true
14	yes	rain	71	80	true

FIGURE 4.6
Test data.

Row ...	Play	prediction...	confidence(...	confidence(...	Outlook	Temperature	Humidity	Wind
1	yes	no	1	0	sunny	85	85	false
2	no	yes	0	1	overcast	80	90	true
3	yes	yes	0	1	overcast	83	78	false
4	yes	yes	0	1	rain	70	96	false
5	yes	no	1	0	rain	68	80	true
6	no	no	1	0	rain	65	70	true
7	yes	yes	0	1	overcast	64	65	true
8	no	no	1	0	sunny	72	95	false
9	yes	yes	0	1	sunny	69	70	false
10	no	no	1	0	sunny	75	80	false
11	yes	yes	0	1	sunny	68	70	true
12	yes	yes	0	1	overcast	72	90	true
13	no	yes	0	1	overcast	81	75	true
14	yes	no	1	0	rain	71	80	true

FIGURE 4.7
Results of applying the simple Decision Tree model.

Note the prediction test dataset has both the predicted and the original class label. The model has predicted correct class for nine of the records, but not for all. The five incorrect predictions are highlighted in Fig. 4.7.

The decision tree model developed using the training dataset is shown in Fig. 4.8. This is a simple decision tree with only three nodes. The leaf nodes are pure with a clean split of data. In practical applications, the tree will have dozens of nodes and the split will have mixed classes in the leaf nodes.

The *performance evaluation* block compares the predicted class label and the original class label in the test dataset to compute the performance metrics like accuracy, recall, etc. Fig. 4.9 shows the accuracy results of the model and the confusion matrix. It is evident that the model has been able to get 9 of the 14 class predictions correct and 5 of the 14 (in boxes) wrong, which translates to about 64% accuracy.

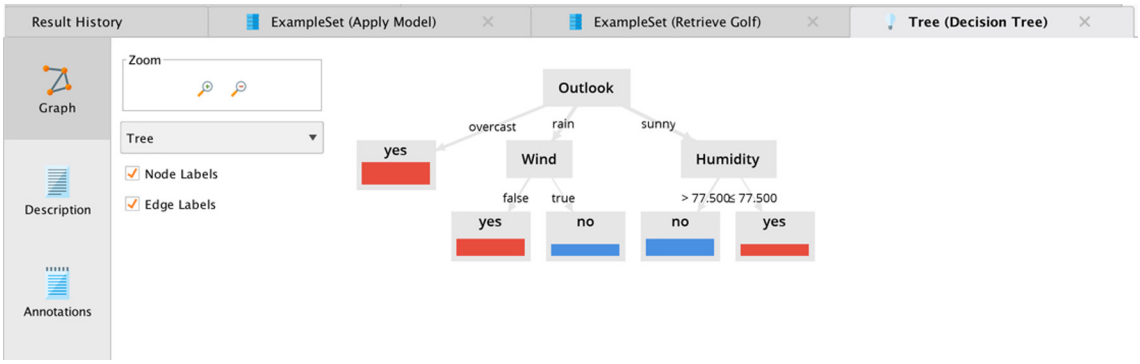


FIGURE 4.8
Decision Tree for Golf dataset.

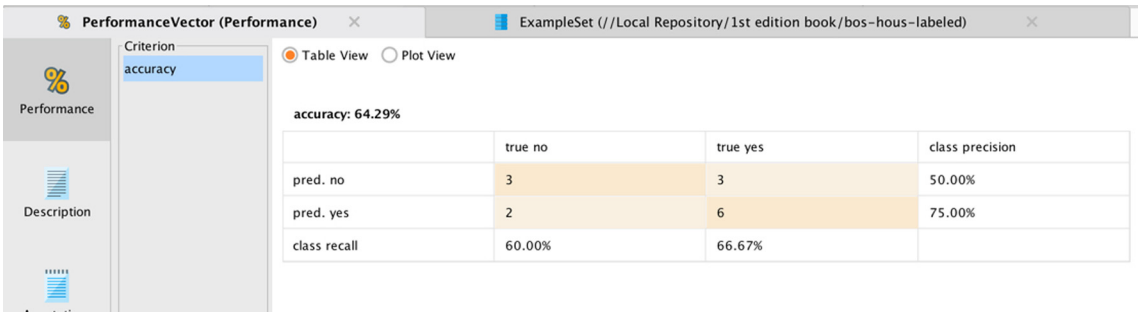


FIGURE 4.9
Performance vector.

Implementation 2: Prospect Filtering

A more involved business application will be examined to better understand how to apply decision trees for real-world problems. Credit scoring is a fairly common data science problem. Some types of situations where credit scoring could be applied are:

1. Prospect filtering: Identify which prospects to extend credit to and determine how much credit would be an acceptable risk.
2. Default risk detection: Decide if a particular customer is likely to default on a loan.
3. Bad debt collection: Sort out those debtors who will yield a good cost (of collection) to benefit (of receiving payment) performance.

The German Credit dataset from the University of California-Irvine Machine Learning (UCI-ML) data repository[1] will be used with RapidMiner to build a decision tree for addressing a *prospect filtering problem*. There are four main steps in setting up any supervised learning algorithm for a predictive modeling exercise:

1. Read in the cleaned and prepared data typically from a database or a spreadsheet, but the data can be from any source.
2. Split data into training and testing samples.
3. Train the decision tree using the training portion of the dataset.
4. Apply the model on the testing portion of the dataset to evaluate the performance of the model.

Step 1 may seem rather elementary but can confuse many beginners and, thus, sometime will be spent explaining this in somewhat more detail.

Step 1: Data Preparation

The raw data is in the format shown in Table 4.3. It consists of 1000 samples and a total of 20 attributes and 1 label or target attribute. There are seven numeric attributes and the rest are categorical or qualitative, including the label, which is a binominal variable. The label attribute is called Credit Rating and can take the value of 1 (good) or 2 (bad). In the data 70% of the samples fall into the good credit rating class. The descriptions for the data are shown in Table 4.3. Most of the attributes are self-explanatory, but the raw data has encodings for the values of the qualitative variables. For example, attribute 4 is the *purpose of the loan* and can assume any of 10 values (A40 for new car, A41 for used car, and so on). The full details of these encodings are provided under the dataset description on the UCI-ML website.

[1] http://archive.ics.uci.edu/ml/datasets/ All datasets used in this book are available at the companion website.

Table 4.3 A View of the Raw German Credit Data

Checking Account Status	Duration in Months	Credit History	Purpose	Credit Amount	Savings Account/ Bonds	Present Employment since	Credit Rating
A11	6	A34	A43	1169	A65	A75	1
A12	48	A32	A43	5951	A61	A73	2
A14	12	A34	A46	2096	A61	A74	1
A11	42	A32	A42	7882	A61	A74	1
A11	24	A33	A40	4870	A61	A73	2
A14	36	A32	A46	9055	A65	A73	1
A14	24	A32	A42	2835	A63	A75	1
A12	36	A32	A41	6948	A61	A73	1
A14	12	A32	A43	3059	A64	A74	1
A12	30	A34	A40	5234	A61	A71	2
A12	12	A32	A40	1295	A61	A72	2
A11	48	A32	A49	4308	A61	A72	2

RapidMiner's easy interface allows quick importing of spreadsheets. A useful feature of the interface is the panel on the left, called the *Operators*. Simply typing in text in the box provided automatically pulls up all available RapidMiner operators that match the text. In this case, an operator to needs to read an Excel spreadsheet, and so one would simply type excel in the box. Either double-click on the *Read Excel* operator or drag and drop it into the Main Process panel—the effect is the same. Once the *Read Excel* operator appears in the main process window as shown in Fig. 4.10, the data import process needs to be configured. What this means is telling RapidMiner which columns to import, what is contained in the columns, and if any of the columns need special treatment.

This is probably the most cumbersome part about this step. RapidMiner has a feature to automatically detect the type of values in each attribute (Guess Value types). But it is a good exercise for the analyst to make sure that the right columns are picked (or excluded) and the value types are correctly guessed. If not, as seen in Fig. 4.11, the value type can be changed to the correct setting by clicking on the button below the attribute name.

Once the data is imported, the target variable must be assigned for analysis, also known as a label. In this case, it is the Credit Rating. Finally, it is a good idea to run RapidMiner and generate results to ensure that all columns are read correctly.

An optional step is to convert the values from A121, A143, etc., to more meaningful qualitative descriptions. This is accomplished by the use of

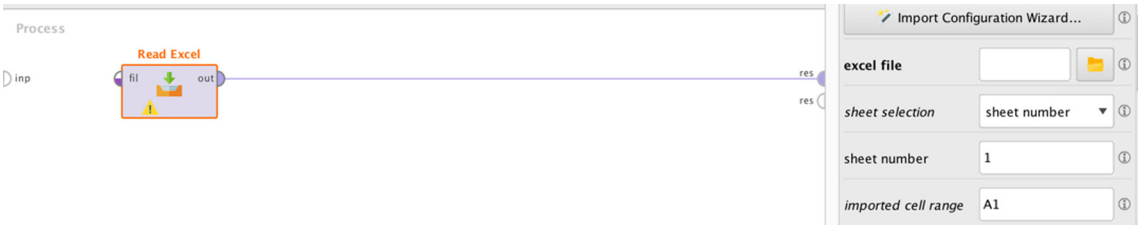


FIGURE 4.10
Using the Read Excel operator.

Checking Ac	Duration in	Credit Histo	Purpose	Credit Amo	Savings Acc	Present Em	Installment	Personal St	Other debtc	Present resi	Property	Age
polyn...	integer	polyn...	polyn...	integer	polyn...	polyn...	integer	polyn...	polyn...	integer	polyn...	integer
attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute
A11	6	A34	A43	1169	A65	A75	4	A93	A101	4	A121	67
A12	48	A32	A43	5951	A61	A73	2	A92	A101	2	A121	22
A14	12	A34	A46	2096	A61	A74	2	A93	A101	3	A121	49
A11	42	A32	A42	7882	A61	A74	2	A93	A103	4	A122	45
A11	24	A33	A40	4870	A61	A73	3	A93	A101	4	A124	53
A14	36	A32	A46	9055	A65	A73	2	A93	A101	4	A124	35
A14	24	A32	A42	2835	A63	A75	3	A93	A101	4	A122	53

FIGURE 4.11
Verifying data read-in and adjusting attribute value types if necessary.

another operator called *Replace (Dictionary)*, which will replace the values with bland encodings such as A121 and so on with more descriptive values. A dictionary will need to be created and supplied to RapidMiner as a comma-separated value (csv) file to enable this. Such a dictionary is easy to create and is shown in Fig. 4.12; Note that RapidMiner needs to be informed which column in the dictionary contains old values and which contain new values.

The last pre-processing step shown here is converting the numeric label into a binomial one by connecting the example output of *Replace (Dictionary)* to a *Numerical to Binominal* operator. To configure the *Numerical to Binominal* operator.

Finally, change the name of the label variable from Credit Rating to Credit Rating = Good so that it makes more sense when the integer values get converted to true or false after passing through the *Numerical to Binominal* operator. This can be done using the *Rename* operator. When this setup is run, the dataset shown in Fig. 4.13 will be generated. Comparing to Fig. 4.11, see

Row No.	OldValue	NewValue
1	A30	no credits taken all credits paid back duly
2	A31	all credits at this bank paid back duly
3	A32	existing credits paid back duly till now
4	A33	delay in paying off in the past
5	A34	critical account other credits existing (not at this bank)
6	A40	new car
7	A41	used car
8	A42	furniture equipment
9	A43	radio television
10	A44	domestic appliances
11	A45	repairs
12	A46	education

FIGURE 4.12
Attribute value replacement using a dictionary.

Row No.	Credit Rati...	Checking A...	Duration in...	Credit Hist...	Purpose	Credit Amo...	Savings Ac...	Present Em...	Installment...	Personal St...
1	false	Less than 0 ...	6	critical acco...	radio televis...	1169	unknown no...	Greater tha...	4	male single
2	true	0 to 200 DM	48	existing cre...	radio televis...	5951	Less than 1...	1 to 4 years	2	female divor...
3	false	no checking ...	12	critical acco...	education	2096	Less than 1...	4 to 7 years	2	male single
4	false	Less than 0 ...	42	existing cre...	furniture eq...	7882	Less than 1...	4 to 7 years	2	male single
5	true	Less than 0 ...	24	delay in pay...	new car	4870	Less than 1...	1 to 4 years	3	male single
6	false	no checking ...	36	existing cre...	education	9055	unknown no...	1 to 4 years	2	male single
7	false	no checking ...	24	existing cre...	furniture eq...	2835	500 to 100...	Greater tha...	3	male single
8	false	0 to 200 DM	36	existing cre...	used car	6948	Less than 1...	1 to 4 years	2	male single
9	false	no checking ...	12	existing cre...	radio televis...	3059	Greater tha...	4 to 7 years	2	male divorc...
10	true	0 to 200 DM	30	critical acco...	new car	5234	Less than 1...	unemployed	4	male marrie...

FIGURE 4.13
Data transformed for Decision Tree analysis.

that the label attribute is the first one shown and the values are *true* or *false*. The statistics tab of the results can be examined to get more information about the distributions of individual attributes and also to check for missing values and outliers. In other words, one must make sure that the data preparation step is properly executed before proceeding. In this implementation, there is no to worry about this because the dataset is relatively clean (for

instance, there are no missing values), and one could proceed directly to the model development phase.

Step 2: Divide dataset Into Training and Testing Samples

As with all supervised model building, data must be separated into two sets: one for training or developing an acceptable model, and the other for testing or ensuring that the model would work equally well on a different dataset. The standard practice is to split the available data into a training set and a testing set. Typically, the training set contains 70%–90% of the original data. The remainder is set aside for testing. The *Split Validation* operator sets up splitting, modeling, *and* the validation check in one operator. Choose *stratified sampling* with a split ratio of 0.9 (90% training). Stratified sampling[2] will ensure that both training and testing samples have equal distributions of class values. The final sub step here is to connect the output from the *Numerical to Binominal* operator output to the *Split Validation* operator input (see Fig. 4.14).

Step 3: Modeling Operator and Parameters

A demonstration of how to build a decision tree model on this data will now be given. The *Validation* operator allows one to build a model and apply it on validation data in the same step. This means that two operations—model building and model evaluation—must be configured using the same operator. This is accomplished by double-clicking on the Validation operator, which is what is called a *nested* operator. When this operator is opened, note that there are two parts inside (see Fig. 4.15). The left box is where the *Decision Tree* operator has to be placed and the model will be built using the 90% of training data samples. The right box is for applying this trained model on the remaining 10% of the testing data samples using the *Apply Model* operator and evaluating the performance of the model using the *Performance* operator.

Step 4: Configuring the Decision Tree Model

The main parameters to pay attention to are the Criterion pull-down menu and the minimal gain box. This is essentially a partitioning criterion and offers information gain, Gini index, and gain ratio as choices. The first two criteria were covered earlier, and gain ratio will be briefly explained in the next section.

As discussed earlier in this chapter, decision trees are built up in a simple five-step process by increasing the information contained in the reduced

[2] Although not necessary, it is sometimes useful to check the *use local random seed* option, so that it is possible to compare models between different iterations. Fixing the random seed ensures that the same examples are chosen for training (and testing) subsets each time the process is run.

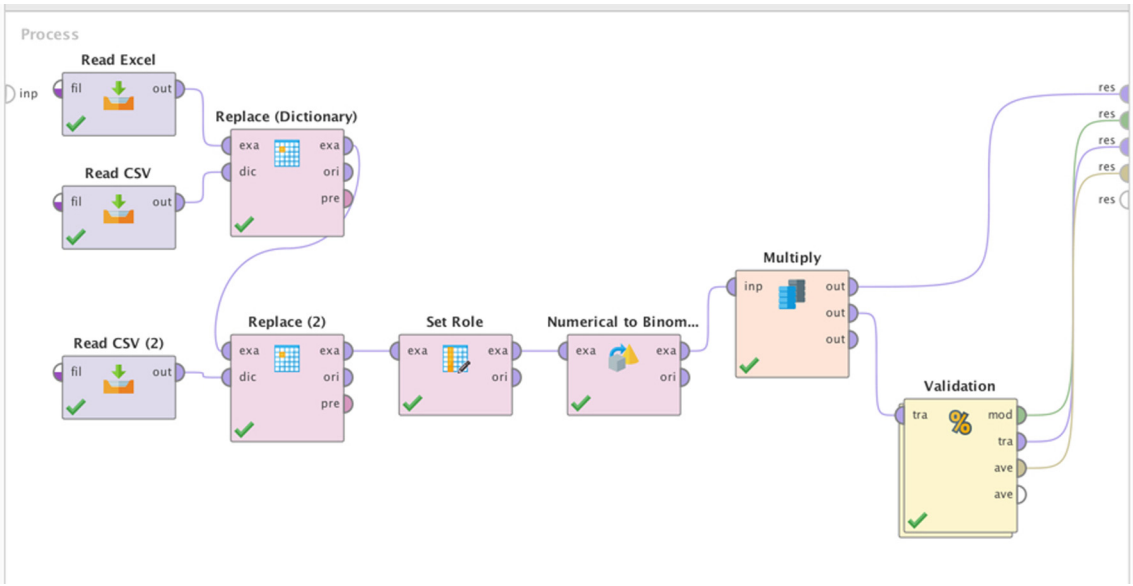


FIGURE 4.14
Decision Tree process.

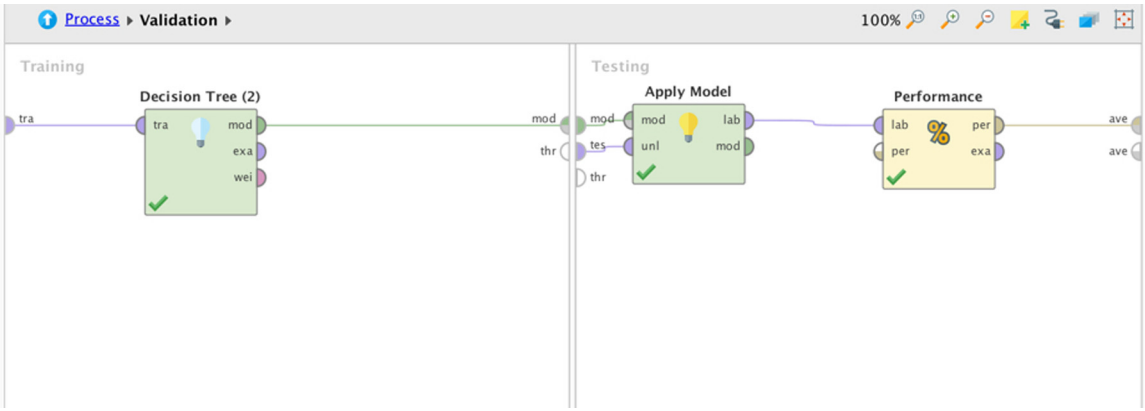


FIGURE 4.15
Setting up the split validation process.

dataset following each split. Data by its nature contains uncertainties. Uncertainties could possibly be systematically reduced, thus, increasing information by activities like sorting or classifying. When data have been sorted or classified to achieve the greatest reduction in uncertainty, basically, the greatest increase in information has been achieved. It has already been shown that entropy is a good measure of uncertainty and how keeping track of it allows information to be quantified. So, back to the options that are available within RapidMiner for splitting decision trees:

1. *Information gain:* This is computed as the information before the split minus the information after the split. It works fine for most cases, unless there are a few variables that have a large number of values (or classes). Information gain is *biased* toward choosing attributes with a large number of values as root nodes. This is not a problem, except in extreme cases. For example, each customer ID is unique and, thus, the variable has too many values (each ID is a unique value). A tree that is split along these lines has no predictive value.
2. *Gain ratio (default):* This is a modification of information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the *intrinsic information* of a split into account. Intrinsic information can be explained using the golf example. Suppose each of the 14 examples had a unique ID attribute associated with them. Then the intrinsic information for the ID attribute is given by $14 \times (-1/14 \times \log(1/14)) = 3.807$. The gain ratio is obtained by dividing the information gain for an attribute by its intrinsic information. Clearly attributes that have high intrinsic information (high uncertainty) tend to offer low gains upon splitting and, hence, would not be preferred in the selection process.
3. *Gini index:* This is also used sometimes but does not have too many advantages over gain ratio.
4. *Accuracy:* This is also used to improve performance. The best way to select values for these parameters is by using many of the optimizing operators.

The other important parameter is the *minimal gain* value. Theoretically this can take any range from 0 upwards. The default is 0.1. It has been set as 0.01 for this example.

The other parameters *minimal size for a split*, *minimal leaf size*, *maximal depth* are determined by the size of the dataset. In this case, the values have been set as 4,5, and 5 respectively. The model is ready for training. Next, two more operators are added, *Apply Model* and *Performance (Binominal*

Classification), and the analysis is ready to be run. Configure the *Performance (Binominal Classification)* operator by selecting the *accuracy*, area under ROC (receiver operator characteristic) curve (*AUC*), *precision*, and *recall* options.[3]

Remember to connect the ports correctly as this can be a source of confusion:

1. "mod"el port of the Testing window to "mod" on *Apply Model*
2. "tes"ting port of the Testing window to "unl"abeled on *Apply Model*
3. "lab"eled port of *Apply Model* to "lab"eled on *Performance*
4. "per"formance port on the *Performance* operator to "ave"rageable port on the output side of the testing box

The final step before running the model is to go back to the main process view (see Fig. 4.15) and connect the output ports model and "ave" of the *Validation* operator to the main process outputs.

Step 5: Process Execution and Interpretation

When the model is setup and run as explained, RapidMiner generates two tabs in the Results perspective. The *Performance Vector (Performance)* tab shows a confusion matrix that lists the model accuracy on the testing data, along with the other options selected above for the *Performance (Binominal Classification)* operator in step 3. The *Tree* (Decision Tree) tab shows a graphic of the tree that was built on the training data (see Fig. 4.16). Fig. 4.17 shows the tree model in the form of rules. Several important points must be highlighted before the performance of this model is discussed:

1. The root node—Checking Account Status—is the most important predictor in the dataset.
2. If the *Checking Account Status = No account*, a prediction can be made without the influence of other attributes.
3. For rest of the *Checking Account Status* values, other parameters come into effect and play an increasingly important role in deciding if someone is likely to have a *"good"* or *"bad"* credit rating.
4. Watch out for overfitting. Overfitting refers to the process of building a model specific to the training data that achieves close to full accuracy on the training data. However, when this model is applied to new data or if the training data changes somewhat, then there is a significant degradation in its performance. Overfitting is a potential issue with all supervised models, not just decision trees. One way this situation could be avoided is by changing the decision tree criterion *"Minimal leaf size"* to something like 10. But doing so, the classification influence of all the other parameters is also lost, except the root node.

[3] Performance criteria such as these are explained in more detail in Chapter 8, Model Evaluation.

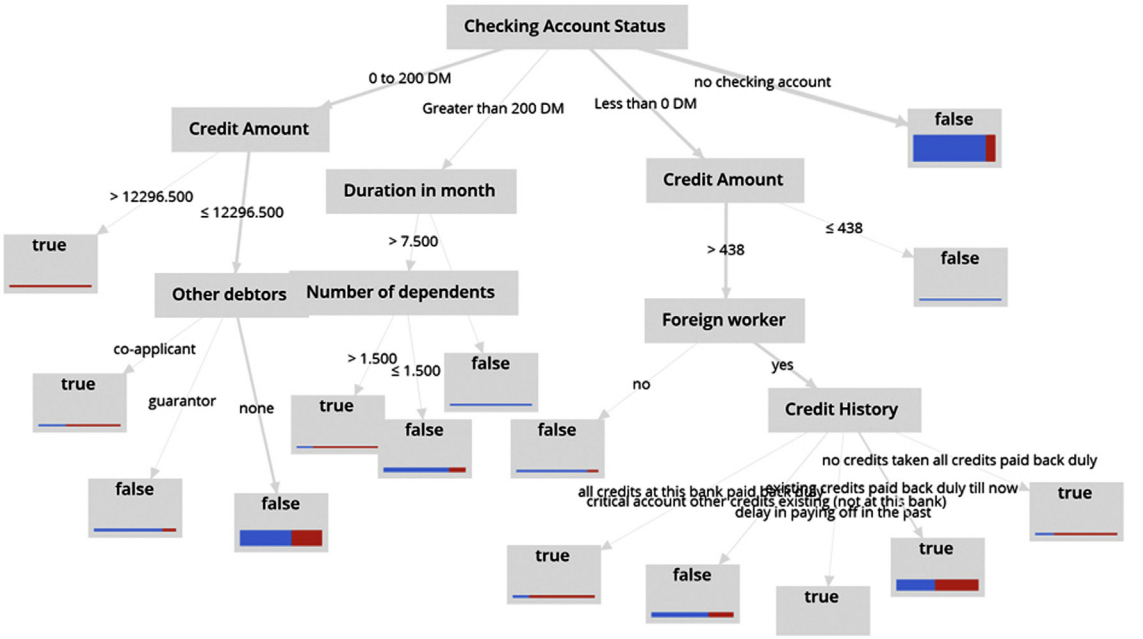


FIGURE 4.16
Decision Tree.

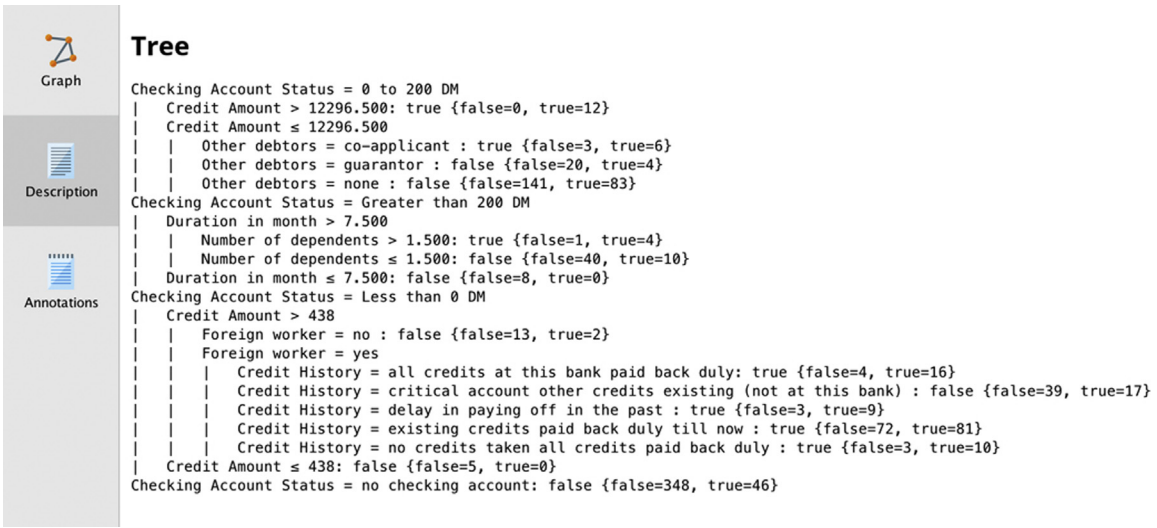


FIGURE 4.17
Decision Tree rules.

accuracy: 67.00%

	true false	true true	class precision
pred. false	65	28	69.89%
pred. true	5	2	28.57%
class recall	92.86%	6.67%	

FIGURE 4.18
Performance vector.

Now look at the Performance result. As seen in Fig. 4.18, the model's overall accuracy on the testing data is 67%. The model has a class recall of 92.86% for the "true" class implying that it is able to pick out customers with good credit rating with good accuracy. However, its class recall for the "false" class is an abysmal 6.67%! That is, the model can only pick out a potential defaulter in 1 out of 15 cases!

One way to improve this performance is by penalizing false negatives by applying a cost for every such instance. This is handled by another operator called *MetaCost*, which is described in detail in the next chapter on logistic regression. When a parameter search optimization is performed by iterating through three of the decision tree parameters, splitting criterion, minimum gain ratio, and maximal tree depth, significantly improved performance is gained. More details on how to set this type of optimization are provided in Chapter 15, Getting started with RapidMiner.

In addition to assessing the model's performance by aggregate measures such as accuracy, one can also use gain/lift charts, ROC charts, and AUC charts. An explanation of how these charts are constructed and interpreted is given in Chapter 8, Model Evaluation.

The RapidMiner process for a decision tree covered in the implementation section can be accessed from the companion site of the book at www.IntroDataScience.com. The RapidMiner process (*.rmp files) can be downloaded to one's computer and imported to RapidMiner through File > Import Process. Additionally, all the datasets used in this book can be downloaded from http://www.IntroDataScience.com.

4.1.3 Conclusion

Decision trees are one of the most commonly used predictive modeling algorithms in practice. The reasons for this are numerous. Some of the distinct advantages of using decision trees in many classification and prediction applications will be explained below along with some common pitfalls.

1. Easy to interpret and explain to non-technical users: As seen in the few examples discussed so far, decision trees are intuitive and easy to explain to non-technical people, who are typically the consumers of analytics.

2. Decision trees require relatively little effort from users for data preparation: If one has a dataset consisting of widely ranging attributes, for example, revenues recorded in millions and loan age recorded in years, many algorithms require scale normalization before model building and application. Such variable transformations are not required with decision trees because the tree structure will remain the same with or without the transformation. Another feature that saves data preparation time: missing values in training data will not impede partitioning the data for building trees. Decision trees are also not sensitive to outliers since the partitioning happens based on the proportion of samples within the split ranges and not on absolute values.
3. Nonlinear relationships between parameters do not affect tree performance. As described in Chapter 5, Regression Methods, highly nonlinear relationships between variables will result in failing checks for simple regression models, and thus, rendering such models invalid. However, decision trees do not require any assumptions of linearity in the data. Thus, one can use them in scenarios where one knows the parameters are nonlinearly related.
4. Decision trees implicitly perform variable screening or feature selection. When a decision tree is fitted to a training dataset, the top few nodes on which the tree is split are essentially the most important variables within the dataset and feature selection is completed automatically. In fact, RapidMiner has an operator for performing variable screening or feature selection using the information gain ratio. In Chapter 12, Time Series Forecasting, the importance of feature selection in data science will be discussed. A few common techniques for performing feature selection or variable screening will be introduced in that chapter.

However, all these advantages need to be tempered with the one key disadvantage of decision trees: without proper pruning or limiting tree growth, they tend to overfit the training data, making them somewhat poor predictors.

4.2 RULE INDUCTION

Rule induction is a data science process of deducing if-then rules from a dataset. These symbolic decision rules explain an inherent relationship between the attributes and class labels in a dataset. Many real-life experiences are based on intuitive rule induction. For example, one could come up with a rule that states that "if it is 8:00 a.m. on a weekday, then highway traffic will be heavy" and "if it is 8:00 p.m. on a Sunday, then the traffic will be light."

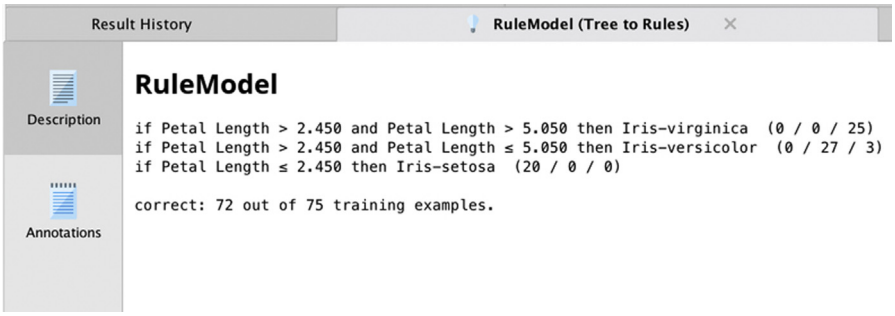


FIGURE 4.28
Rules based on Decision Tree.

The parameters for a decision tree are the same as reviewed in Section 4.1 of this chapter. The RapidMiner process can be saved and executed. The result set consists of a set of rule model, usually with repetitive conjuncts in antecedents, a fingerprint of rules derived from trees. Note the difference between the rules that are developed for the Rule Induction operator and the rules developed from *Tree to Rules* operator. The rules generated from Tree to Rules are shown in Fig. 4.28.

4.2.3 Conclusion

Classification using rules provides a simple framework to identify a relationship between attributes and the class label that is not only used as a predictive model, but also a descriptive model. Rules are closely associated to decision trees. They split the data space in a rectilinear fashion and generate a mutually exclusive and exhaustive rule set. When the rule set is not mutually exclusive, then the data space can be divided by complex and curved decision boundaries. Single rule learners are the simplest form of data science model and indicate the most powerful predictor in the given set of attributes. Since rule induction is a greedy algorithm, the result may not be the most globally optimal solution and like decision trees, rules can overlearn the example set. This scenario can be mitigated by pruning. Given the wide reach of rules, rule induction is commonly used as a tool to express the results of data science, even if other data science algorithms are used to create the model.

4.3 *k*-NEAREST NEIGHBORS

The predictive data science using decision trees and rule induction techniques were built by generalizing the relationship within a dataset and using it to predict the outcome of new unseen data. If one needs to predict the loan

interest rate based on credit score, income level, and loan amount, one approach is to develop a mathematical relationship such as an equation $y = f(X)$ based on the known data and then using the equation to predict the interest rate for new unseen data points. These approaches are called *eager learners* because they attempt to find the best approximation of the actual relationship between the input and target variables. But there is also a simple alternative to this approach. One can "predict" the interest rate for a potential borrower with a known credit score, income level, and loan amount by *looking up* the interest rate of other customer loan records with a similar credit score, a closely matching income level and loan amount from the training dataset. This alternative class of learners adopts a blunt approach, where no "learning" is performed from the training dataset; rather the training dataset is used as a lookup table to match the input variables and find the outcome. These approaches that memorize the training set are called *lazy learners*.

The underlying idea here is somewhat similar to the old adage, "birds of a feather flock together." Similar records congregate in a neighborhood in *n*-dimensional space, with the same target class labels. This is the central logic behind the approach used by the *k*-nearest neighbor algorithm, or simply referred *k*-NN. The entire training dataset is "memorized" and when unlabeled example records need to be classified, the input attributes of the new unlabeled records are compared against the entire training set to find the closest match. The class label of the closest training record is the predicted class label for the unseen test record. This is a nonparametric method, where no generalization or attempt to find the distribution of the dataset is made (Altman, 1992). Once the training records are in memory, the classification of the test record is straightforward. The closest training record needs to be found for each test record. Even though no mathematical generalization or rule generation is involved, finding the closet training record for a new unlabeled record can be a tricky problem to solve, particularly when there is no exact match of training data available for a given test data record.

PREDICTING THE TYPE OF FOREST

Satellite imaging and digital image processing have provided a wealth of data about almost every part of the earth's landscape. There is a strong motivation for forestry departments, government agencies, universities, and research bodies to understand the makeup of forests, species of trees and their health, biodiversity, density, and forest condition. Field studies for developing forest databases and classification projects are quite tedious and expensive tasks. However, this process can be aided with the leveraging of satellite imagery, limited field data, elevation models, aerial photographs, and survey data (McInerney, 2005). The objective is to classify whether the particular

(*Continued*)

(Continued)

landscape is a forest or not and further predict the type of trees and species.

The approach to classifying a landscape involves dividing the area into land units (e.g., a pixel in a satellite image) and creating a vector of all the measurements for the land unit. Each unit's measurements are then compared against the measurements of known preclassified units. For every new unclassified pixel, one can find a pixel in the pre-classified catalog, which has measurements similar to the measurement of the pixel to be predicted. Say the pre-classified pixel with the closest measurement corresponds to birch trees. Thus, the pixel area can be predicted to be a birch forest. Each pixel's measurement is compared to measurements of the preclassified dataset to determine the like pixels and, hence, the same forest types. This is the core concept of the *k*-NN algorithm that is used to classify landscape areas (Haapanen, Lehtinen, Miettinen, Bauer, & Ek, 2001).

4.3.1 How It Works

Any record in a dataset can be visualized as a point in an *n*-dimensional space, where *n* is the number of attributes. While it is hard for us to visualize in more than three dimensions, mathematical functions are scalable to any dimension and, hence, all the operations that can be done in two-dimensional spaces and be performed in *n*-dimensional space. Consider the standard Iris dataset (150 examples, four attributes, one class label. See Fig. 3.1 and Table 3.1) and focus on two petal attributes, petal length and petal width. The scatterplot of these two dimensions is shown in Fig. 4.29. The colors indicate the species of Iris, the target class variable. For an unseen test record A, with (petal length, petal width) values (2.1, 0.5), one can visually deduce that the predicted species for the values of data point A would be *I. setosa*. This is based on the fact that test data point A is in the neighborhood of other data points that belong to the species *I. setosa*. Similarly, unseen test data point B has values (5.7, 1.9) and is in the neighborhood of *I. virginica*, hence, the test data point can be classified as *I. virginica*. However, if the data points are in between the boundaries of two species, for data points such as (5.0, 1.8), then the classification can be tricky because the neighborhood has more than one species in the vicinity. An efficient algorithm is needed in order to resolve these corner cases and measure the nearness of data points with more than two dimensions. One technique is to find the nearest training data point from an unseen test data point in multi-dimensional space. That is how the *k*-NN algorithm works.

The *k* in the *k*-NN algorithm indicates the number of close training record(s) that need to be considered when making the prediction for an unlabeled test record. When $k = 1$, the model tries to find the *first* nearest record and adopts

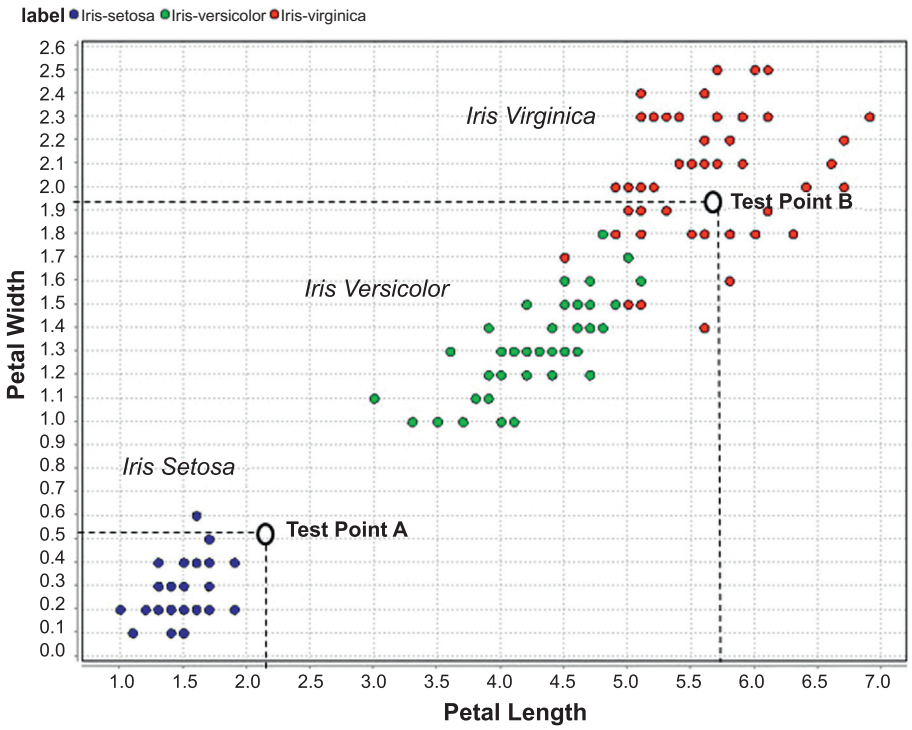


FIGURE 4.29

Two-dimensional plot of Iris dataset: petal length and petal width. Classes are stratified with colors.

the class label of the first nearest training record as the predicted target class value. Fig. 4.30 provides an example a training set with two dimensions and the target class values as circles and triangles. The unlabeled test record is the dark square in the center of the scatterplot. When $k = 1$, the predicted target class value of an unlabeled test record is *triangle* because the closest training record is a triangle. But, what if the closest training record is an outlier with the incorrect class in the training set? Then, all the unlabeled test records near the outlier will get wrongly classified. To prevent this misclassification, the value of k can be increased to, say, 3. When $k = 3$, the nearest *three* training records are considered instead of one. From Fig. 4.30, based on the majority class of the nearest three training records, the predicted class of the test record can be concluded as *circle*. Since the class of the target record is evaluated by voting, k is usually assigned an odd number for a two-class problem (Peterson, 2009).

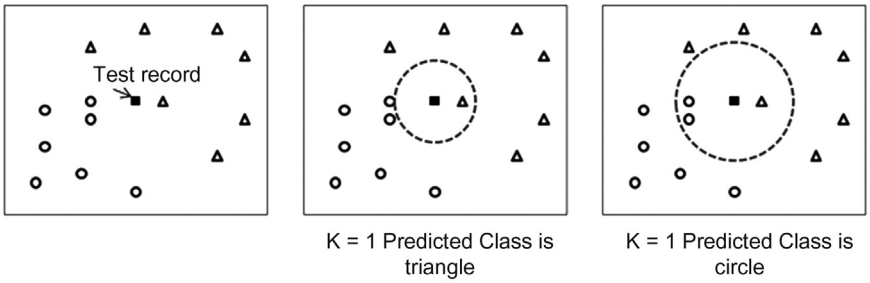


FIGURE 4.30

(A) Dataset with a record of unknown class. (B) Decision boundary with $k = 1$ around unknown class record. (C) Decision boundary with $k = 3$ around unknown test record.

The key task in the *k*-NN algorithm is determination of the *nearest* training record from the unlabeled test record using a measure of proximity. Once the nearest training record(s) are determined, the subsequent class voting of the nearest training records is straightforward. The various techniques used to measure proximity are discussed here.

Measure of Proximity

The effectiveness of the *k*-NN algorithm hinges on the determination of how similar or dissimilar a test record is when compared with the memorized training record. A measure of proximity between two records is a measure of the proximity of its attributes. To quantify similarity between two records, there is a range of techniques available such as calculating distance, correlation, Jaccard similarity, and cosine similarity (Tan et al., 2005).

Distance

The distance between two points X (x_1, x_2) and Y (y_1, y_2) in two-dimensional space can be calculated by Eq. (4.3):

$$\text{Distance } d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \tag{4.3}$$

One can generalize the two-dimensional distance formula shown in Eq. (4.3) for datasets with n attributes, where X is $(x_1, x_2, \ldots, x_n)$ and Y is $(y_1, y_2, \ldots, y_n)$, as:

$$\text{Distance } d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{4.4}$$

For example, the first two records of a four-dimensional Iris dataset is $X = (4.9, 3.0, 1.4, 0.2)$ and $Y = (4.6, 3.1, 1.5, 0.2)$. The distance between X and Y is: $d = \sqrt{(0.3)^2 + (0.1)^2 + (0.1)^2 + (0)^2} = 0.33$ centimeters.

All the attributes in the Iris dataset are homogenous in terms of measurements (size of the flower parts) and units (centimeters). However, in a typical practical dataset, it is common to see attributes in different measures (e.g., credit score, income) and varied units. One problem with the distance approach is that it depends on the scale and units of the attributes. For example, the difference in credit score between two records could be a few hundred points, which is minor in magnitude compared to the difference in income, which could be in the thousands. Consider two pairs of data points with credit score and annual income in USD. Pair A is (500, $40,000) and (600, $40,000). Pair B is (500, $40,000) and (500, $39,800). The first data point in both the pairs is same. The second data point is different from the first data point, with only one attribute changed. In Pair A, credit score is 600, which is significantly different to 500, while the income is the same. In Pair B, the income is down by $200 when compared to the first data point, which is only a 0.5% change. One can rightfully conclude that the data points in Pair B are more similar than the data points in Pair A. However, the distance [Eq. (4.4)] between data points in Pair A is 100 and the distance between Pair B is 200! The variation in income overpowers the variation in credit score. The same phenomenon can be observed when attributes are measured in different units, scales, etc. To mitigate the problem caused by different measures and units, all the inputs of *k*-NN are normalized, where the data values are rescaled to fit a particular range. Normalizing all the attributes provides a fair comparison between them. Normalization can be performed using a few different methods. Range transformation rescales all the values of the attributes to specified min and max values, usually 0 to 1. Z-transformation attempts to rescale all the values by subtracting the mean from each value and dividing the result by the standard deviation, resulting in a transformed set of values that have a mean of 0 and a standard deviation of 1. For example, when the Iris dataset is normalized using Z-transformation, sepal length, which takes values between 4.3 and 7.9 cm, and has a standard deviation of 0.828, is transformed to values between -1.86 and 2.84, with a standard deviation of 1. The distance measurement discussed so far is also called *Euclidean distance*, which is the most common distance measure for numeric attributes. In addition to the Euclidean, *Manhattan*, and *Chebyshev* distance measures are sometimes used to calculate the distance between two numeric data points. Consider two data points X (1,2) and Y (3,1), as shown in Fig. 4.31. The Euclidean distance between X and Y is the straight-line distance between X and Y, which is 2.7. The Manhattan distance is the sum of the difference between individual attributes, rather than the root of squared difference. The Manhattan distance between X and Y is $(3-1) + (2-1) = 3$. The Manhattan distance is also called the taxicab distance, due to the similarities of the visual path traversed by a vehicle around city blocks (In Fig. 4.31, the total distance that is covered

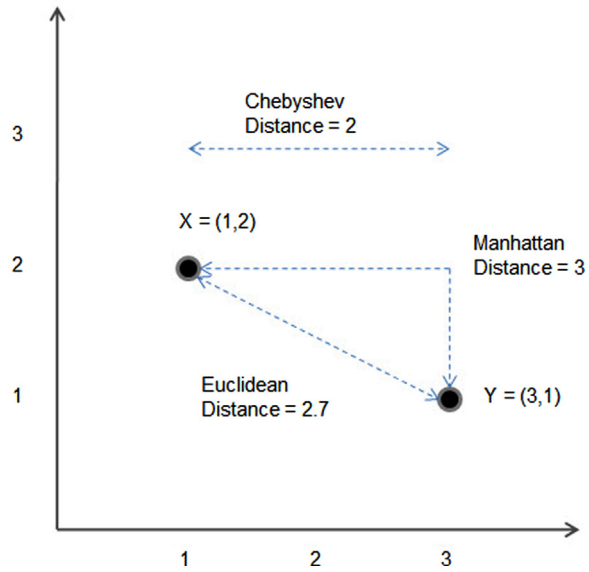


FIGURE 4.31
Distance measures.

by a cab that has to travel from X to Y in terms of number of city blocks is two blocks to the right and one block down). The Chebyshev distance is the maximum difference between all attributes in the dataset. In this example, the Chebyshev distance is the max of $[(3-1), (1-2)] = 2$. If Fig. 4.31 were a chess board, the Chebyshev distance would be the minimum number of moves required by the king to go from one position to another and the Manhattan distance is the minimum number of squares covered by the move of a rook from one position to another. All three aforementioned distance measures can be further generalized with one formula, the *Minkowski* distance measure. The distance between two points X $(x_1, x_2, \ldots, x_n)$ and Y $(y_1, y_2, \ldots, y_n)$ in n-dimensional space is given by Eq. (4.5):

$$d = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} \tag{4.5}$$

When $p = 1$, the distance measure is the Manhattan distance, when $p = 2$ the distance measure is the Euclidean distance, and when $p = \infty$ the distance measure is the Chebyshev distance. p is also called the *norm* and Eq. (4.5) is called the p-norm distance. The choice of distance measure depends on the data (Grabusts, 2011). The Euclidean measure is the most commonly used distance measure for numeric data. The Manhattan distance is used for binary attributes. For a new dataset with no prior knowledge, there is no

rule-of-thumb for an ideal distance measure. Euclidean distance would be a good start and the model can be tested with a selection of other distance measures and the corresponding performance.

Once the nearest k neighbors are determined, the process of determining the predicted target class is straightforward. The predicted target class is the majority class of the nearest k neighbors. Eq. (4.6) provides the prediction of the k-NN algorithm:

$$y' = \text{majority class}(y_1, y_2, y_3, ...y_k) \tag{4.6}$$

where y' is the predicted target class of the test data point and y_i is the class of i^{th} neighbor n_i.

Weights

The premise of the k-NN algorithm is that data points closer to each other are similar and, hence, have the same target class labels. When k is more than one, it can be argued that the closest neighbors should have more say in the outcome of the predicted target class than the farther neighbors (Hill & Lewicki, 2007). The farther away neighbors should have less influence in determining the final class outcome. This can be accomplished by assigned weights for all the neighbors, with the weights increasing as the neighbors get closer to the test data point. The weights are included in the final multi-voting step, where the predicted class is calculated. Weights (w_i) should satisfy two conditions: they should be proportional to the distance of the test data point from the neighbor and the sum of all weights should be equal to one. One of the calculations for weights shown in Eq. (4.7) follows an exponential decay based on distance:

$$w_i = \frac{e^{-d(x,n_i)}}{\sum_{i=1}^{k} e^{-d(x,n_i)}} \tag{4.7}$$

where w_i is the weight of i^{th} neighbor n_i, k the is total number of neighbors, and x is the test data point. The weight is used in predicting target class y':

$$y' = \text{majority class}(w_1 * y_1,\ w_2 * y_2,\ w_3 * y_3, \ldots,\ w_k * y_k) \tag{4.8}$$

where y_i is the class outcome of neighbor n_i.

The distance measure works well for numeric attributes. However, if the attribute is categorical, the distance between two points is either 0 or 1. If the attribute values are the same, the distance is 0 and if the attribute values are different, the distance is 1. For example, distance between (overcast, sunny) = 1 and distance between (sunny, sunny) = 0. If the attribute is ordinal with more than two values, then the ordinal values can be converted to an integer data type with values 0, 1, 2, …, $n - 1$ and the converted attribute can be treated as a numeric attribute for distance calculation. Obviously,

converting ordinal into numeric retains more information than using it as a categorical data type, where the distance value is either 0 or 1.

Correlation similarity

The correlation between two data points X and Y is the measure of the linear relationship between the attributes X and Y. Pearson correlation takes a value from -1 (perfect negative correlation) to $+1$ (perfect positive correlation) with the value of zero being no correlation between X and Y. Since correlation is a measure of *linear* relationship, a zero value does not mean there is no relationship. It just means that there is no linear relationship, but there may be a quadratic or any other higher degree relationship between the data points. Also, the correlation between one data point and another will now be explored. *This is quite different from correlation between variables*. Pearson correlation between two data points X and Y is given by:

$$\text{Correlation}(X, Y) = \frac{s_{xy}}{s_x \times s_y} \tag{4.9}$$

where s_{xy} is the covariance of X and Y, which is calculated as:

$$s_{x\,y} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})$$

and s_x and s_y are the standard deviation of X and Y, respectively. For example, the Pearson correlation of two data points X (1,2,3,4,5) and Y (10,15,35,40,55) is 0.98.

Simple matching coefficient

The simple matching coefficient is used when datasets have binary attributes. For example, let X be (1,1,0,0,1,1,0) and Y be (1,0,0,1,1,0,0). One can measure the similarity between these two data points based on the simultaneous occurrence of 0 or 1 with respect to total occurrences. The simple matching coefficient for X and Y can be calculated as:

$$\begin{aligned}\text{Simple matching coefficient (SMC)} &= \frac{\textit{Matching occurences}}{\textit{Total occurences}} \\ &= \frac{m_{00} + m_{11}}{m_{10} + m_{01} + m_{11} + m_{00}}\end{aligned} \tag{4.10}$$

In this example, m_{11} = occurrences where $(X = 1$ and $Y = 1) = 2$; m_{10} = occurrences where $(X = 1$ and $Y = 0) = 2$; m_{01} = occurrences where $(X = 0$ and $Y = 1) = 1$; m_{00} = occurrences where $(X = 0$ and $Y = 0) = 2$. Simple matching coefficient is, $(2 + 2)$ / $(2 + 2 + 1 + 2)$ = 4/7.

Jaccard similarity

If X and Y represent two text documents, each word will be an attribute in a dataset called a term document matrix or document vector. Each record in

the document dataset corresponds to a separate document or a text blob. This is explained in greater detail in Chapter 9, Text Mining. In this application, the number of attributes would be very large, often in the thousands. However, most of the attribute values will be zero. This means that two documents do not contain the same rare words. In this instance, what is interesting is that the comparison of the *occurrence* of the same word and non-occurrence of the same word does not convey any information and can be ignored. The Jaccard similarity measure is similar to the simple matching similarity but the nonoccurrence frequency is ignored from the calculation. For the same example X (1,1,0,0,1,1,0) and Y (1,0,0,1,1,0,0),

$$\begin{aligned}\text{Jaccard coefficient} &= \frac{\textit{Common occurences}}{\textit{Total occurences}} \\ &= \frac{m_{11}}{m_{10}+m_{01}+m_{11}} = \frac{2}{5}\end{aligned} \tag{4.11}$$

Cosine similarity

Continuing with the example of the document vectors, where attributes represent either the presence or absence of a word. It is possible to construct a more informational vector with the number of occurrences in the document, instead of just 1 and 0. Document datasets are usually long vectors with thousands of variables or attributes. For simplicity, consider the example of the vectors with X (1,2,0,0,3,4,0) and Y (5,0,0,6,7,0,0). The cosine similarity measure for two data points is given by:

$$\text{Cosine similarity}(|X,Y|) = \frac{x\cdot y}{||x||\ \ ||y||} \tag{4.12}$$

where $x\cdot y$ is the dot product of the x and y vectors with, for this example,

$$x\cdot y = \sum_{i=1}^{n} x_i y_i \text{ and } ||\text{x}|| = \sqrt{x\cdot x}$$

$$x\cdot y = \sqrt{1\times 5+2\times 0+0\times 0+0\times 6+3\times 7+4\times 0+0\times 0} = 5.1$$

$$||x|| = \sqrt{1\times 1+2\times 2+0\times 0+0\times 0+3\times 3+4\times 4+0\times 0} = 5.5$$

$$||y|| = \sqrt{5\times 5+0\times 0+0\times 0+6\times 6+7\times 7+0\times 0+0\times 0} = 10.5$$

$$\text{Cosine similarity}(|x\cdot y|) = \frac{x\cdot y}{||x||\ \ ||y||} = \frac{5.1}{5.5\times 10.5} = 0.08$$

The cosine similarity measure is one of the most used similarity measures, but the determination of the optimal measure comes down to the data structures. The choice of distance or similarity measure can also be parameterized, where multiple models are created with each different measure. The model with a distance measure that best fits the data with the smallest generalization error can be the appropriate proximity measure for the data.

4.3.2 How to Implement

Implementation of lazy learners is the most straightforward process amongst all the data science methods. Since the key functionality is referencing or looking up the training dataset, one could implement the entire algorithm in spreadsheet software like MS Excel, using lookup functions. Of course, if the complexity of the distance calculation or when the number of attributes rises, then one may need to rely on data science tools or programming languages. In RapidMiner, *k*-NN implementation is similar to other classification and regression process, with data preparation, modeling, and performance evaluation operators. The modeling step memorizes all the training records and accepts input in the form of real and nominal values. The output of this modeling step is just the dataset of all the training records.

Step 1: Data Preparation

The dataset used in this example is the standard Iris dataset with 150 examples and four numeric attributes. First, all attributes will need to be normalized using the *Normalize* operator, from the Data Transformation > Value Modification > Numerical Value Modification folder. The *Normalize* operator accepts numeric attributes and generates transformed numeric attributes. The user can specify one of four normalization methods in the parameter configurations: Z-transformation (most commonly used), range transformation, proportion transformation, and interquartile range. In this example, Z-transformation is used because all the attributes are being standardized.

The dataset is then split into two equal exclusive datasets using the *Split Data* operator. Split data (from Data Transformation > Filtering > Sampling) is used to partition the test and training datasets. The proportion of the partition and the sampling method can be specified in the parameter configuration of the split operator. For this example, the data is split equally between the training and test sets using shuffled sampling. One half of the dataset is used as training data for developing the *k*-NN model and the other half of the dataset is used to test the validity of the model.

Step 2: Modeling Operator and Parameters

The *k*-NN modeling operator is available in Modeling > Classification > Lazy Modeling. These parameters can be configured in the operator settings:

1. *k:* The value of *k* in *k*-NN can be configured. This defaults to one nearest neighbor. This example uses $k = 3$.

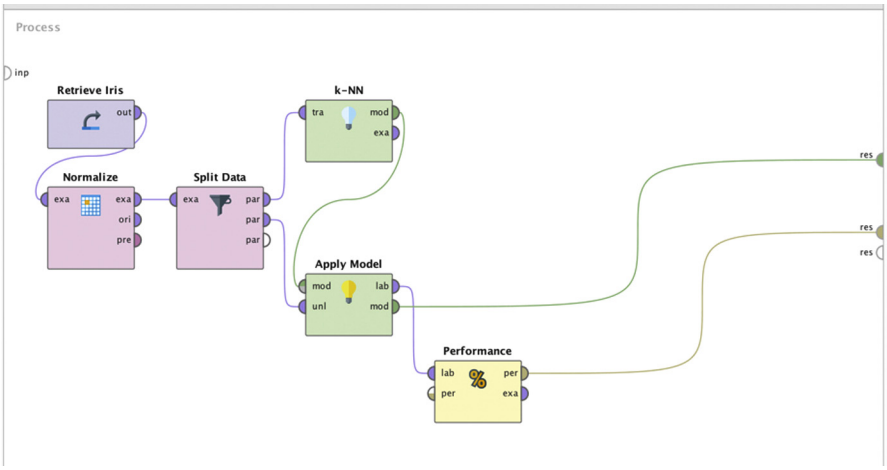


FIGURE 4.32
Data mining process for *k*-NN algorithm. *k*-NN, *k*-Nearest Neighbor.

2. *Weighted vote:* In the case of $k > 1$, this setting determines if the algorithm needs to take into consideration the distance value for the weights, while predicting the class value of the test record.
3. *Measure types:* There are more than two dozen distance measures available in RapidMiner. These measures are grouped in Measure Types. The selection of Measure Types drives the options for the next parameter (Measure).
4. *Measure:* This parameter selects the actual measure like Euclidean distance, Manhattan distance, and so on. The selection of the measure will put restrictions on the type of input the model receives. Depending on the weighting measure, the input data type choices will be limited and, hence, a data type conversion is required if the input data contains attributes that are not compatible with that measure.

Similar to other classification model implementations, the model will need to be applied to test the dataset, so the effectiveness of the model can be evaluated. Fig. 4.32 shows the RapidMiner process where the initial Iris dataset is split using a split operator. A random 75 of the initial 150 records are used to build the *k*-NN model and the rest of the data is the test dataset. The *Apply Model* operator accepts the test data and applies the *k*-NN model to predict the class type of the species. The *Performance* operator is then used to compare the predicted class with the labeled class for all of the test records.

Step 3: Execution and Interpretation

After the output from the *Performance* operator is connected to the result ports, as shown in Fig. 4.32, the model can be executed. The result output is observed as:

1. *k-NN model:* The model for *k*-NN is just the set of training records. Hence, no additional information is provided in this view, apart from the statistics of training records. Fig. 4.33 shows the output model.
2. *Performance vector:* The output of the *Performance* operator provides the confusion matrix with correct and incorrect predictions for all of the test dataset. The test set had 75 records. Fig. 4.34 shows the accurate prediction of 71 records (sum of diagonal cells in the matrix) and 4 incorrect predictions.
3. Labeled test dataset: The prediction can be examined at the record level.

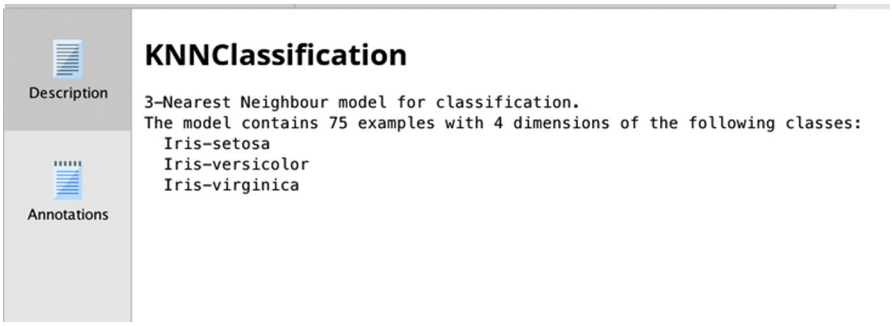


FIGURE 4.33

k-NN model output. *k*-NN, *k*-Nearest Neighbor.

PerformanceVector (Performance) | KNNClassification (k-NN)

Table View | Plot View

accuracy: 96.00%

	true Iris-setosa	true Iris-versicolor	true Iris-virginica	class precision
pred. Iris-setosa	20	0	0	100.00%
pred. Iris-versicolor	0	26	2	92.86%
pred. Iris-virginica	0	1	26	96.30%
class recall	100.00%	96.30%	92.86%	

FIGURE 4.34

Performance vector for *k*-NN model. *k*-NN, *k*-Nearest Neighbor.

4.3.3 Conclusion

The *k*-NN model requires normalization to avoid bias by any attribute that has large or small units in the scale. The model is quite robust when there are any missing attribute values in the test record. If a value in the test record is missing, the entire attribute is ignored in the model, and the model can still function with reasonable accuracy. In this implementation example, if the sepal length of a test record is not known, then the sepal length is ignored in the model. *k*-NN becomes a three-dimensional model instead of the original four dimensions.

As a lazy learner, the relationship between input and output cannot be explained, as the model is just a memorized set of all training records. There is no generalization or abstraction of the relationship. Eager learners are better at explaining the relationship and providing a description of the model.

Model building in *k*-NN is just memorizing and does not require much time. But, when a new unlabeled record is to be classified, the algorithm needs to find the distance between the unseen record and *all* the training records. This process can get expensive, depending on the size of training set and the number of attributes. A few sophisticated *k*-NN implementations index the records so that it is easy to search and calculate the distance. One can also convert the actual numbers to ranges so as to make it easy to index and compare it against the test record. However, *k*-NN is difficult to be used in time-sensitive applications like serving an online advertisement or real-time fraud detection.

k-NN models can handle categorical inputs but the distance measure will be either 1 or 0. Ordinal values can be converted to integers so that one can better leverage the distance function. Although the *k*-NN model is not good at generalizing the input-output relationship, it is still quite an effective model and leverages the existing relationships between attributes and class label in the training records. For good quality outcome, it requires a significant number of training records with the maximum possible permutations values in input attributes.

4.4 NAÏVE BAYESIAN

The data science algorithms used for classification tasks are quite diverse. The objective of all these algorithms is the same—prediction of a target variable. The method of prediction is drawn from a range of multidisciplinary techniques. The naïve Bayes algorithm finds its roots in statistics and probability theory. In general, classification techniques try to predict class labels by best approximating the relationship between the attributes and the class label.

Every day, we mentally estimate a myriad of outcomes based on past evidence. Consider the process of guessing commuting time to work. First, commute time depends heavily on when one is leaving home or work. If one is traveling during peak hours, the commute is going to be longer. Weather conditions like rain, snow, or dense fog will slow down the commute. If the day is a school holiday, like summer break, then the commute will be lighter than on school days. If there is any road work, the commute usually takes longer. When more than one adverse factor is at play, then the commute will be even longer than if it is just one isolated factor is in the play. This *if-then* knowledge is based on previous experience of commuting when one or more factors come into play. Our experience creates a model in our brain and we mentally run the model before pulling out of the driveway!

Take the case of defaults in home mortgages and assume the average overall default rate is 2%. The likelihood of an average person defaulting on their mortgage loan is 2%. However, if a given individual's credit history is above average (or excellent), then the likelihood of their default would be less than average. Furthermore, if one knows that the person's annual income is above average with respect to loan value, then the likelihood of default falls further. As more evidence is obtained on the factors that impact the outcome, improved guesses can be made about the outcome using probability theory. The naïve Bayesian algorithm leverages the probabilistic relationship between the attributes and the class label. The algorithm makes a strong and sometimes *naïve* assumption of independence between the attributes, thus, its name. The independence assumption between attributes may not always hold true. It can be assumed that annual income and home value are independent of each other. However, homeowners with high income tend to buy more expensive houses. Although the independence assumption doesn't always hold true, the simplicity and robustness of the algorithm offsets the limitation introduced by the assumption.

PREDICTING AND FILTERING SPAM EMAIL

Spam is unsolicited bulk email sent to a wide number of email users. At best it is an annoyance to recipients but many of the spam emails hide a malicious intent by hosting false advertisements or redirecting clicks to phishing sites. Filtering spam email is one of the essential features provided by email service providers and administrators. The key challenge is balance between incorrectly flagging a legitimate email as spam (false positive) versus not catching all the spam messages. There is no perfect spam filtering solution and spam detection is a catch-up game. The spammers always try to deceive and outsmart the spam filters and email administrators fortify the filters for various new spam scenarios. Automated spam filtering based on algorithms provides a promising solution in containing spam and in learning frameworks to update the changing filtering solutions (Process Software, 2013).

Some words occur in spam emails more often than in legitimate email messages. For example, the probability of the occurrence for words like free, mortgage, credit, sale, Viagra, etc., is higher in spam mails than in normal emails. The exact probabilities can be calculated if one

(*Continued*)

(Continued)

has a sample of previously known spam emails and regular emails. Based on the known word probabilities, the overall probability of an email being spam can be computed based on all the words in the email and the probability of each word being in spam versus regular emails. This is the foundation of Bayesian spam filtering systems (Zdziarski, 2005). Any misclassified spam messages that are subsequently reclassified by the user (by marking it as spam) are an opportunity to refine the model, making spam filtering adaptive to new spam techniques. Though current spam reduction methods use a combination of different algorithms, Bayesian-based spam filtering remains one of the foundational elements of spam prediction systems (Sahami, Dumais, Heckerman, & Horvitz, 1998).

4.4.1 How It Works

The naïve Bayesian algorithm is built on the Bayes' theorem, named after Reverend Thomas Bayes. Bayes' work is described in "Essay Towards Solving a Problem in the Doctrine of Chances" (1763), published posthumously in the *Philosophical Transactions of the Royal Society of London* by Richard Price. The Bayes' theorem is one of the most influential and important concepts in statistics and probability theory. It provides a mathematical expression for how a degree of subjective belief changes to account for new evidence. First, the terminology used in Bayes' theorem need to be discussed.

Assume $\mathbf{X}$ is the evidence (attribute set) and Y is the outcome (class label). Here $\mathbf{X}$ is a set, not individual attributes, hence, $\mathbf{X} = \{X_1, X_2, X_3, \ldots, X_n\}$, where X_i is an individual attribute, such as credit rating. The probability of outcome $P(Y)$ is called *prior probability*, which can be calculated from the training dataset. Prior probability shows the likelihood of an outcome in a given dataset. For example, in the mortgage case, $P(Y)$ is the default rate on a home mortgage, which is 2%. $P(Y|\mathbf{X})$ is called the *conditional probability*, which provides the probability of an outcome given the evidence, that is, when the value of $\mathbf{X}$ is known. Again, using the mortgage example, $P(Y|\mathbf{X})$ is the average rate of default given that an individual's credit history is known. If the credit history is excellent, then the probability of default is likely to be less than 2%. $P(Y|\mathbf{X})$ is also called *posterior probability*. Calculating posterior probability is the objective of data science using Bayes' theorem. This is the likelihood of an outcome as the conditions are learnt.

Bayes' theorem states that:

$$P(Y|\mathbf{X}) = \frac{P(Y) \times P(\mathbf{X}|Y)}{P(\mathbf{X})} \tag{4.13}$$

$P(\mathbf{X}|Y)$ is another conditional probability, called the *class conditional probability*. $P(\mathbf{X}|Y)$ is the probability of the existence of conditions given an outcome. Like $P(Y)$, $P(\mathbf{X}|Y)$ can be calculated from the training dataset as well. If the training set of loan defaults is known, the probability of an "excellent" credit rating can be calculated given that the default is a "yes." As indicated in the Bayes'

theorem, class conditional probability is crucial to calculating posterior probability. $P(\mathbf{X})$ is basically the probability of the evidence. In the mortgage example, this is simply the proportion of individuals with a given credit rating. To classify a new record, one can compute $P(Y|\mathbf{X})$ for *each class of* Y and see which probability "wins." Class label Y with the highest value of $P(Y|\mathbf{X})$ wins for given condition $\mathbf{X}$. Since $P(\mathbf{X})$ is the same for every class value of the outcome, one does not have to calculate this and it can be assumed as a constant. More generally, for an example set with n attributes $\mathbf{X} = \{X_1, X_2, X_3 \ldots X_n\}$,

$$P(Y|\mathbf{X}) = \frac{P(Y) \times \prod_{i=1}^{n} P(\mathbf{X}_i|Y)}{P(\mathbf{X})} \tag{4.14}$$

If one knows how to calculate class conditional probability $P(\mathbf{X}|Y)$ or $\prod_{i=1}^{n} P(\mathbf{X}_i|Y)$, then it is easy to calculate posterior probability $P(Y|\mathbf{X})$. Since $P(X)$ is constant for every value of Y, it is enough to calculate the numerator of the equation $P(Y) \times \prod_{i=1}^{n} P(\mathbf{X}_i|Y)$ for every class value.

To further explain how the naïve Bayesian algorithm works, the modified Golf dataset shown in Table 4.4 will be used. The Golf table is an artificial dataset with four attributes and one class label. Note that the categorical data type is being used for ease of explanation (temperature and humidity have been converted from the numeric type). In Bayesian terms, weather condition is the *evidence* and decision to play or not play is the *belief*. Altogether there are 14 examples with 5 examples of Play = no and nine examples of Play = yes. The objective is to predict if the player will Play (yes or no), given the weather condition, based on learning from the dataset in Table 4.4. Here is the step-by-step explanation of how the Bayesian model works.

Table 4.4 Golf Dataset With Modified Temperature and Humidity Attributes

No.	Temperature X_1	Humidity X_2	Outlook X_3	Wind X_4	Play (Class Label) Y
1	High	Med	Sunny	false	no
2	High	High	Sunny	true	no
3	Low	Low	Rain	true	no
4	Med	High	Sunny	false	no
5	Low	Med	Rain	true	no
6	High	Med	Overcast	false	yes
7	Low	High	Rain	false	yes
8	Low	Med	Rain	false	yes
9	Low	Low	Overcast	true	yes
10	Low	Low	Sunny	false	yes
11	Med	Med	Rain	false	yes
12	Med	Low	Sunny	true	yes
13	Med	High	Overcast	true	yes
14	High	Low	Overcast	false	yes

Step 1: Calculating Prior Probability P(Y)

Prior probability $P(Y)$ is the probability of an outcome. In this example set there are two possible outcomes: Play = yes and Play = no. From Table 4.4, 5 out of 14 records with the "no" class and 9 records with the "Yes" class. The probability of outcome is

$$P(Y = \text{no}) = 5/14$$
$$P(Y = \text{yes}) = 9/14$$

Since the probability of an outcome is calculated from the dataset, it is important that the dataset used for data science is *representative* of the population, if sampling is used. The class-stratified sampling of data from the population will be ideal for naïve Bayesian modeling. The class-stratified sampling ensures the class distribution in the sample is the same as the population.

Step 2: Calculating Class Conditional Probability $P(X_i|Y)$

Class conditional probability is the probability of *each* attribute value for an attribute, for each outcome value. This calculation is repeated for all the attributes: Temperature (X_1), Humidity (X_2), Outlook (X_3), and Wind (X_4), and for every distinct outcome value. Here is a calculation of the class conditional probability of Temperature (X_1). For each value of the Temperature attribute, $P(X_1|Y = \text{no})$ and $P(X_1|Y = \text{yes})$ can be calculated by constructing a class conditional probability table as shown in Table 4.5. From the dataset there are five $Y = \text{no}$ records and nine $Y = \text{yes}$ records. Out of the five $Y = \text{no}$ records, the probability of occurrence can be calculated for when the temperature is high, medium, and low. The values will be 2/5, 1/5, and 2/5, respectively. The same process can be repeated when the outcome is $Y = \text{yes}$.

Similarly, the calculation can be repeated to find the class conditional probability for the other three attributes: Humidity (X_2), Outlook (X_3), and Wind (X_4). This class conditional probability table is shown in Table 4.6.

Step 3: Predicting the Outcome Using Bayes' Theorem

With the class conditional probability tables all prepared they can now be used in the prediction task. If a new, unlabeled test record (Table 4.7) has the conditions Temperature = high, Humidity = low, Outlook = sunny, and

Table 4.5 Class Conditional Probability of Temperature

Temperature (X_1)	$P(X_1\|Y = \text{no})$	$P(X_1\|Y = \text{yes})$
High	2/5	2/9
Med	1/5	3/9
Low	2/5	4/9

Table 4.6 Conditional Probability of Humidity, Outlook, and Wind

Humidity (X_2)	**$P(X_2\|Y = \text{no})$**	**$P(X_2\|Y = \text{yes})$**
High	2/5	2/9
Low	1/5	4/9
Med	2/5	3/9
Outlook (X_3)	**$P(X_3\|Y = \text{no})$**	**$P(X_3\|Y = \text{yes})$**
Overcast	0/5	4/9
Rain	2/5	3/9
Sunny	3/5	2/9
Wind (X_4)	**$P(X_4\|Y = \text{no})$**	**$P(X_4\|Y = \text{yes})$**
False	2/5	6/9
True	3/5	3/9

Table 4.7 Test Record

No.	**Temperature X_1**	**Humidity X_2**	**Outlook X_3**	**Wind X_4**	**Play (Class Label) Y**
Unlabeled test	high	Low	Sunny	False	?

Wind = false, what would the class label prediction be? Play = yes or no? The outcome class can be predicted based on Bayes' theorem by calculating the posterior probability $P(Y|\mathbf{X})$ for both values of Y. Once $P(Y = \text{yes}|\mathbf{X})$ and $P(Y = \text{no}|\mathbf{X})$ are calculated, one can determine which outcome has higher probability and the predicted outcome is the one that has the highest probability. While calculating both conditional probabilities using Eq. (4.14), it is sufficient to just calculate the numerator portion, as $P(\mathbf{X})$ is going to be the same for both the outcome classes (Table 4.7).

$$P(Y = \text{yes}|\text{X}) = \frac{P(Y) * \prod_{i=1}^{n} p(X_i|Y)}{P(X)}$$

$$= P(Y = \text{yes}) * \{P(\text{Temp} = \text{high}|Y = \text{yes}) * P(\text{Humidity} = \text{low}|Y = \text{yes}) * P(\text{Outlook} = \text{sunny}|Y = \text{yes}) * P(\text{Wind} = \text{false}|Y = \text{yes})\} / P(X)$$

$$= \frac{9/14 * \{2/9 * 4/9 * 2/9 * 6/9\}}{P(X)}$$

$$= \frac{0.0094}{P(X)}$$

$$P(Y = \text{no}|\text{X}) = 5/14 * \{2/5 * 4/5 * 3/5 * 2/5\}$$

$$= \frac{0.0274}{P(X)}$$

Both the estimates can be normalized by dividing both conditional probability by (0.0094 + 0.027) to get:

$$\text{Likelihood of (Play} = \text{yes)} = \frac{0.0094}{0.0274 + 0.0094} = 26\%$$

$$\text{Likelihood of (Play} = \text{no)} = \frac{0.0094}{0.0274 + 0.0094} = 74\%$$

In this case $P(Y = \text{yes}|X) < P(Y = \text{no}|X)$, hence, the prediction for the unlabeled test record will be Play = no.

The Bayesian modeling is relatively simple to understand, once one gets past the probabilistic concepts, and is easy to implement in practically any programing language. The computation for model building is quite simple and involves the creation of a lookup table of probabilities. Bayesian modeling is quite robust in handling missing values. If the test example set does not contain a value, suppose temperature is not available, the Bayesian model simply omits the corresponding class conditional probability for all the outcomes. Having missing values in the test set would be difficult to handle in decision trees and regression algorithms, particularly when the missing attribute is used higher up in the node of the decision tree or has more weight in regression. Even though the naïve Bayes algorithm is quite robust to missing attributes, it does have a few limitations. Here are couple of the most significant limitations and methods of mitigation.

Issue 1: Incomplete Training Set

Problems arise when an attribute value in the testing record has no example in the training record. In the Golf dataset (Table 4.4), if an unseen test example consists of the attribute value Outlook = overcast, the probability of $P(\text{Outlook} = \text{overcast}|Y = \text{no})$ is zero. Even if one of the attribute's class conditional probabilities is zero, by nature of the Bayesian equation, the entire posterior probability will be zero.

$$P(Y = \text{no}|\text{X}) = P(Y = \text{No}) * \{P(\text{Team} = \text{high}|Y = \text{no}) * P(\text{Humidity} = \text{low}|$$
$$Y = \text{no}) * P(\text{Outlook} = \text{overcast}|Y = \text{no}) * P(\text{Wind} = \text{false}|Y = \text{no})\}$$
$$\frac{}{P(X)}$$
$$= \frac{5/14 * \{2/5 * 1/5 * 0 * 2/5\}}{P(X)}$$
$$= 0$$

In this case $P(Y = \text{yes}|\mathbf{X}) > P(Y = \text{no}|\mathbf{X})$, and the test example will be classified as Play = yes. If there are no training records for any other attribute value, like Temperature = low for outcome *yes*, then probability of both outcomes, $P(Y = \text{no}|\mathbf{X})$ and $P(Y = \text{yes}|\mathbf{X})$, will also be zero and an arbitrary prediction shall be made because of the dilemma.

To mitigate this problem, one can assign small default probabilities for the missing records instead of zero. With this approach, the absence of an attribute value doesn't wipe out the value of $P(\mathbf{X}|Y)$, albeit it will reduce the probability to a small number. This technique is called *Laplace correction*. Laplace correction adds a controlled error in all class conditional probabilities. If the training set contains Outlook = overcast, then $P(X|Y = \text{no}) = 0$. The class conditional probability for all the three values for Outlook is 0/5, 2/5, and 3/5, Y = no. Controlled error can be added by adding 1 to all numerators and 3 for all denominators, so the class conditional probabilities are 1/8, 3/8, and 4/8. The sum of all the class conditional probabilities is still 1. Generically, the Laplace correction is given by corrected probability:

$$P(X_i|Y) = \frac{0 + \mu p_3}{5 + \mu}, \frac{2 + \mu p_2}{5 + \mu}, \frac{3 + \mu p_2}{5 + \mu} \tag{4.15}$$

where $p_1 + p_2 + p_3 = 1$ and μ is the correction.

Issue 2: Continuous Attributes

If an attribute has continuous numeric values instead of nominal values, this solution will not work. The continuous values can always be converted to nominal values by discretization and the same approach as discussed can be used. But discretization requires exercising subjective judgment on the bucketing range, leading to loss of information. Instead, the continuous values can be preserved as such and the probability density function can used. One assumes the probability distribution for a numerical attribute follows a normal or Gaussian distribution. If the attribute value is known to follow some other distribution, such as Poisson, the equivalent probability density function can be used. The probability density function for a normal distribution is given by:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.16}$$

where μ is the mean and σ is the standard deviation of the sample.

In the Golf dataset shown in Table 4.8, temperature and humidity are continuous attributes. In such a situation, the mean and standard deviation can be computed for both class labels (Play = yes and Play = no) for temperature and humidity (Table 4.9).

If an unlabeled test record has a Humidity value of 78, the probability density can be computed using Eq. (4.16), for both outcomes. For outcome Play = yes, if the values $x = 78$, $\mu = 73$, and $\sigma = 6.16$ are plugged in to the probability density function, the equation renders the value 0.04. Similarly,

Table 4.8 Golf Dataset with Continuous Attributes

No.	Outlook X_1	Humidity X_2	Temperature X_3	Wind X_4	Play Y
1	Sunny	85	85	false	no
2	Sunny	80	90	true	no
6	Rain	65	70	true	no
8	Sunny	72	95	false	no
14	Rain	71	80	true	no
3	Overcast	83	78	false	yes
4	Rain	70	96	false	yes
5	Rain	68	80	false	yes
7	Overcast	64	65	true	yes
9	Sunny	69	70	false	yes
10	Rain	75	80	false	yes
11	Sunny	75	70	true	yes
12	Overcast	72	90	true	yes
13	Overcast	81	75	false	yes

Table 4.9 Mean and Deviation for Continuous Attributes

Play Value		Humidity X_2	Temperature X_3
Y = no	Mean	74.60	84.00
	Deviation	7.89	9.62
Y = yes	Mean	73.00	78.22
	Deviation	6.16	9.88

for outcome Play = no, $x = 78$, $\mu = 74.6$, $\sigma = 7.89$ can be plugged in and the probability density is computed to obtain 0.05:

$$P(\text{temperature} = 78|Y = \text{yes}) = 0.04$$
$$P(\text{temperature} = 78|Y = \text{no}) = 0.05$$

These values are probability densities and *not* probabilities. In a continuous scale, the probability of temperature being exactly at a particular value is zero. Instead, the probability is computed for a range, such as temperatures from 77.5 to 78.5 units. Since the same range is used for computing the probability density for both the outcomes, Play = yes and Play = no, it is not necessary to compute the actual probability. Hence, these temperature values can be substituted in the Bayesian Eq. 4.14 for calculating class conditional probability $P(\mathbf{X}|Y)$.

Issue 3: Attribute Independence

One of the fundamental assumptions in the naïve Bayesian model is *attribute independence*. Bayes' theorem is guaranteed only for independent attributes. In many real-life cases, this is quite a stringent condition to deal with. This is why the technique is called "naïve" Bayesian, because it assumes an attribute's independence. In practice the naïve Bayesian model works fine with slightly correlated features (Rish, 2001). This problem can be handled by pre-processing the data. Before applying the naïve Bayesian algorithm, it makes sense to remove strongly correlated attributes. In the case of all numeric attributes, this can be achieved by computing a weighted correlation matrix. An advanced application of Bayes' theorem, called a Bayesian belief network, is designed to handle datasets with attribute dependencies.

The independence of categorical attributes can be tested by the *chi-square* (χ^2) test for independence. The chi-square test is calculated by creating a contingency table of observed frequency like the one shown in Table 4.10A. A contingency table is a simple cross tab of two attributes under consideration.

A contingency table of expected frequency (Table 4.10B) is created based on the equation:

$$E_{r,c} = \frac{(\text{row total} \times \text{column total})}{(\text{table total})} \tag{4.17}$$

The chi-square statistic (χ^2) calculates the sum of the difference between these two tables. χ^2 is calculated by Eq. (4.18). In this equation, O is observed frequency and E is expected frequency:

$$x^2 = \sum \frac{(O-E)^2}{E} \tag{4.18}$$

If the chi-square statistic (χ^2) is less than the critical value calculated from the chi-square distribution for a given confidence level, then one can assume the two variables under consideration are independent, for practical purposes.

Table 4.10 Contingency Tables with *observed frequency* (A) and *expected frequency* (B)

	(A) Wind—Observed Frequency				(B) Wind—Expected Frequency		
Outlook	**False**	**True**	**Total**	**Outlook**	**False**	**True**	**Total**
Overcast	2	2	**4**	Overcast	2.29	1.71	**4**
Rain	3	2	**5**	Rain	2.86	2.14	**5**
Sunny	3	2	**5**	Sunny	2.86	2.14	**5**
Total	**8**	**6**	**14**	**Total**	**8**	**6**	**14**

4.4.2 How to Implement

The naïve Bayesian model is one of the few data science techniques that can be easily implemented in any programing language. Since the conditional probability tables can be prepared in the model building phase, the execution of the model in runtime is quick. Data science tools have dedicated naïve Bayes classifier functions. In RapidMiner, the *Naïve Bayes* operator is available under Modeling > Classification. The process of building a model and applying it to new data is similar to with decision trees and other classifiers. The *naïve Bayesian* models can accept both numeric and nominal attributes.

Step 1: Data Preparation

The Golf dataset shown in Table 4.8 is available in RapidMiner under Sample > Data in the repository section. The Golf dataset can just be dragged and dropped in the process area to source all 14 records of the dataset. Within the same repository folder, there is also a Golf-Test dataset with a set of 14 records used for testing. Both datasets need to be added in the main process area. Since the Bayes operator accepts numeric and nominal data types, no other data transformation process is necessary. Sampling is a common method to extract the training dataset from a large dataset. It is especially important for naïve Bayesian modeling for the training dataset to be representative and proportional to the underlying dataset.

Step 2: Modeling Operator and Parameters

The *Naïve Bayes* operator can now be connected to the Golf training dataset. The *Naïve Bayesian* operator has only one parameter option to set: whether or not to include Laplace correction. For smaller datasets, Laplace correction is strongly encouraged, as a dataset may not have all combinations of attribute values for every class value. In fact, by default, Laplace correction is checked. Outputs of the *Naïve Bayes* operator are the model and original training dataset. The model output should be connected to *Apply Model* (Model Application folder) to execute the model on the test dataset. The output of the *Apply Model* operator is the labeled test dataset and the model.

Step 3: Evaluation

The labeled test dataset that one gets after using the *Apply Model* operator is then connected to the *Performance—Classification* operator to evaluate the performance of the classification model. The *Performance—Classification* operator can be found under Evaluation>Performance Measurement> Performance. Fig. 4.35 shows the complete naïve Bayesian predictive classification process. The output ports should be connected to the result ports and the process can be saved and executed.

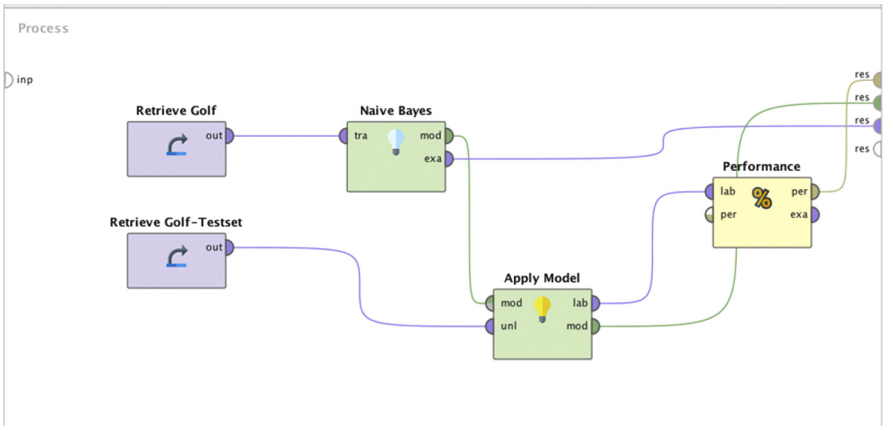


FIGURE 4.35
Data mining process for naïve Bayes algorithm.

Step 4: Execution and Interpretation

The process shown in Fig. 4.35 has three result outputs: a model description, performance vector, and labeled dataset. The labeled dataset contains the test dataset with the predicted class as an added column. The labeled dataset also contains the confidence for each label class, which indicates the prediction strength.

The model description result contains more information on class conditional probabilities of all the input attributes, derived from the training dataset. The Charts tab in model description contains probability density functions for the attributes, as shown in Fig. 4.36. In the case of continuous attributes, the decision boundaries can be discerned across the different class labels for the Humidity attribute. It has been observed that when Humidity exceeds 82, the likelihood of Play = no increases. The Distribution Table shown in Fig. 4.37 contains the familiar class conditional probability table similar to Tables 4.5 and 4.6.

The performance vector output is similar to previously discussed classification algorithms. The performance vector provides the confusion matrix describing accuracy, precision, and recall metrics for the predicted test dataset.

4.4.3 Conclusion

The Bayesian algorithm provides a probabilistic framework for a classification problem. It has a simple and sound foundation for modeling data and

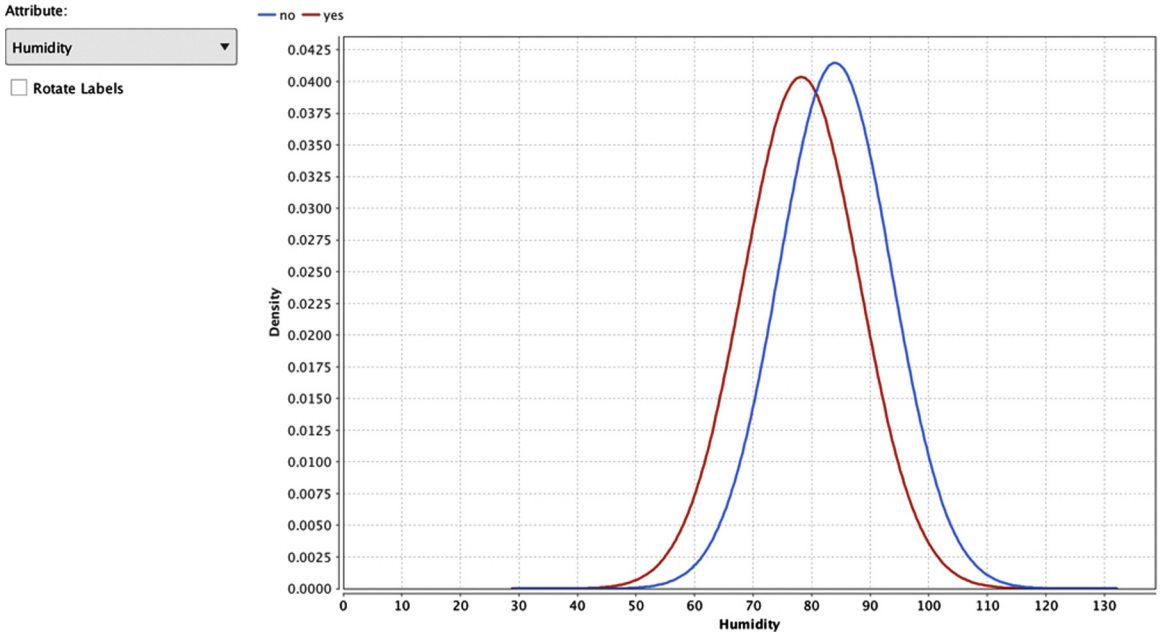
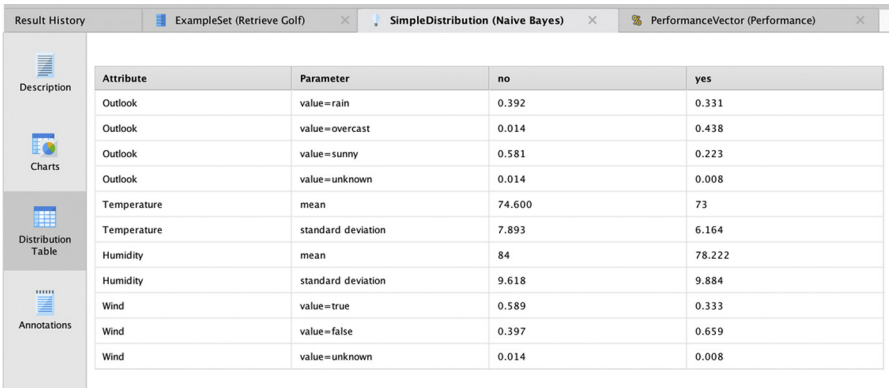


FIGURE 4.36
Naïve Bayes model output: probability density function for humidity attribute.



Attribute	Parameter	no	yes
Outlook	value=rain	0.392	0.331
Outlook	value=overcast	0.014	0.438
Outlook	value=sunny	0.581	0.223
Outlook	value=unknown	0.014	0.008
Temperature	mean	74.600	73
Temperature	standard deviation	7.893	6.164
Humidity	mean	84	78.222
Humidity	standard deviation	9.618	9.884
Wind	value=true	0.589	0.333
Wind	value=false	0.397	0.659
Wind	value=unknown	0.014	0.008

FIGURE 4.37
Naïve Bayes distribution table output.

is quite robust to outliers and missing values. This algorithm is deployed widely in text mining and document classification where the application has a large set of attributes and attributes values to compute. The naïve Bayesian classifier is often a great place to start for a data science project as it serves as a good benchmark for comparison to other models. Implementation of the Bayesian model in production systems is quite straightforward and the use of data science tools is optional. One major limitation of the model is the assumption of independent attributes, which can be mitigated by advanced modeling or decreasing the dependence across the attributes through preprocessing. The uniqueness of the technique is that it leverages new information as it arrives and tries to make a best prediction considering new evidence. In this way, it is quite similar to how our minds work. Talking about the mind, the next algorithm mimics the biological process of human neurons!

4.5 ARTIFICIAL NEURAL NETWORKS

The objective of a supervised learner is to model the relationship between input and output variables. The neural network technique approaches this problem by developing a functional relationship between input and output variables by mimicking the architecture of the biological process of a *neuron*. Although the developers of this technique have used many biological terms to explain the inner workings of neural network modeling process, it has a simple mathematical foundation. Consider the linear model:

$$Y = 1 + 2X_1 + 3X_2 + 4X_3$$

where Y is the calculated output and X_1, X_2, and X_3 are input attributes. 1 is the intercept and 2, 3, and 4 are the scaling factors or coefficients for the input attributes X_1, X_2, and X_3, respectively. This simple linear model can be represented in a topological form as shown in Fig. 4.38.

In this topology, X_1 is the input value and passes through a node, denoted by a circle. Then the value of X_1 is multiplied by its weight, which is 2, as noted in the connector. Similarly, all other attributes (X_2 and X_3) go through a node and scaling transformation. The last node is a special case with no input variable; it just has the intercept. Finally, the values from all the connectors are summarized in an output node that yields the predicted output Y. The topology shown in Fig. 4.38 represents the simple linear model $Y = 1 + 2X_1 + 3X_2 + 4X_3$. The topology also represents a simple *artificial neural network* (ANN). The neural networks model more complex nonlinear

CHAPTER 5

Regression Methods

In this chapter, one of the most commonly used data science techniques—fitting data with functions or *function fitting* will be explored. The basic idea behind function fitting is to predict the value (or class) of a dependent attribute y, by combining the predictor attributes **X** into a function, $y = f(\mathbf{X})$. Function fitting involves many different techniques and the most common ones are *linear regression* for numeric prediction and *logistic regression* for classification. These two, form the majority of the material in this chapter. Regression models continue to be one of the most common analytics tools used by practitioners today.[1]

Regression is a relatively old technique dating back to the Victorian era (1830s to the early 1900s). Much of the pioneering work was done by Sir Francis Galton, a distant relative of Charles Darwin, who came up with the concept of *regressing toward the mean* while systematically comparing children's heights against their parents' heights. He observed there was a strong tendency for tall parents to have children slightly shorter than themselves, and for short parents to have children slightly taller than themselves. Even if the parents' heights were at the tail ends of a bell curve or normal distribution, their children's heights tended toward the mean of the distribution. Thus, in the end, all the samples regressed toward a population mean. Therefore, this trend was called *regression* by Galton (Galton, 1888) and, thus, the foundations for linear regression were laid.

In the first section of this chapter, the theoretical framework for the simplest of function-fitting methods: the *linear regression model*, will be provided. The main focus will be on a case study that demonstrates how to build regression models. Due to the nature of the function-fitting approach, one limitation

[1] Rexer Analytics survey available from http://www.rexeranalytics.com.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00005-8

that modelers have to deal with is what is called the *curse of dimensionality*. As the number of predictors **X**, increases, not only will our ability to obtain a good model reduce, it also adds computational and interpretational complexity. *Feature selection* methods will be introduced that can reduce the number of predictors or factors required to a minimum and still obtain a good model. The mechanics of implementation, will be explored, to do the data preparation, model building, and validation. Finally, in closing some checkpoints to ensure that linear regression is used correctly will be described.

In the second section of this chapter *logistic regression* will be discussed. Strictly speaking, it is a classification technique, closer in its application to decision trees or Bayesian methods. But it shares an important characteristic with linear regression in its function-fitting methodology and, thus, merits inclusion in this chapter, rather than the previous one on classification.

5.1 LINEAR REGRESSION

Linear regression is not only one of the oldest data science methodologies, but it also the most easily explained method for demonstrating function

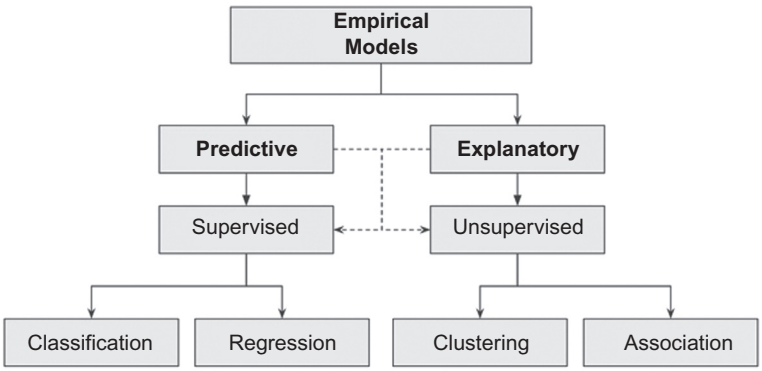
PREDICTING HOME PRICES

What features would play a role in deciding the value of a home? For example, locality, the number of rooms, its age, the quality of schools in the vicinity, its location with respect to major sources of employment, and its accessibility to major highways, are some of the important considerations most potential home buyers would like to factor in. But which of these are the most significant influencers of the price? Is there a way to determine these? Once these factors are known, can they be incorporated into a model that can be used for predictions? The case study that will be discussed later in this chapter addresses this problem, using multiple linear regressions to predict the median home prices in an urban region given the characteristics of a home.

A common goal that all businesses have to address in order to be successful is growth, in revenues and profits. Customers are what will enable this to happen. Understanding and increasing the likelihood that someone will buy again from the company is, therefore, critical. Another question that would help strategically, for example in customer segmentation, is being able to predict how much money a customer is likely to spend, based on data about their previous purchase habits. Two very important distinctions need to be made here: understanding why someone purchased from the company will fall into the realm of *explanatory modeling*, whereas, predicting how much someone is likely to spend will fall into the realm of predictive modeling. Both these types of models fall under a broader category of *surrogate* or empirical models which relies on historical data to develop rules of behavior as opposed to *system* models which use fundamental principles (such as laws of physics or chemistry) to develop rules. See Fig. 1.2 for a taxonomy of data science. In this chapter, the predictive capability of models will be focused on as opposed to the explanatory capabilities. Historically much of applied linear regression in statistics has been used for explanatory needs. Later on in this chapter with the case of logistic regression, it will be demonstrated, how *both* needs can be met with good analytical interpretation of models.

(*Continued*)

(Continued)



fitting. The basic idea is to come up with a function that explains and predicts the value of the target variable when given the values of the predictor variables.

5.1.1 How it Works

A simple example is shown in Fig. 5.1: if one would like to know the effect of the number of rooms in a house (predictor) on its median sale price (target). Each data point on the chart corresponds to a house (Harrison, 1978). It is evident that on average, increasing the number of rooms tends to also increase median price. This general statement can be captured by drawing a straight line through the data. The problem in linear regression is, therefore, finding a line (or a curve) that best explains this tendency. If there are two predictors, then the problem is to find a surface (in a three-dimensional space). With more than two predictors, visualization becomes difficult and one has to revert to a general statement where the dependent variables are expressed as a linear combination of independent variables:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \tag{5.1}$$

Consider the problem with one predictor. Clearly, one can fit an infinite number of straight lines through a given set of points such as the ones shown in Fig. 5.1. How does one know which one is the best? A metric is

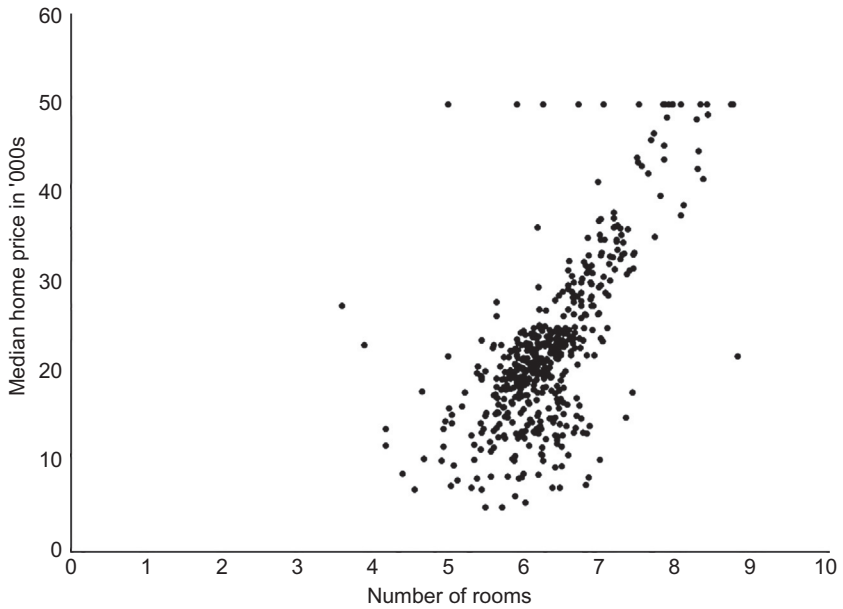


FIGURE 5.1
A simple regression model.

needed, one that helps quantify the different straight line fits through the data. Once this metric is found, then selecting the best line becomes a matter of finding the optimum value for this quantity.

A commonly used metric is the concept of an error function. Suppose one fits a straight line through the data. In a single predictor case, the *predicted* value, $\hat{y}$, for a value of x that exists in the dataset is then given by:

$$\hat{y} = b_0 + b_1 x \tag{5.2}$$

Then, error is simply the difference between the actual target value and predicted target value:

$$e = y - \hat{y} = y - (b_0 + b_1 x) \tag{5.3}$$

This equation defines the error at a single location (x, y) in the dataset. One could easily compute the error for all existing points to come up with an aggregate error. Some errors will be positive, and others will be negative. The difference can be squared to eliminate the sign bias and an average error for a given fit can be calculated as:

$$\frac{J}{n} = \frac{\sum e^2}{n} = \frac{\sum (y_i - \dot{y}_i)^2}{n} = \frac{\sum (y_i - b_0 - b_i x_i)^2}{n} \tag{5.4}$$

where n represents the number of points in the dataset. J is the total squared error. For a given dataset, the best combination of (b_0, b_1) can then be found,

which minimizes the total error, e. This is a classical minimization problem, which is handled with methods of calculus. Stigler provides some interesting historical details on the origins of the method of least squares, as it is known (Stigler, 1999). Using the methods of calculus, the values of b can be found, which minimize the total error J. Specifically, one can take partial derivatives of J with respect to b_1 and b_0 and set them equal to zero. Chain rule of differential calculus gives us:

$$\begin{aligned}
&\partial J/\partial b_1 = \partial J/\partial \hat{y} \quad \partial \hat{y}/\partial b_1 \\
&\Rightarrow \partial J/\partial b_1 = 2(\Sigma(y_i - b_0 - b_1 x_i))\partial \hat{y}/\partial b_1 = 0 \\
&\Rightarrow \Sigma(y_i - b_0 - b_1 x_i)(-x_i) = 0 \\
&\Rightarrow -\Sigma(y_i x_i) + \Sigma(b_0 x_i) + \Sigma(b_1 x^2{}_i) = 0 \\
&\Rightarrow \Sigma(y_i x_i) = b_0 \Sigma(x_i) + b_1 \Sigma(x^2{}_i)
\end{aligned} \tag{5.5}$$

Similarly, one can use:

$$\begin{aligned}
&\partial J/\partial b_0 = 2(\Sigma(y_i - b_0 - b_1 x))\partial \hat{y}/\partial b_0 = 0 \\
&\Rightarrow \Sigma(y_i - b_0 - b_1 x_i)(-1) = 0 \\
&\Rightarrow -\Sigma(y_i) + \Sigma(b_0.1) + \Sigma(b_1 x_i)1 = 0 \\
&\Rightarrow -\Sigma(y_i) + b_0 \Sigma(1) + b_1 \Sigma(x_i) = 0 \\
&\Rightarrow \Sigma(y_i) = b_0 N + b_1 \Sigma(x_i)
\end{aligned} \tag{5.6}$$

Eqs. (5.5) and (5.6) are two equations in two unknowns, b_0 and b_1, which can be further simplified and solved to yield the expressions:

$$b_1 = (\Sigma x_i y_i - \bar{y}\Sigma x_i)/(\Sigma x^2{}_i - \bar{x} - \Sigma x_i) \tag{5.7}$$

$$b_0 = (\bar{y}\Sigma x^2{}_i - \bar{x}\Sigma x_i y_i)/(\Sigma x^2{}_i - \bar{x}\Sigma x_i) \tag{5.8}$$

b_1 can also be written as (5.9a):

$$b_1 = \text{Correlation } (y, x) \times \frac{s_y}{s_x} \tag{5.9a}$$

$$b_0 = y_{\text{mean}} - b_1 \times x_{\text{mean}} \tag{5.9b}$$

where Correlation(x, y) is the correlation between x and y and s_y, s_x are the standard deviations of y and x. Finally, x_{mean} and y_{mean} are the respective mean values.

Practical linear regression algorithms use an optimization technique known as *gradient descent* (Fletcher, 1963; Marquardt, 1963) to identify the combination of b_0 and b_1 which will minimize the error function given in Eq. (5.4). The advantage of using such methods is that even with several predictors, the optimization works fairly robustly. When such a process is applied to the simple example shown, one gets an equation of the form:

$$\text{Median price} = 9.1 \times (\textit{number of rooms}) - 34.7 \tag{5.10}$$

where b_1 is 9.1 and b_0 is -34.7. From this equation, it can be calculated that for a house with six rooms, the value of the median price is about 20 (the prices are expressed in thousands of US dollars, c. 1970). In Fig. 5.1 it's

evident that for a house with six rooms, the actual price can range between 10.5 and 25. An infinite number of lines could have been fit in this band, which would have all predicted a median price within this range—but the algorithm chooses the line that minimizes the average error over the full range of the independent variable and is, therefore, the best fit for the given dataset.

For some of the points (houses) shown in Fig. 5.1 (at the top of the chart, where median price = 50) the median price appears to be independent of the number of rooms. This could be because there may be other factors that also influence the price. Thus, more than one predictor will need to be modeled and *multiple linear regression (MLR)*, which is an extension of simple linear regression, will need to be used. The algorithm to find the coefficients of the regression Eq. (5.1) can be easily extended to more than one dimension.

The single variable expression for error function in (5.4) can be generalized to multiple variables quite easily, $\hat{y}_i = b_0 + b_1x_1 + ... + b_Dx_D$. If we let $x = [x_0,x_1,...,x_D]$ and recognize that the intercept term can be written as b_0x_0 where $x_0 = 1$, then we can write (5.4) as $E = \sum_{i=1}^{N}(y_i - B^Tx_i)^2$ where B is a vector of weights $[b_0, b_1,...,b_D]$ for a dataset with D columns or features and N samples. Similar to how we computed (5.7) and (5.8), we can take a derivative of E with respect to each weight B and will end up with D equations to be solved for D weights (one corresponding to each feature). The partial derivative for each weight is

$$\partial E/\partial b_j = \partial E/\partial \hat{y} * \partial \hat{y}_i/\partial b_j$$
$$\Rightarrow \partial E/\partial b_j = 2\Sigma\left(y_i - B^Tx_i\right)\partial \hat{y}_i/\partial b_j$$
$$\Rightarrow \partial E/\partial b_j = 2\Sigma\left(y_i - B^Tx_i\right)(-x_i)$$
$$\Rightarrow \Sigma y_i(-x_i) - B^T\Sigma(x_i)(-x_i)$$

When we consider all D weights at the same time, this can be very simply written in matrix form as follows:

$$\partial E/\partial B = -\left(Y^TX\right) + B\left(X^TX\right) \tag{5.11}$$

Here B is a $1 \times D$ matrix or vector. As in the case of simple linear regression, we can set this derivative to zero, to solve for the weights, B and get the following expression: $-(Y^TX) + B(X^TX) = 0$. In this case, solving for B now becomes a matrix inversion problem and results in $B = (X^TX)^{-1}Y^TX$. The reader can verify as an exercise that this matrix is dimensionally consistent (Hint: Use the fact that the matrix shape of X is $N \times D$, Y is $N \times 1$ and B is $1 \times D$). MLR can be applied in any situation where a numeric prediction, for example "how much will something sell for," is required. This is in contrast to making categorical predictions such as "will someone buy/not buy" or "will/will not fail," where classification tools such as decision trees or logistic regression models are used. In order to ensure regression models are not arbitrarily deployed, several checks must be performed on the model to ensure that the regression is accurate which will be the focus of a later section in this chapter.

Table 5.1 Sample View of the Classic Boston Housing Dataset

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Target = MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1

Table 5.2 Attributes of Boston Housing Dataset

1. CRIM per capita crime rate by town
2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS proportion of non-retail business acres per town
4. CHAS Charles River dummy variable (=1 if tract bounds river; 0 otherwise)
5. NOX nitric oxide concentrations (parts per 10 million)
6. RM average number of rooms per dwelling
7. AGE proportion of owner-occupied units built prior to 1940
8. DIS weighted distances to five Boston employment centers
9. RAD index of accessibility to radial highways
10. TAX full-value property-tax rate per $10,000
11. PTRATIO pupil-teacher ratio by town
12. B 1000(Bk − 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT percent lower status of the population
14. MEDV Median value of owner-occupied homes in $1000's

The housing example can be extended in order to include additional variables. This comes from a study of urban environments conducted in the late 1970s[2] (Harrison, 1978). The objectives of this are:

1. Identify which of the several attributes are required to accurately predict the median price of a house.
2. Build a multiple linear regression model to predict the median price using the most important attributes.

The original data consist of thirteen predictors and one response variable, which is the variable that needs to be predicted. The predictors include physical characteristics of the house (such as number of rooms, age, tax, and location) and neighborhood features (schools, industries, zoning) among others. The response variable is of course the median value (MEDV) of the house in thousands of dollars. Table 5.1 shows a snapshot of the dataset, which has altogether 506 examples. Table 5.2 describes the features or attributes of the dataset.

[2] http://archive.ics.uci.edu/ml/datasets/Housing.

5.1.2 How to Implement

In this section, how to set up a RapidMiner process to build a multiple linear regression model for the Boston Housing dataset will be demonstrated. The following will be described:

1. Building a linear regression model
2. Measuring the performance of the model
3. Understanding the commonly used options for the *Linear Regression* operator
4. Applying the model to predict MEDV prices for unseen data

Step 1: Data Preparation

As a first step, the data is separated into a training set and an unseen test set. The idea is to build the model with the training data and test its performance on the unseen data. With the help of the *Retrieve* operator, import the raw data (available in the companion website www.IntroDataScience.com) into the RapidMiner process. Apply the *Shuffle* operator to randomize the order of the data so that when the two partitions are seperated, they are statistically similar. Next, using the *Filter Examples Range* operator, divide the data into two sets as shown in Fig. 5.2. The raw data has 506 examples, which will be linearly split into a training set (from row 1 to 450) and a test set (row 451−506) using the two operators.

Insert the *Set Role* operator, change the role of MEDV to label and connect the output to a *Split Validation* operator's input training port as shown in Fig. 5.3. The training data is now going to be further split into a training set and a validation set (keep the default Split Validation options as

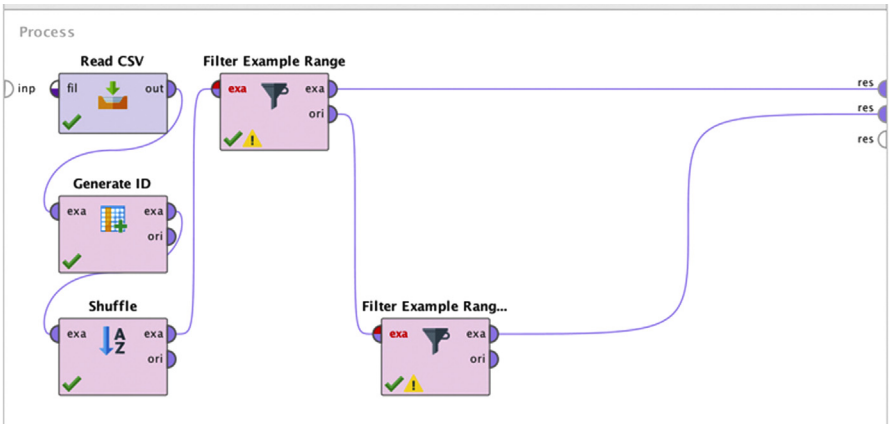


FIGURE 5.2
Separating the data into training and testing samples.

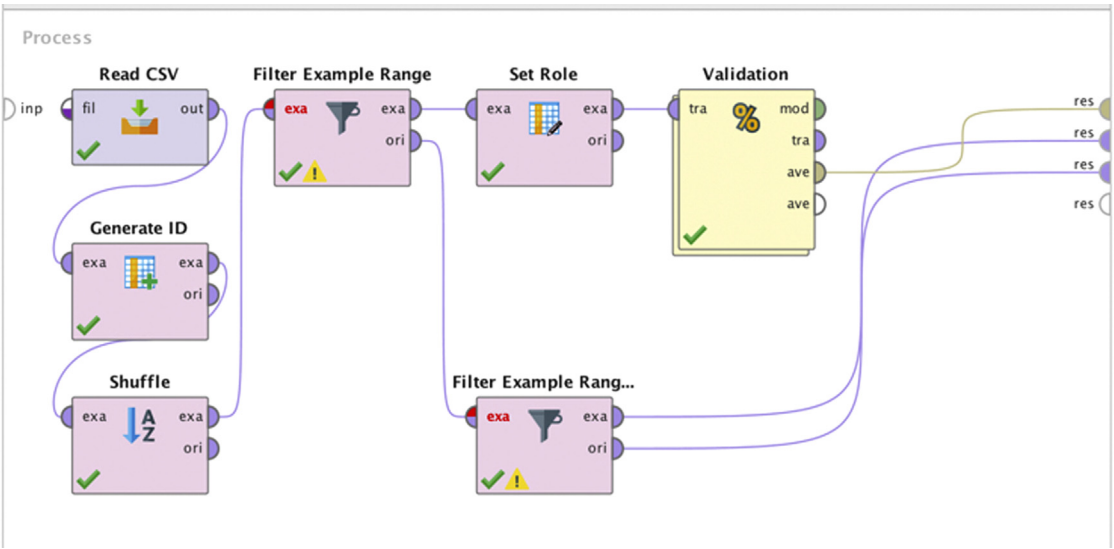


FIGURE 5.3
Using the split validation operator.

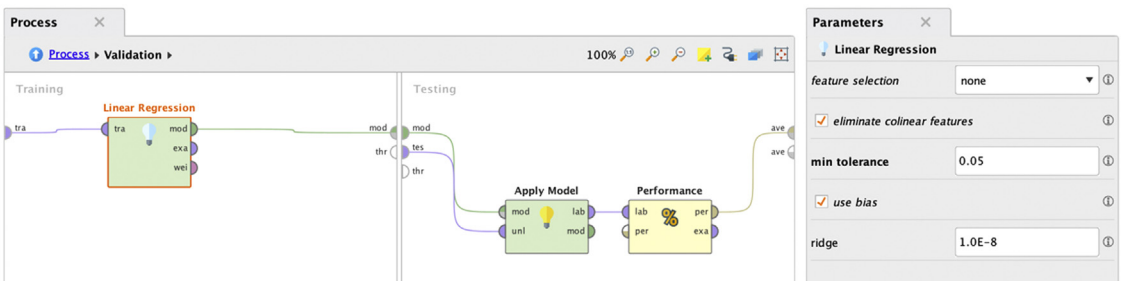


FIGURE 5.4
Applying the linear regression operator and measuring performance.

is, i.e., *relative*, *0.7*, and *shuffled*). This will be needed in order to measure the performance of the linear regression model. It is also a good idea to set the local random seed (to default the value of 1992), which ensures that RapidMiner selects the same samples if this process is run at a later time.

After this step, double-click the *Validation* operator to enter the nested process. Inside this process insert the *Linear Regression* operator on the left window and *Apply Model* and *Performance (Regression)* in the right window as shown in Fig. 5.4. Click on the *Performance* operator and check squared error, correlation, and squared correlation inside the Parameters options selector on the right (Fig. 5.5).

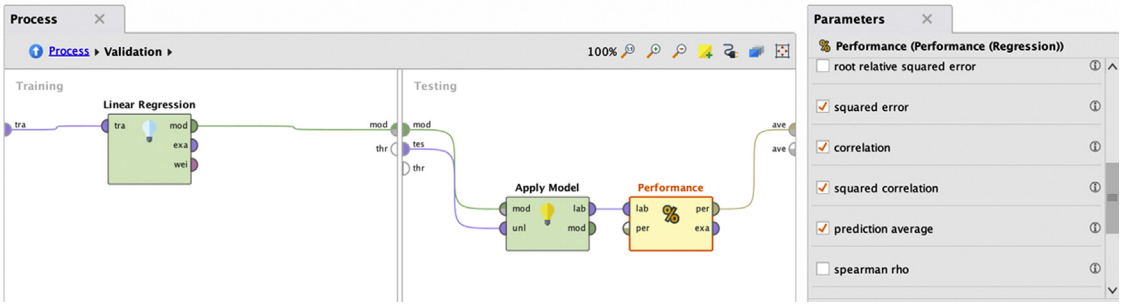


FIGURE 5.5
Selecting performance criteria for the MLR.

Step 2: Model Building

Select the *Linear Regression* operator and change the *feature selection* option to none. Keep the default eliminate collinear features checked, which will remove factors that are linearly correlated from the modeling process. When two or more attributes are correlated to one another, the resulting model will tend to have coefficients that cannot be intuitively interpreted and, furthermore, the statistical significance of the coefficients also tends to be quite low. Also keep the use bias checked to build a model with an intercept [the b_0 in Eq. (5.2)]. Keep the other default options intact (Fig. 5.4).

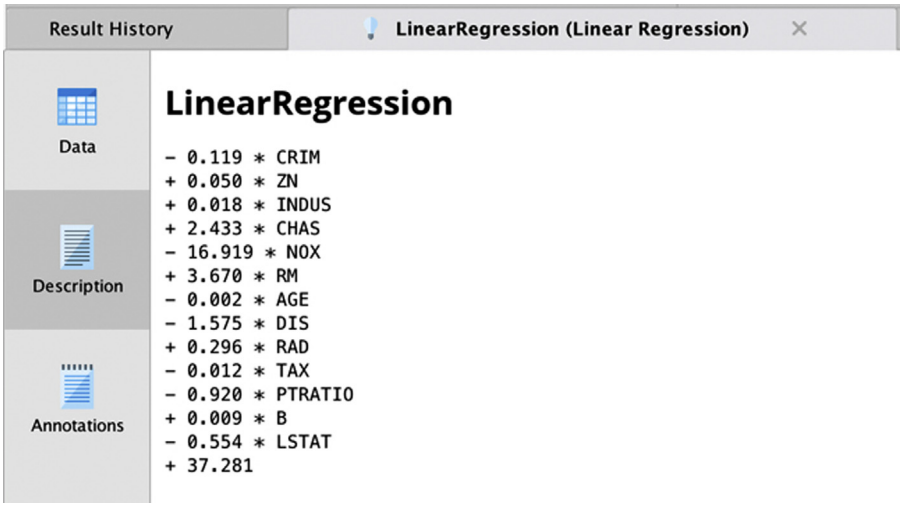
When this process is run the results shown in Fig. 5.6 will be generated.

Step 3: Execution and Interpretation

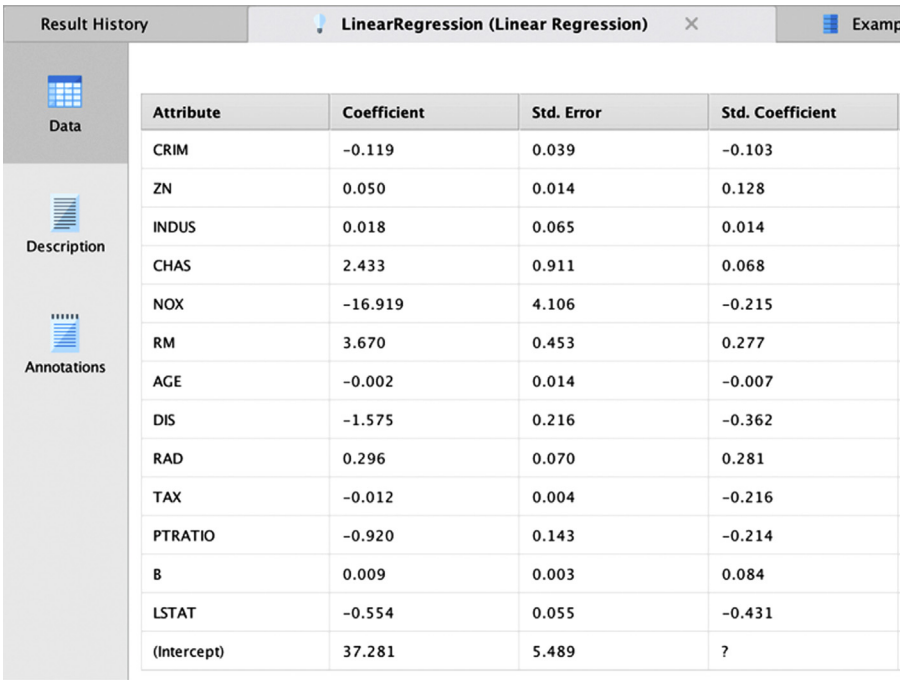
There are two views that one can examine in the *Linear Regression* output tab: the Description view, which actually shows the function that is fitted (Fig. 5.6A) and the more useful Data view, which not only shows the coefficients of the linear regression function, but also gives information about the significance of these coefficients (Fig. 5.6B). The best way to read this table is to sort it by double-clicking on the column named Code, which will sort the different factors according to their decreasing level of significance. RapidMiner assigns four stars (****) to any factor that is highly significant.

In this model, no *feature selection* method was used and as a result all 13 factors are in the model, including AGE and INDUS, which have very low significance. However, if the same model were to be run by selecting any of the options that are available in the drop-down menu of the *feature selection* parameter, RapidMiner would have removed the least significant factors from the model. In the next iteration, the *greedy* feature selection is used, and this will have removed the least significant factors, INDUS and AGE, from the function (Fig. 5.7A & B).

(A)



(B)



Attribute	Coefficient	Std. Error	Std. Coefficient
CRIM	-0.119	0.039	-0.103
ZN	0.050	0.014	0.128
INDUS	0.018	0.065	0.014
CHAS	2.433	0.911	0.068
NOX	-16.919	4.106	-0.215
RM	3.670	0.453	0.277
AGE	-0.002	0.014	-0.007
DIS	-1.575	0.216	-0.362
RAD	0.296	0.070	0.281
TAX	-0.012	0.004	-0.216
PTRATIO	-0.920	0.143	-0.214
B	0.009	0.003	0.084
LSTAT	-0.554	0.055	-0.431
(Intercept)	37.281	5.489	?

FIGURE 5.6

(A) Description of the linear regression model. (B) Tabular view of the model. Sort the table according to significance by double-clicking on the Code column.

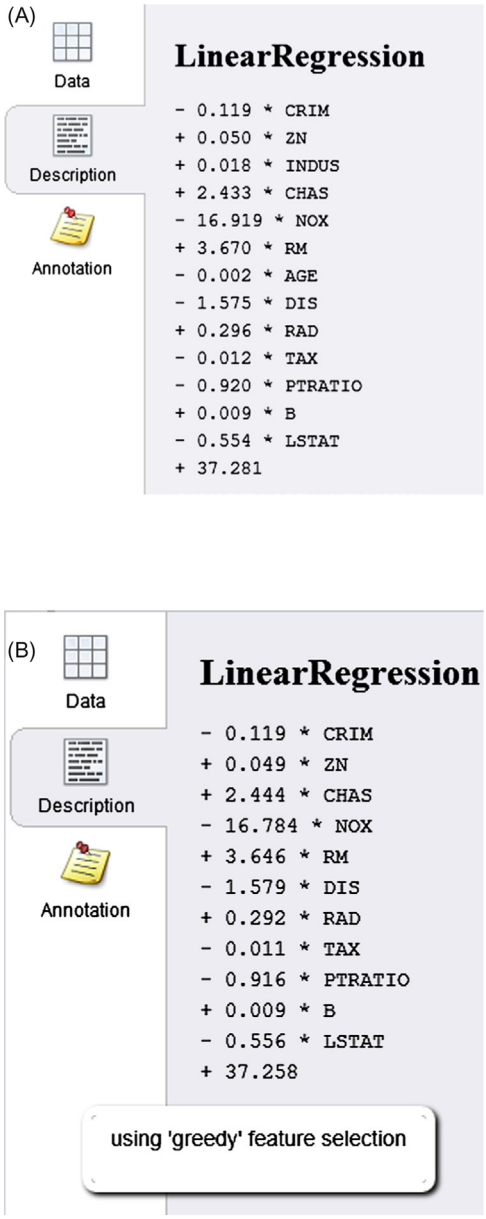


FIGURE 5.7
(A) Model without any feature selection. (B) Model with greedy feature selection.

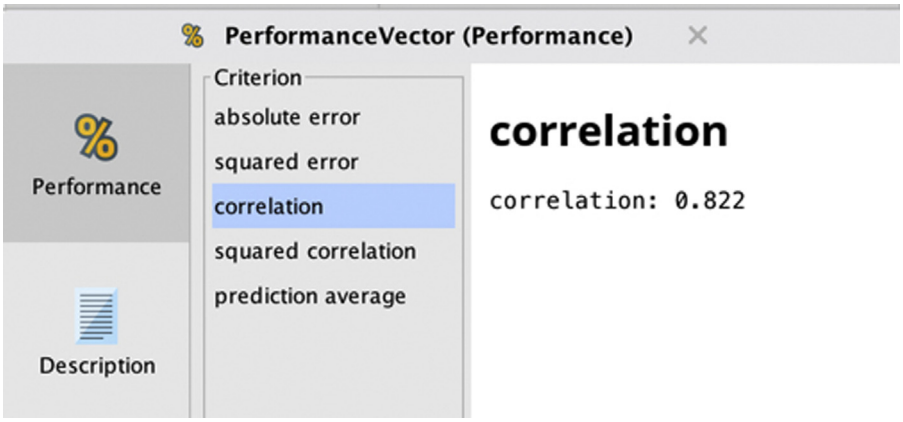


FIGURE 5.8
Generating the r^2 for the model.

Feature selection in RapidMiner can be done automatically within the *Linear Regression* operator as described or by using external wrapper functions such as forward selection and backward elimination. These will be discussed separately in Chapter 14, Feature Selection.

The second output to pay attention to is the Performance: a handy check to test the goodness of fit in a regression model is the *squared correlation*. Conventionally this is the same as the adjusted r^2 for a model, which can take values between 0.0 and 1.0, with values closer to 1 indicating a better model. For either of the models shown above, a value around 0.822 can be obtained (Fig. 5.8). The squared error output was also requested: the raw value in itself may not reveal much, but this is useful in comparing two different models. In this case it was around 25.

One additional insight that can be extracted from the modeling process is ranking of the factors. The easiest way to check this is to rank by *p*-value. As seen in Fig. 5.9, RM, LSTAT, and DIS seem to be the most significant factors. This is also reflected in their absolute *t*-stat values. The *t*-stat and *P*-values are the result of the hypothesis tests conducted on the regression coefficients. For the purposes of predictive analysis, the key takeaway is that a higher *t*-stat signals that the null hypothesis—which assumes that the coefficient is zero—can be safely rejected. The corresponding *p*-value indicates the probability of wrongly rejecting the null hypothesis. Its already been noted how the number of rooms (RM) was a good predictor of the home prices, but it was unable to explain all of the variations in median price. The r^2 and squared

Attribute	Coefficient	Std. Error	Std. Coeff...	Tolerance	t-Stat	p-Value ↑	Code
LSTAT	-0.556	0.052	-0.432	0.490	-10.661	0	****
RM	3.646	0.443	0.275	0.581	8.236	0.000	****
DIS	-1.579	0.199	-0.363	0.823	-7.928	0.000	****
(Intercept)	37.258	5.440	?	?	6.849	0.000	****
PTRATIO	-0.916	0.140	-0.213	0.793	-6.547	0.000	****
NOX	-16.784	3.794	-0.213	0.812	-4.424	0.000	****
RAD	0.292	0.068	0.276	0.769	4.318	0.000	****
ZN	0.049	0.014	0.128	0.877	3.465	0.001	****
TAX	-0.011	0.004	-0.208	0.749	-3.219	0.001	***
CRIM	-0.119	0.039	-0.103	0.843	-3.088	0.002	***
B	0.009	0.003	0.083	0.905	2.953	0.003	***
CHAS	2.444	0.905	0.068	0.991	2.701	0.007	***

FIGURE 5.9
Ranking variables by their *P*-values.

error for that one-variable model were 0.405 and 45, respectively. This can be verified by rerunning the model built so far using only one independent variable, the number of rooms, RM. This is done by using the *Select Attributes* operator, which has to be inserted in the process before the *Set Role* operator. When this model is run, the equation shown earlier, Eq. (5.10), will be obtained in the model *Description*. By comparing the corresponding values from the MLR model (0.676 and 25) to the simple linear regression model, it's evident that both of these quantities have improved, thus, affirming the decision to use multiple factors.

One now has a more comprehensive model that can account for much of the variability in the response variable, MEDV. Finally, a word about the sign of the coefficients: LSTAT refers to the percentage of low-income households in the neighborhood. A lower LSTAT is correlated with higher median home price, and this is the reason for the negative coefficient on LSTAT.

Step 4: Application to Unseen Test Data

This model is now ready to be deployed against the unseen data that was created at the beginning of this section using the second *Filter Examples* operator (Fig. 5.2). A new *Set Role* operator will need to be added, select MEDV under parameters and set it to target role prediction from the pull-down menu. Add another *Apply Model* operator and connect the output of *Set Role* to its unlabeled port; additionally, connect the output model from the Validation

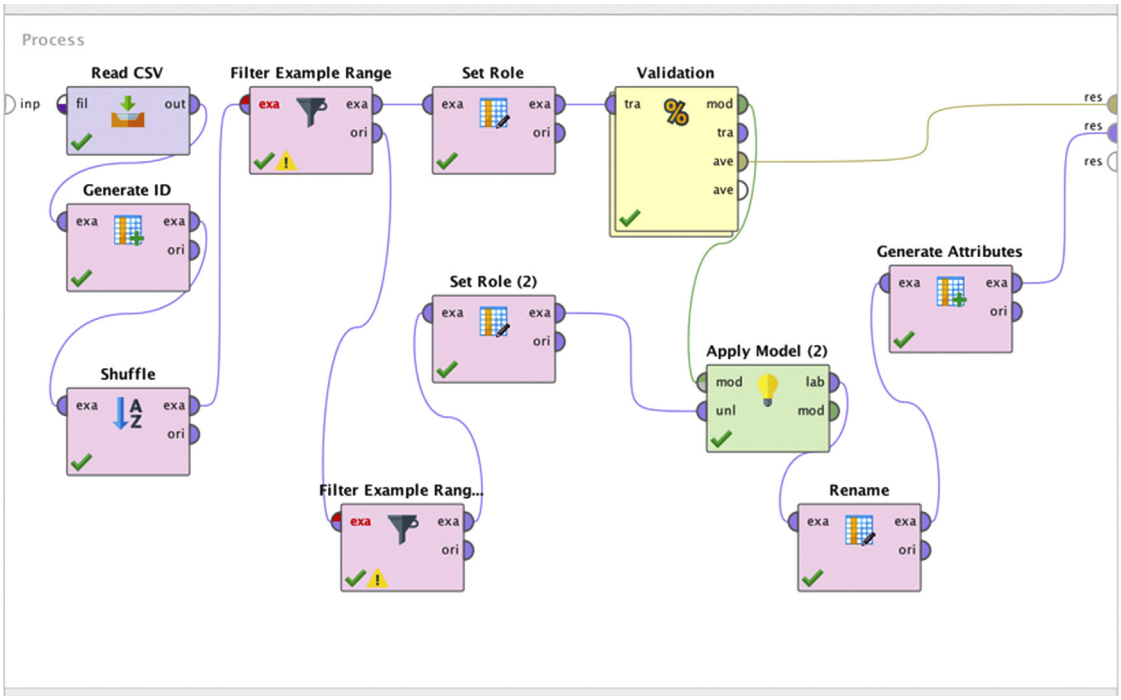


FIGURE 5.10
Setting up a process to do the comparison between the unseen data and the model predicted values.

process to the input model port of the new *Apply Model*. What has been done is, the attribute MEDV has been changed from the unseen set of 56 examples to a prediction. When the model is applied to this example set, one will be able to compare the prediction (MEDV) values to the original MEDV values (which exist in the set) to test how well the model would behave on new data. The difference between prediction (MEDV) and MEDV is termed residual. Fig. 5.10 shows one way to quickly check the residuals for the models application. A *Rename* operator would be needed to change the name of "prediction (MEDV)" to predictedMEDV to avoid confusing RapidMiner when the next operator, *Generate Attributes*, is used to calculate residuals (try without using the *Rename* operator to understand this issue as it can pop up in other instances where *Generate Attributes* is used). Fig. 5.11 shows the statistics for the new attribute residuals which indicate that the mean is close to 0 (−0.27) but the standard deviation (and hence, variance) at 4.350 is not quite small. The histogram also seems to indicate that the residuals are not quite normally distributed, which would be another motivation to continue to improve the model.

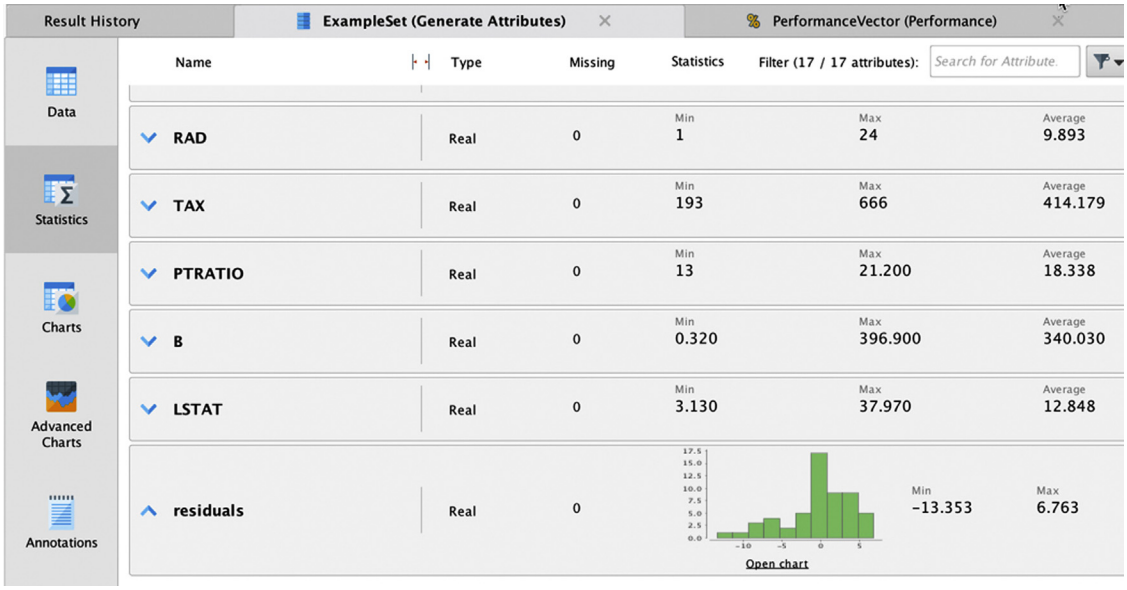


FIGURE 5.11
Statistics of the residuals for the unseen data show that some model optimization may be necessary.

5.1.3 Checkpoints

This section on linear regression can be wrapped up with a brief discussion of several checkpoints to ensure that any models are valid. This is a critical step in the analytics process because all modeling follows the garbage in, garbage out (GIGO) dictum. It is incumbent upon the data scientist to ensure these checks are completed.

Checkpoint 1: One of the first checkpoints to consider before accepting any regression model is to quantify the r^2, which is also known as the coefficient of determination which effectively explains how much variability in the dependent variable is explained by the independent variables (Black, 2008). In most cases of linear regression, the r^2 value lies between 0 and 1. The ideal range for r^2 varies across applications; for example, in social and behavioral science models typically low values are acceptable. Generally, extremely low values ($\sim$ <0.2) indicate that the variables in the model do not explain the outcome satisfactorily. A word of caution about overemphasizing the value of r^2: when the intercept is set to zero (in RapidMiner, when one unchecks *use bias*, Fig. 5.5), r^2 values tend to be inflated because of the manner in which they are calculated. In such situations where a zero intercept are required, it makes sense to use other checks such as the mean and variance of the residuals.

The most common metric used for measuring how well a regression model fits the data is by using the coefficient of determination, r^2. This is defined as:

$$r^2 = 1 - \text{SSE}/\text{SSYY} \tag{5.12}$$

SSE is simply the sum of squared errors which is given by $\sum e^2$ in Eq. (5.4). SSYY is the aggregate mean deviation defined by:

$$\text{SSYY} = \sum (y - \bar{y})^2 \tag{5.13}$$

Intuitively, it is easy to see that if SSE is close to zero, then r^2 is close to 1—a perfect fit. If SSE is close to SSYY then r^2 is close to zero—the model is simply predicting the average value of y for all values of x. The nice thing about r^2 is that because it only depends on y and not the weights or independent variables, it can be used for any form of regression model: simple or multiple.

Checkpoint 2: This leads to the next check, which is to ensure that all error terms in the model are normally distributed. To do this check in RapidMiner, a new attribute called error can be generated, which is the difference between the predicted MEDV and the actual MEDV in the test dataset. This can be done using the *Generate Attributes* operator. This is what was done in step 5 in the last section. Passing checks 1 and 2 will ensure that the independent and dependent variables are related. However, this does not imply that the independent variable is the cause and the dependent is the effect. Remember that *correlation is not causation*!

Checkpoint 3: Highly nonlinear relationships, result in simple regression models failing these checks. However, this does not mean that the two variables are not related. In such cases it may become necessary to resort to somewhat more advanced analytical methods to test the relationship. This is best described and motivated by Anscombe's quartet, presented in Chapter 3, Data Exploration.

Checkpoint 4: In addition to testing the goodness of a model fit using r^2, it is also important to ensure that there is no *overfitting* of data. Overfitting refers to the process of developing a model that is so well tuned to represent the training data that its least squares error is as low as possible, but this error becomes high when the model is used on unseen or new data. That is, the model fails to generalize. The following example illustrates this and also develops intuition for avoiding such behavior by using what is called Regularization.

Consider some sample data which represent an underlying simple function $y = 3x + 1$. If the data were to be plotted it would look like Fig. 5.12.

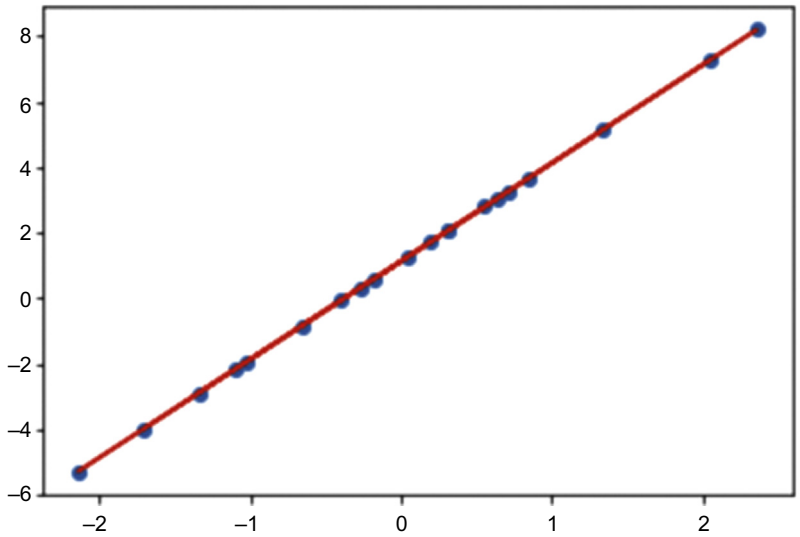


FIGURE 5.12
Linear regression line.

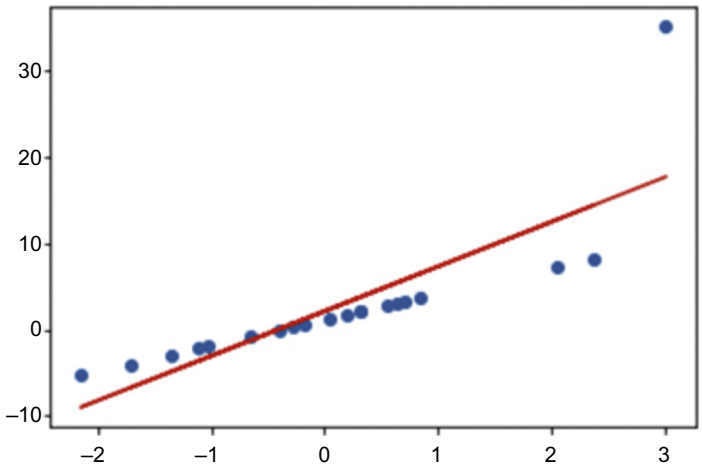


FIGURE 5.13
Linear regression line with an outlier.

A regression model could be fit to this data and a nice linear fit obtained, as shown by the line, as well as obtaining the following coefficients: $b_0 = 1.13$ and $b_1 = 3.01$, which is close to the underlying function. Now suppose that there is a new data point [3, 30] which is somewhat of an outlier and now the model has to be refit, the results would be similar to those shown in Fig. 5.13.

Compared to the previous fit, the outlier tends to pull the fitted line up toward the outlying point. In other words, as the model tries to minimize the squared distance between it and every point, a few outlying points tend to exert a disproportionate influence on the model's characteristics.

Another way to look at this is that the model tries to fit all the training data points to the best of its ability or causes *overfitting*. A consequence of overfitting is that the overall error on the training data will be minimized, but if the same model is tried on new data points (which were not used for training), the error tends to increase till it's out of proportion. One symptom of overfitting is the generation of large coefficients. In the example above, these coefficients are now $b_0 = 2.14$ and $b_1 = 6.93$, that is, they have increased by a factor of nearly 2 or more in each case

In order to avoid overfitting by ensuring that none of the weights or coefficients become large, one can add a penalty factor to the cost function that penalizes large weights. This process is known as *Ridge regression or L2-norm regularization*. The penalty includes the sum of the squared magnitude of all weights, $||b||^2 = b^2{}_1 + b^2{}_2 + \ldots$, that is, *L2-Norm* of b_m, where m is the number of attributes.

The cost function is modified as shown:

$$J_{RIDGE} = \Sigma N_i = (y_i - B^T x_i)^2 + \lambda |b|^2 \tag{5.14}$$

By following the usual steps and switching to a matrix form shown earlier, one arrives at the new solution for the weights as:

$$B = (\lambda I + X^T X)^{-1} X^T Y \tag{5.15}$$

where I is the identity matrix and λ is a penalty factor > 0.

Compare this to the standard solution that can be derived from Eq. (5.11):

$$B = (X^T X)^{-1} Y^T X. \tag{5.16}$$

When L2-norm is implemented, the fit looks much improved and the coefficients are $b_0 = 1.5$ and $b_1 = 4.02$ which are closer to the underlying function (shown in Fig. 5.14).

Ridge regression tends to push all the weights toward zero in order to minimize the cost function.

There is another scenario of overfitting, that involves selecting more than the optimal number of independent variables (and, thus, weights). Not all features exert the same influence on the predicted outcome, some features have more influence than others. However, as more features are included in a model, the training error continues to reduce. But the test error may spiral out of control and result in another form of overfitting.

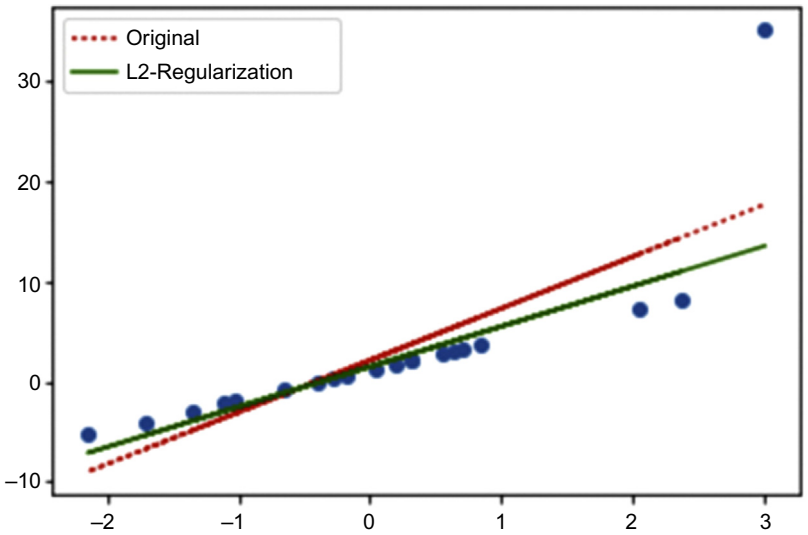


FIGURE 5.14
L2-regularization.

Lasso regression or *L1-norm regularization* addresses this concern where the goal is to select the optimal number of features. The formulation is similar to Ridge, but using the L1-norm: $||b|| = |b_1| + |b_2| + \cdots$

$$J_{LASSO} = \Sigma N_i = (y_i - B^T x_i)^2 + \lambda |b| \tag{5.17}$$

Sometimes the features are correlated with each other—when this happens the X^TX matrix becomes singular and the closed form solution cannot be used. However, this should not be any concern if gradient descent is used to approximate the solution. Gradient descent is a technique that allows us to incrementally evaluate the coefficients, **b** for which the error **J** is minimized, when obtaining a closed form derivative for **dJ/db** is not possible. The idea is to take small computational steps toward the direction of minimum **J** by choosing the fastest path. In this case we can however get the derivative of the error function in closed form, which turns out to be:

$$\partial J / \partial b = -2X^T Y + 2X^T Xb + \lambda \text{sign}(b) = 0 \tag{5.18}$$

here $\text{sign}(b) = 1$ if $b > 0$, -1 if $b < 0$, 0 if $b = 0$.

Typically for practical situations, such closed form derivatives are seldom available and the gradient descent formulation is the alternative. The final gradient descent setup for both types of regularization are:

LASSO or L1:

$$b_{i+1} = b_i - \eta X^T(Y - \hat{Y}) + \lambda \times \text{sign}(b_i) \tag{5.19}$$

RIDGE or L2:

$$b_{i+1} = b_i - \eta X^T(Y - \hat{Y}) + \lambda \times b_i \tag{5.20}$$

where η is the same *learning rate* parameter that crops up in neural networks (Chapter 4: Classification) and deep learning (Chapter 10). The number of steps, i is determined by the rate of convergence of the solution or by other stopping criteria. In RapidMiner, Rigde regression is implemented by providing a non-zero penalty factor in the box for 'ridge' under parameters—see Fig. 5.4.

5.2 LOGISTIC REGRESSION

From a historical perspective, there are two main classes of data science techniques: those that evolved (Cramer, 2002) from statistics (such as regression) and those that emerged from a blend of statistics, computer science, and mathematics (such as classification trees). Logistic regression arose in the mid-twentieth century as a result of the simultaneous development of the concept of the *logit* in the field of biometrics and the advent of the digital computer, which made computations of such terms easy. So, to understand logistic regression, the logit concept first needs to be explored. The chart in Fig. 5.15, adapted from data shown in Cramer (2002) shows the evolving trend from initial acceptance of the logit concept in the mid-1950s to the surge in references to this concept toward the latter half of the twentieth century. The chart is an indicator of how important logistic regression has become over the last few decades in a variety of scientific and business applications.

To introduce the logit, a simple example will be used. Recall that linear regression is the process of finding a function to fit the x's that vary linearly with y with the objective of being able to use the function as a model for prediction. The key assumption here are that both the predictor and target variables are continuous, as seen in the chart in Fig. 5.16. Intuitively, one can state that when x increases, y increases along the slope of the line. For example, advertisement spend and sales.

What happens if the target variable is not continuous? Suppose the target variable is the response to advertisement campaigns—if more than a threshold number of customers buy for example, then the response is considered

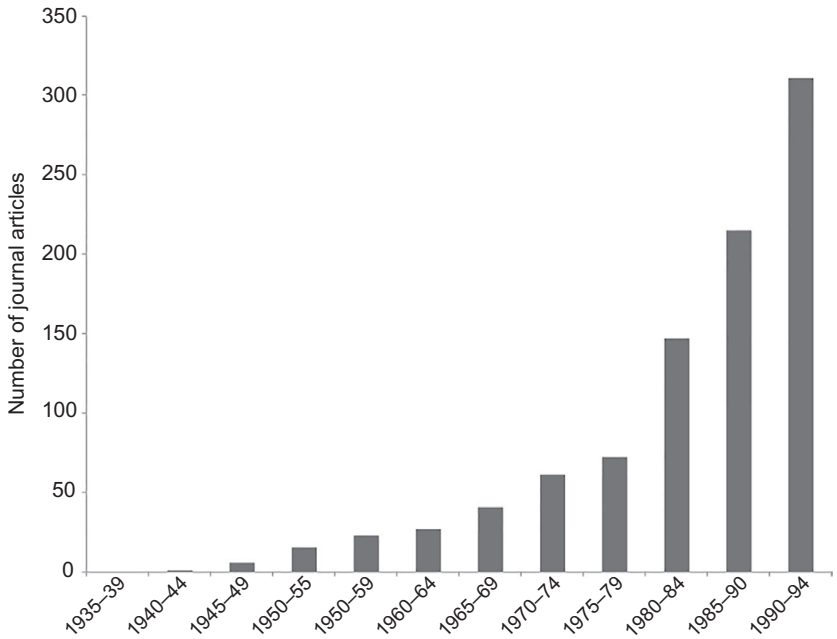
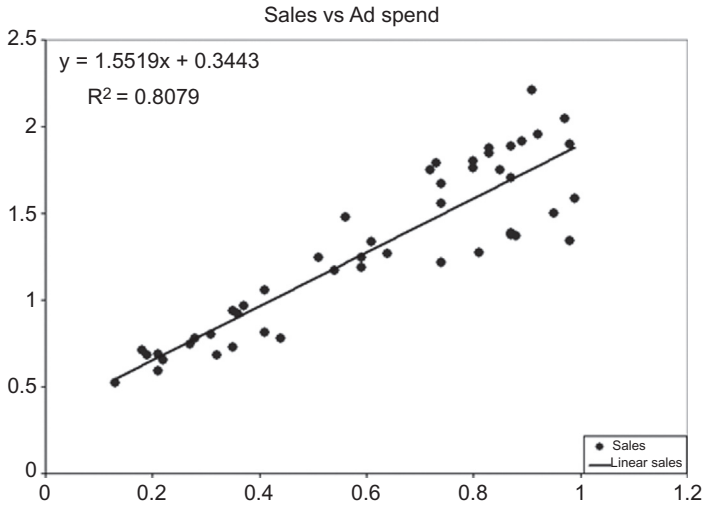


FIGURE 5.15
Growth of logistic regression applications in statistical research.



Linear Regression Model. We can make an intuitive assessment that increase in *Ad spend* also increases *Sales*. Using the straight line, we may also be able to predict.

FIGURE 5.16
Goal of linear regression.

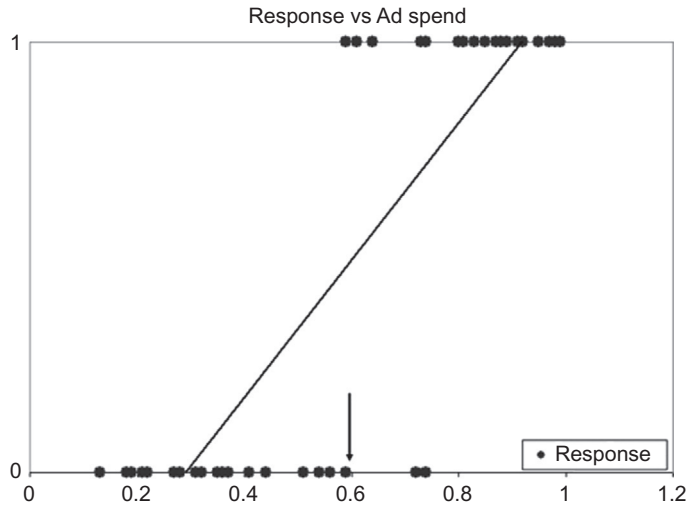


FIGURE 5.17
Fitting a linear model to discrete data.

to be 1; if not the response is 0. In this case, the target (y) variable is discrete (as in Fig. 5.17); the straight line is no longer a fit as seen in the chart. Although one can still estimate—approximately—that when x (advertising spend) increases, y (response or no response to a mailing campaign) also increases, there is no gradual transition; the y value abruptly jumps from one binary outcome to the other. Thus, the straight line is a poor fit for this data.

On the other hand, take a look at the S-shaped curve in Fig. 5.18. This is certainly a better fit for the data shown. If the equation to this "sigmoid" curve is known, then it can be used as effectively as the straight line was used in the case of linear regression.

Logistic regression is, thus, the process of obtaining an appropriate nonlinear curve to fit the data when the target variable is discrete. How is the sigmoid curve obtained? How does it relate to the predictors?

5.2.1 How It Works

The dependent variable, y, will be re-examined. If it is binomial, that is, it can take on only two values (yes/no, pass/fail, respond/does not respond, and so on), then y can be coded to assume only two values: 1 or 0.

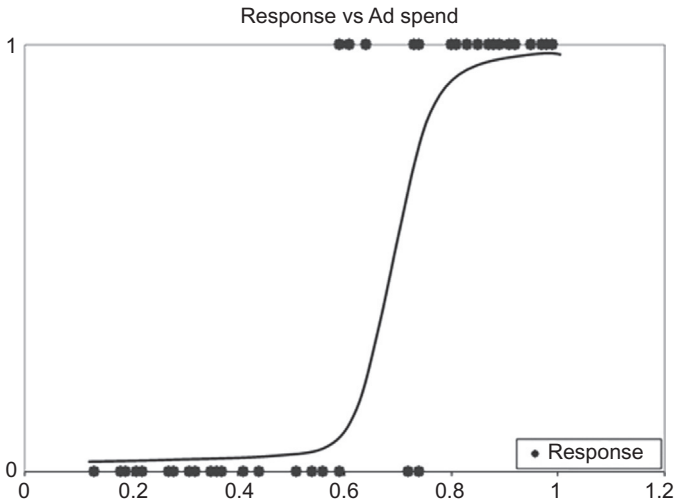


FIGURE 5.18
Fitting a nonlinear curve to discrete data.

The challenge is to find an equation that functionally connects the predictors, x, to the outcome y where y can only take on two values: 0 or 1. However, the predictors themselves may have no restrictions: they could be continuous or categorical. Therefore, the functional range of these unrestricted predictors is likely to also be unrestricted (between $-\infty$ to $+\infty$). To overcome this problem, one must map the continuous function to a discrete function. This is what the logit helps to achieve.

How Does Logistic Regression Find the Sigmoid Curve?

As observed in Eq. (5.1), a straight line can be depicted by only two parameters: the slope (b_1) and the intercept (b_0). The way in which x's and y are related to each other can be easily specified by b_0 and b_1. However, an S-shaped curve is a much more complex shape and representing it parametrically is not as straightforward. So how does one find the mathematical parameters to relate the x's to the y?

It turns out that if the target variable y is transformed to the *logarithm of the odds of* y, then the *transformed* target variable is *linearly* related to the predictors, x. In most cases where the use of logistic regression is needed; the y is usually a yes/no type of response. This is usually interpreted as the

probability of an event happening ($y = 1$) or not happening ($y = 0$). This can be deconstructed as:

- If y is an event (response, pass/fail, etc.),
- and p is the probability of the event happening ($y = 1$),
- then $(1 - p)$ is the probability of the event *not* happening ($y = 0$),
- and $p/(1 - p)$ are the *odds* of the event happening.

The logarithm of the odds, $\log(p/(1 - p))$ is linear in the predictors, X, and $\log(p/(1 - p))$ or the log of the odds is called the *logit function*.

The logit can be expressed as a linear function of the predictors X, similar to the linear regression model shown in Eq. (5.1) as:

$$\text{logit} = \log p/(1 - p) = b_0 x + b_1 \tag{5.21}$$

For a more general case, involving multiple independent variables, x, there is:

$$\text{logit} = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \tag{5.22}$$

The logit can take any value from $-\infty$ to $+\infty$. For each row of predictors in a dataset, the logit an now be computed. From the logit, it is easy to then compute the probability of the response y (occurring or not occurring) as seen below:

$$p = e^{\text{logit}}/(1 + e^{\text{logit}}) \tag{5.23}$$

The logistic regression model from Eq. (5.22) ultimately delivers the probability of y occurring (i.e., $y = 1$), given specific value(s) of x via Eq. (5.23). In that context, a good definition of logistic regression is that it is a mathematical modeling approach in which a best-fitting, yet least-restrictive model is selected to describe the relationship between several independent explanatory variables and a dependent binomial response variable. It is least-restrictive because the right side of Eq. (5.22) can assume any value from $-\infty$ to $+\infty$. Cramer (2002) provides more background on the history of the logit function.

From the data given the x's are known and using Eqs. (5.22) and (5.23) one can compute the p for any given x. But to do that, the coefficients first need to be determined, b, in Eq. (5.22). How is this done? Assume that one starts out with a trial of values for b. Given a training data sample, one can compute the quantity:

$$p^y \times (1-p)^{(1-y)}$$

where y is the original outcome variable (which can take on 0 or 1) and p is the probability estimated by the logit equation [Eq. (5.23)]. For a specific training sample, if the actual outcome was $y = 0$ and the model estimate of p was high (say 0.9), that is, the model was wrong, then this quantity reduces

to 0.1. If the model estimate of probability was low (say 0.1), that is, the model was good, then this quantity increases to 0.9. Therefore, this quantity, which is a simplified form of a *likelihood* function, is *maximized for good estimates* and *minimized for poor estimates*. If one computes a summation of the simplified likelihood function across *all* the training data samples, then a high value indicates a good model (or good fit) and vice versa.

In reality, gradient descent or other nonlinear optimization techniques are used to search for the coefficients, b, with the objective of maximizing the likelihood of correct estimation (or $p^y \times (1 - p)^{(1 - y)}$, summed over all training samples). More sophisticated formulations of likelihood estimators are used in practice (Eliason, 1993).

A Simple but Tragic Example

In the 1912 shipwreck of the HMS Titanic, hundreds of people perished as the ship struck an iceberg in the North Atlantic (Hinde, 1998). When the data is dispassionately analyzed, a couple of basic patterns emerge. 75% of the women and 63% of the first-class passengers survived. If a passenger was a woman and if she traveled first class, her probability of survival was 97%! The scatterplot in Fig. 5.19 depicts this in an easy to understand way (see the bottom left cluster).

A data science competition used the information from this event and challenged analysts to develop an algorithm that could classify the passenger list into survivors and non-survivors.[3] The training dataset provided will be used as an example to demonstrate how logistic regression could be employed to make this prediction and also to interpret the coefficients from the model.

Table 5.3 shows part of a reduced dataset consisting only of three variables: travel class of the passenger (pclass = 1st, 2nd, or 3rd), sex of the passenger (0 for male and 1 for female), and the label variable "survived" (true or false). When a logistic regression model is fit to this data consisting of 891 samples, the following equation is obtained for predicting the class *survived = false* (the details of a generic setup process will be described in the next section):

$$\text{logit} = -0.6503 - 2.6417 \times \text{sex} + 0.9595 \times \text{pclass} \tag{5.24}$$

Comparing this to Eq. (5.22), $b_0 = -0.6503$, $b_1 = -2.6417$, and $b_2 = 0.9595$. How are these coefficients interpreted? In order to do this, Eq. (5.23) will need to be recalled,

[3] http://www.kaggle.com/c/titanic-gettingStarted.

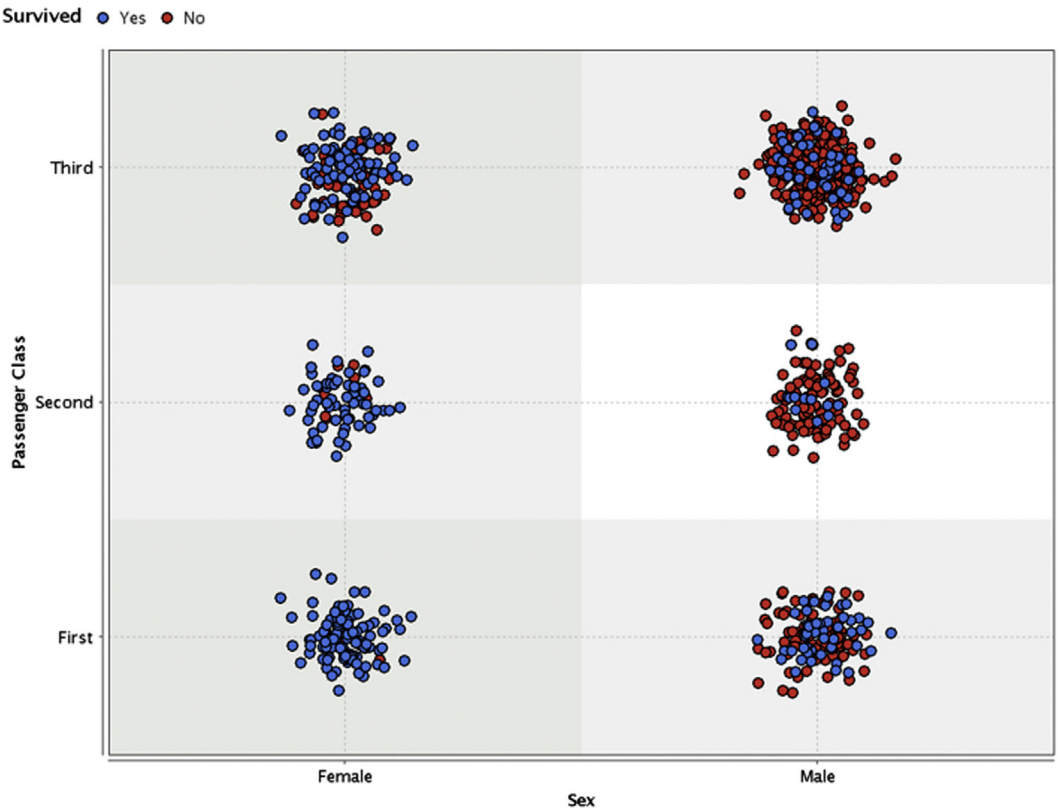


FIGURE 5.19
Probability of survival in the Titanic wreck based on gender and travel class.

$$p = e^{\text{logit}}/[1 + e^{\text{logit}}]$$

which indicates that as logit increases to a large positive quantity, the probability that the passenger did not survive (survived = false) approaches 1. More specifically, when logit approaches $-\infty$, p approaches 0 and when logit approaches $+\infty$, p approaches 1. The negative coefficient on variable *sex* indicates that this probability reduces for females (*sex* = 1) and the positive coefficient on variable p indicates that the probability of not surviving (*survived* = false) increases the higher the *number* of the travel class. This verifies the intuitive understanding that was provided by the scatterplot shown in Fig. 5.19.

The odds form of the logistic regression model can also be examined, which is given as:

$$\text{odds (survived = false)} = e^{-0.6503} \times 2.6103^{\text{pclass}} \times 0.0712^{\text{sex}} \tag{5.25}$$

Table 5.3 Portion of the Dataset From the Titanic Example

pclass	Sex	Survived?
3.0	Male	0.0
1.0	Female	1.0
3.0	Female	1.0
1.0	Female	1.0
3.0	Male	0.0
3.0	Male	0.0
1.0	Male	0.0
3.0	Male	0.0
3.0	Female	1.0
2.0	Female	1.0
3.0	Female	1.0
1.0	Female	1.0
3.0	Male	0.0
3.0	Male	0.0
3.0	Female	0.0
2.0	Female	1.0
3.0	Male	0.0

Recall that logit is simply given by log(odds) and essentially the same equation as Eq. (5.24) is used. A key fact to observe is that a positive coefficient in the logit model translates into a coefficient higher than 1 in the odds model (the number 2.6103 in the above equation is $e^{0.9595}$ and 0.0712 is $e^{-2.6417}$) and a negative coefficient in the logit model translates into coefficients smaller than 1 in the odds model. Again, it is clear that the odds of not surviving increases with travel class and reduces with gender being female.

An *odds ratio analysis* will reveal the value of computing the results in this format. Consider a female passenger ($sex = 1$). The survivability for this passenger could be calculated if she was in 1st class ($pclass = 1$) versus if she was in 2nd class as an odds ratio:

$$\begin{aligned}&\text{odds (survived = false 2nd class)}/\text{odds(survived = false 1st class)}\\&\quad = 2.6103^2/2.6103^1 = 2.6103\end{aligned} \tag{5.26}$$

Based on the Titanic dataset, the odds that a female passenger would not survive if she was in 2nd class increases by a factor of 2.6 compared to her odds if she was in 1st class. Similarly, the odds that a female passenger would not survive increases by nearly seven times if she was in 3rd class! In the next section, the mechanics of logistic regression are discussed as well as the process of implementing a simple analysis using RapidMiner.

Table 5.4 A Sample From the Loan Default Dataset

[Busage]	[Daysdelq]	[Default]
87.0	2.0	N
89.0	2.0	N
90.0	2.0	N
90.0	2.0	N
101.0	2.0	N
110.0	2.0	N
115.0	2.0	N
115.0	2.0	N
115.0	2.0	N
117.0	2.0	N

5.2.2 How to Implement

The data used come from an example for a credit scoring exercise. The objective is to predict DEFAULT (Y or N) based on two predictors: loan age (business age) and number of days of delinquency. There are 100 samples Table 5.4.

Step 1: Data Preparation

Load the spreadsheet into RapidMiner. Remember to set the DEFAULT column as Label. Split the data into training and test samples using the *Split Validation* operator.

Step 2: Modeling Operator and Parameters

Add the *Logistic Regression* operator in the training subprocess of the *Split Validation* operator. Add the *Apply Model* operator in the testing subprocess of the *Split Validation* operator. Just use default parameter values. Add the *Performance (Binominal)* evaluation operator in the testing subprocess of *Split Validation* operator. Check the Accuracy, AUC, Precision, and Recall boxes in the parameter settings. Connect all ports as shown in Fig. 5.20.

Step 3: Execution and Interpretation

Run the model and view the results. In particular, check for the kernel model, which shows the coefficients for the two predictors and the intercept. The bias (offset) is -1.820 and the coefficients are given by: w[BUSAGE] = 0.592 and w[DAYSDELQ] = 2.045. Also check the confusion matrix for Accuracy, Precision, and Recall and finally view the ROC curves and check the area under the curve or AUC. Chapter 8, Model Evaluation, provides further details on these important performance measures.

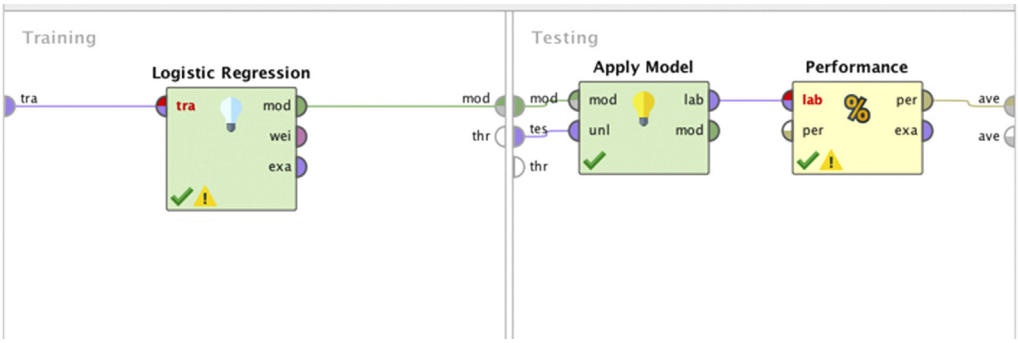


FIGURE 5.20
Setting up the RapidMiner process for a logistic regression model.

accuracy: 83.33%

	true N	true Y	class precision
pred. N	21	3	87.50%
pred. Y	2	4	66.67%
class recall	91.30%	57.14%	

FIGURE 5.21
Confusion matrix for the testing sample.

The accuracy of the model based on the 30% testing sample is 83%. (The ROC curves have an AUC of 0.863). The next step would be to review the kernel model and prepare for deploying this model. Are these numbers acceptable? In particular, pay attention to the class recall (bottom row of the confusion matrix in Fig. 5.21). The model is quite accurate in predicting if someone is NOT a defaulter (91.3%), however, its performance when it comes to identifying if someone IS a defaulter is questionable. For most predictive applications, the cost of wrong class predictions is not uniform. That is, a false positive (in the case of identifying someone as a defaulter, when they are not) may be less expensive than a false negative (in the case of identifying someone as a non-defaulter, when they actually are). There are ways to weight the cost of misclassification, and RapidMiner allows this through the use of the *MetaCost* operator.

Step 4: Using MetaCost

Nest the *Logistic Regression* operator inside a *MetaCost* operator to improve class recall. The *MetaCost* operator is now placed inside *Split Validation* operator. Configure the *MetaCost* operator as shown in Fig. 5.22. Notice that false

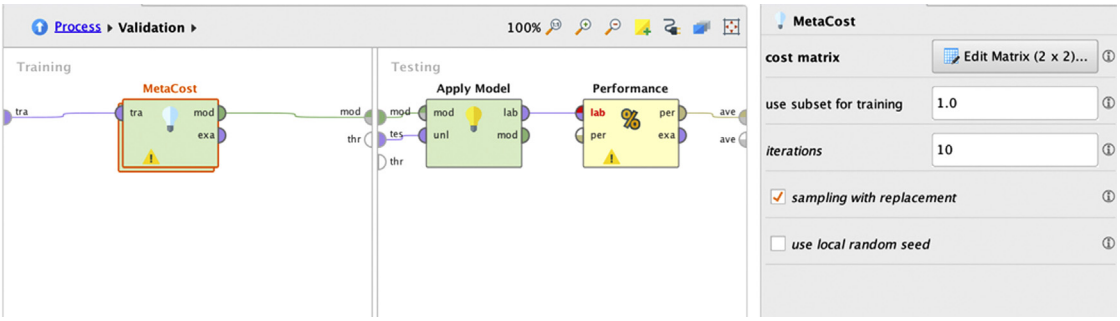


FIGURE 5.22
Configuring the MetaCost operator to improve class recall performance.

accuracy: 83.33%

	true N	true Y	class precision
pred. N	20	2	90.91%
pred. Y	3	5	62.50%
class recall	86.96%	71.43%	

FIGURE 5.23
Improved classification performance with the usage of MetaCost operator.

negatives have twice the cost of false positives. The actual values of these costs can be further optimized using an optimization loop—optimization is discussed for general cases in Chapter 15, Getting Started with RapidMiner.

When this process is run, the new confusion matrix that results is shown in Fig. 5.23. The overall accuracy has not changed much. Note that while the class recall for the Default = Yes class has increased from 57% to 71%, this has come at the price of reducing the class recall for Default = No from 91% to 87%. Is this acceptable? Again, the answer to this comes from examining the actual business costs. More details about interpreting the confusion matrix and evaluating the performance of classification models is provided in Chapter 8, Model Evaluation.

Step 5: Applying the Model to an Unseen Dataset

In RapidMiner, logistic regression is calculated by creating a support vector machine (SVM) with a modified loss function (Fig. 5.24). SVMs were introduced in Chapter 4 on Classification. That's the reason why support vectors are seen at all.

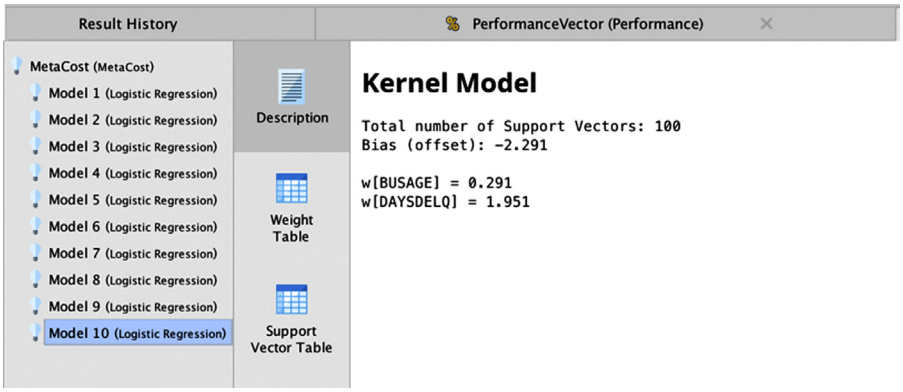


FIGURE 5.24
Default logistic regression model in RapidMiner is based on SVM. *SVM*, Support vector machine.

5.2.3 Summary Points

- Logistic regression can be considered equivalent to using linear regression for situations where, when the target (or dependent) variable is discrete, that is, not continuous. In principle, the response variable or label is binomial. A binomial response variable has two categories: Yes/No, Accept/Not Accept, Default/Not Default, and so on. Logistic regression is ideally suited for business analytics applications where the target variable is a binary decision (fail/pass, response/no response, etc.).
- Logistic regression comes from the concept of the logit. The logit is the logarithm of the odds of the response, y, expressed as a function of independent or predictor variables, x, and a constant term. That is, for example, log (odds of $y = \text{Yes}) = b_1 x + b_0$.
- This logit gives the odds of the Yes event, however if one wants probabilities, the transformed equation will need to be used:
  $p(y = \text{“Yes”}) = 1/(1 + e^{(-b1_x - b_0)})$
- The predictors can be either numerical or categorical for standard logistic regression. However, in RapidMiner, the predictors can only be numerical, because it is based on the SVM formulation.

5.3 CONCLUSION

This chapter explored two of the most common function-fitting methods. Function-fitting methods are one of the earliest data science techniques based on the concept of supervised learning. The multiple linear regression model

CHAPTER 7

Clustering

Clustering is the process of finding meaningful groups in data. In clustering, the objective is not to predict a target class variable, but to simply capture the possible natural groupings in the data. For example, the customers of a company can be grouped based on purchase behavior. In the past few years, clustering has even found a use in political elections (Pearson & Cooper, 2012). The prospective electoral voters can be clustered into different groups so that candidates can tailor the messages to resonate within each group. the difference between classification and clustering can be illustrated with an example. Categorizing a *given* voter as a "soccer mom" (a known user group) or not, based on previously available labeled data, is supervised learning—Classification. Segregating a population of electorates into different groups, based on similar demographics is unsupervised learning—Clustering. The process of identifying whether a data point belongs to a particular known group is classification. The process of dividing the dataset into meaningful groups is clustering. In mostcases, one would not know ahead what groups to look for and, thus, the inferred groups might be difficult to explain. The task of clustering can be used in two different classes of applications: to *describe* a given dataset and as a *preprocessing* step for other data science algorithms.

CLUSTERING TO DESCRIBE THE DATA

The most common application of clustering is to explore the data and find all the possible meaningful groups in the data. Clustering a company's customer records can yield a few groups in such a way that customers within a group are more like each other than customers belonging to a different group. Depending on the clustering technique used, the number of groups or clusters is either user-defined or automatically determined by the algorithm from the dataset. Since clustering is not about predicting the membership of a customer in a well-defined meaningful group (e.g., frequent high-volume purchaser), the similarities of customers within a group need

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00007-1

to be carefully investigated to make sense of the group as a whole. Some of the common applications of clustering to describe the underlying natural structure of data are:

1. ***Marketing:*** Finding the common groups of customers based on all past customer behaviors, and/or purchase patterns. This task is helpful to segment the customers, identify prototype customers (description of a typical customer of a group), and to tailor a marketing message to the customers in a group.
2. ***Document clustering:*** One common text mining task is to automatically group documents (or text blobs) into groups of similar topics. Document clustering provides a way of identifying key topics, comprehending and summarizing these clustered groups rather than having to read through whole documents. Document clustering is used for routing customer support incidents, online content sites, forensic investigations, etc.
3. ***Session grouping:*** In web analytics, clustering is helpful to understand common groups of clickstream patterns and to discover different kinds of clickstream profiles. One clickstream profile may be that of a customer who knows what they want and proceeds straight to checkout. Another profile may be that of a customer who has researched the products, read through customer reviews, and then makes a purchase during a later session. Clustering the web sessions by profile helps the e-commerce company provide features fitting each customer profile.

CLUSTERING FOR PREPROCESSING

Since a clustering process considers all the attributes of the dataset and *reduces* the information to a cluster, which is really another attribute (i.e., the ID of the cluster to which a record would belong to), clustering can be used as a data compression technique. The output of clustering is the cluster ID for each record and it can be used as an input variable for other data science tasks. Hence, clustering can be employed as a preprocessing technique for other data science processes. In general, clustering can be used for two types of preprocessing:

1. ***Clustering to reduce dimensionality:*** In an n-dimensional dataset (n number of attributes), the computational complexity is proportional to the number of dimensions or "*n*." With clustering, *n*-dimensional attributes can be converted or reduced to one categorical attribute—"Cluster ID." This reduces the complexity, although there will be some loss of information because of the dimensionality reduction to one

single attribute. Chapter 14, Feature Selection provides an in-depth look at feature selection techniques.

2. ***Clustering for object reduction:*** Assume that the number of customers for an organization is in the millions and the number of cluster groups is 100. For each of these 100 cluster groups, one "poster child" customer can be identified that represents the typical characteristics of all the customers in that cluster group. The poster child customer can be an actual customer or a fictional customer. The prototype of a cluster is the most common representation of all the customers in a group. Reducing millions of customer records to 100 prototype records provides an obvious benefit. In some applications, instead of processing millions of records, just the prototypes can be processed for further classification or regression tasks. This greatly reduces the record count and the dataset can be made appropriate for classification by algorithms like *k*-nearest neighbor (*k*-NN) where computation complexity depends on the number of records.

TYPES OF CLUSTERING TECHNIQUES

Regardless of the types of clustering applications, the clustering process seeks to find groupings in data, in such a way that data points within a cluster are more *similar* to each other than to data points in the other clusters (Witten & Frank, 2005). One common way of measuring similarity is the Euclidean distance measurement in *n*-dimensional space. In Fig. 7.1 all data points in Cluster 2 are closer to other data points in Cluster 2 than to other data points in Cluster 1. Before explaining the different ways to implement clustering, the different types of clusters have to be defined. Based on a data point's membership to an identified group, a cluster can be:

- ***Exclusive or strict partitioning clusters***: Each data object belongs to one exclusive cluster, like the example shown in Fig. 7.1. This is the most common type of cluster.
- ***Overlapping clusters***: The cluster groups are not exclusive, and each data object may belong to more than one cluster. These are also known as multi-view clusters. For example, a customer of a company can be grouped in a high-profit customer cluster and a high-volume customer cluster at the same time.
- ***Hierarchical clusters***: Each child cluster can be merged to form a parent cluster. For example, the most profitable customer cluster can be further divided into a long-term customer cluster and a cluster with new customers with high-value purchases.

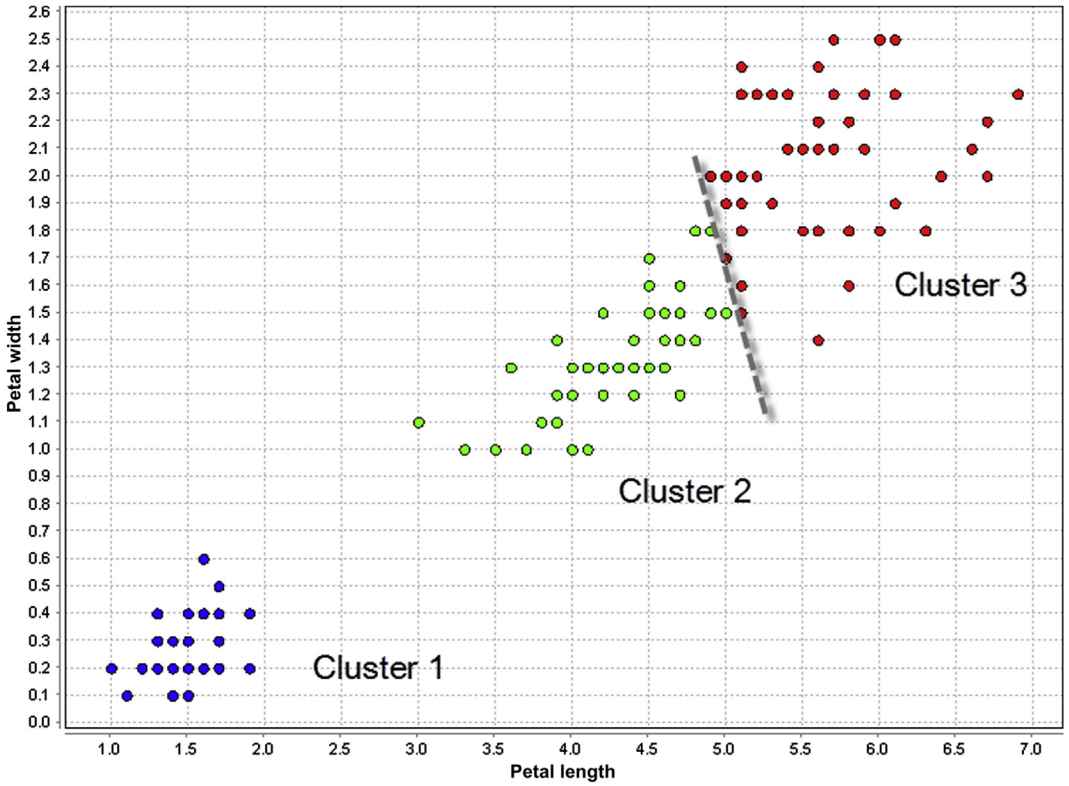


FIGURE 7.1
Example of a clustering of the Iris dataset without class labels.

- ***Fuzzy or probabilistic clusters***: Each data point belongs to all cluster groups with varying degrees of membership from 0 to 1. For example, in a dataset with clusters A, B, C, and D, a data point can be associated with all the clusters with degree A = 0.5, B = 0.1, C = 0.4, and D = 0. Instead of a definite association of a data point with one cluster, fuzzy clustering associates a probability of membership to all the clusters.

Clustering techniques can also be classified based on the algorithmic approach used to find clusters in the dataset. Each of these classes of clustering algorithms differ based on what relationship they leverage between the data objects.

- ***Prototype-based clustering***: In the prototype-based clustering, each cluster is represented by a central data object, also called a prototype. The prototype of each cluster is usually the center of the cluster, hence, this clustering is also called centroid clustering or center-based clustering. For example, in clustering customer segments, each customer

cluster will have a central prototype customer and customers with similar properties are associated with the prototype customer of a cluster.

- ***Density clustering*:** In Fig. 7.1, it can be observed that clusters occupy the area where there are more data points per unit space and are separated by sparse space. A cluster can also be defined as a dense region where data objects are concentrated surrounded by a low-density area where data objects are sparse. Each dense area can be assigned a cluster and the low-density area can be discarded as noise. In this form of clustering not all data objects are clustered since noise objects are unassigned to any cluster.
- ***Hierarchical clustering*:** Hierarchical clustering is a process where a cluster hierarchy is created based on the distance between data points. The output of a hierarchical clustering is a *dendrogram*: a tree diagram that shows different clusters at any point of precision which is specified by the user. There are two approaches to create a hierarchy of clusters. A bottom-up approach is where each data point is considered a cluster, and the clusters are merged to finally form one massive cluster. The top-down approach is where the dataset is considered one cluster and they are recursively divided into different sub-clusters until individual data objects are defined as separate clusters. Hierarchical clustering is useful when the data size is limited. A level of interactive feedback is required to cut the dendrogram tree to a given level of precision.
- ***Model-based clustering*:** Model-based clustering gets its foundation from statistics and probability distribution models; this technique is also called distribution-based clustering. A cluster can be thought of as a grouping that has the data points belonging to the same probability distribution. Hence, each cluster can be represented by a distribution model (like Gaussian or Poisson), where the parameter of the distribution can be iteratively optimized between the cluster data and the model. With this approach, the entire dataset can be represented by a mixture of distribution models. *Mixture of Gaussians* is one of the model-based clustering techniques used where a fixed number of distributions are initialized, and parameters are optimized to fit the cluster data.

In the rest of the chapter, the common implementations of clustering will be discussed. First, *k*-means clustering will be covered, which is a kind of prototype-clustering technique. This is followed by the Density-Based Spatial Clustering of Applications with Noise (DBSCAN), which provides a view into density clustering, and the chapter is concluded with an explanation of a novel approach called self-organizing maps (SOMs).

SEGMENTING CUSTOMER RECORDS

All business entities record most of their interactions with customers, including but not limited to monetary transactions, customer service, customer location and details, online interactions, product usage, and warranty and service information. Take the telecommunications industry as an example. Telecommunications companies have now evolved to provide multiple services to different types of customers by packaging products like phones, wireless, internet, data communications, corporate backbones, entertainment content, home security, etc. To better understand customers, whose numbers often range in the millions, it is necessary to combine multiple datasets about the customers and their interactions with each product. The vastness of the number and variety of attributes in the dataset provides both an opportunity and challenge to better know their customers (Berry & Linoff, 2000a,b). One logical way to understand the customer beyond straightforward classifications like customer type (residential, corporate, government, etc.) or revenue volume (high-, medium-, and low-revenue customers), is to segment the customer based on usage patterns, demographics, geography, and behavior patterns for product usage.

For a customer segmentation task, the data need to be prepared in such a way that each record (row) is associated with each customer and the columns contain all the attributes about the customer, including demographics, address, products used, revenue details, usage details of the product, call volume, type of calls, call duration, time of calls, etc. Table 7.1 shows an example structure of a denormalized customer dataset. Preparing this dataset is going to be a time-consuming task. One of the obvious methods of segmentation is stratifying based on any of the existing attributes. For example, one can segment the data based on a customer's geographical location.

A clustering algorithm consumes this data and groups the customers with similar patterns into clusters based on all the attributes. Based on the data, clustering could be based on a combination of call usage, data patterns, and monthly bills. The resulting clusters could be a group of customers who have low data usage but with high bills at a location where there is weak cellular coverage, which may indicate dissatisfied customers.

The clustering algorithm doesn't explicitly provide the reason for clustering and doesn't intuitively label the cluster groups. While clustering can be performed using a large number of attributes, it is up to the practitioner to carefully select the attributes that will be relevant for clustering. Automated feature selection methods can reduce the dimensions for a clustering exercise. Clustering could be iteratively developed further by selecting or ignoring other attributes in the customer dataset.

Table 7.1 Dataset for Customer Segmentation

Customer ID	Location	Demographics	Call Usage	Data Usage (MB)	Monthly Bill ($)
01	San Jose, CA	Male	1400	200	75.23
02	Miami, FL	Female	2103	5000	125.78
03	Los Angeles, CA	Male	292	2000	89.90
04	San Jose, CA	Female	50	40	59.34

7.1 *K*-MEANS CLUSTERING

k-Means clustering is a prototype-based clustering method where the dataset is divided into *k*-clusters. *k*-Means clustering is one of the simplest and most commonly used clustering algorithms. In this technique, the user specifies the number of clusters (k) that need to be grouped in the dataset.

The objective of *k*-means clustering is to find a *prototype* data point for each cluster; all the data points are then assigned to the nearest prototype, which then forms a cluster. The prototype is called as the *centroid*, the center of the cluster. The center of the cluster can be the mean of all data objects in the cluster, as in *k*-means, or the most represented data object, as in *k*-medoid clustering. The cluster centroid or mean data object does not have to be a real data point in the dataset and can be an imaginary data point that represents the characteristics of all the data points within the cluster.

The *k*-means clustering algorithm is based on the works of Stuart Lloyd and E.W. Forgy (Lloyd, 1982) and is sometimes referred to as the Lloyd–Forgy algorithm or Lloyd's algorithm. Visually, the *k*-means algorithm divides the data space into *k* partitions or boundaries, where the centroid in each partition is the prototype of the clusters. The data objects inside a partition belong to the cluster. These partitions are also called *Voronoi partitions*, and each prototype is a seed in a Voronoi partition. A Voronoi partition is a process of segmenting a space into regions, around a set of points called seeds. All other points are then associated to the nearest seed and the points associated with the seed from a unique partition. Fig. 7.2 shows a sample Voronoi partition around seeds marked as black dots.

k-Means clustering creates *k* partitions in *n*-dimensional space, where *n* is the number of attributes in a given dataset. To partition the dataset, a proximity measure has to be defined. The most commonly used measure for a numeric attribute is the Euclidean distance. Fig. 7.3 illustrates the clustering of the Iris dataset with only the petal length and petal width attributes. This Iris dataset is two-dimensional (selected for easy visual explanation), with numeric attributes and *k* specified as 3. The outcome of *k*-means clustering provides a clear partition space for Cluster 1 and a narrow space for the other two clusters, Cluster 2 and Cluster 3.

7.1.1 How It Works

The logic of finding *k*-clusters within a given dataset is rather simple and always converges to a solution. However, the final result in most cases will be locally optimal where the solution will not converge to the best global solution. The process of *k*-means clustering is similar to Voronoi iteration, where the objective is to divide a space into cells around points. The difference is Voronoi iteration partitions the space, whereas, *k*-means clustering partitions the points in data space. Take the example of a two-dimensional dataset (Fig. 7.4). The step-by-step process of finding three clusters is provided below (Tan, Michael, & Kumar, 2005).

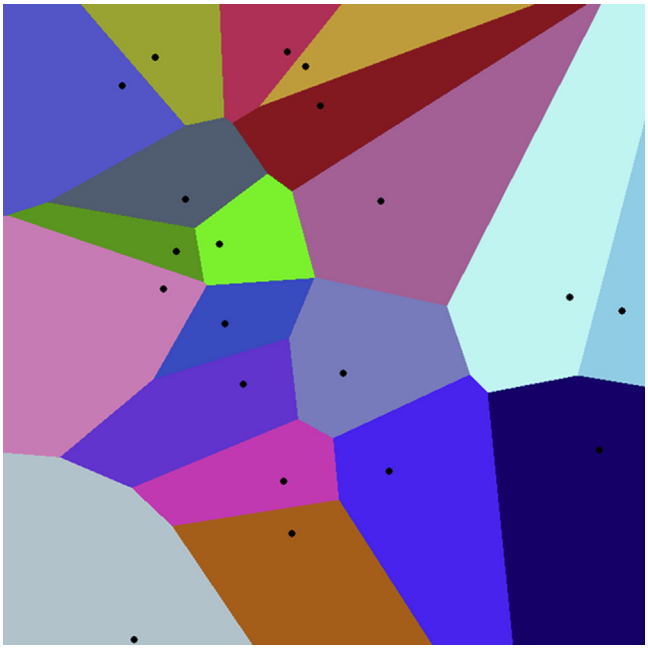


FIGURE 7.2
Voronoi partition "Euclidean Voronoi Diagram" by Raincomplex—personal work. Licensed under Creative Commons Zero, Public Domain Dedication via Wikimedia Commons. http://commons.wikimedia.org/wiki/File:Euclidean_Voronoi_Diagram.png#mediaviewer/File:Euclidean_Voronoi_Diagram.png.[1])

Step 1: Initiate Centroids

The first step in k-means algorithm is to initiate k random centroids. The number of clusters k should be specified by the user. In this case, three centroids are initiated in a given data space. In Fig. 7.5, each initial centroid is given a shape (with a circle to differentiate centroids from other data points) so that data points assigned to a centroid can be indicated by the same shape.

Step 2: Assign Data Points

Once centroids have been initiated, all the data points are now assigned to the nearest centroid to form a cluster. In this context the "nearest" is calculated by a proximity measure. Euclidean distance measurement is the most common proximity measure, though other measures like the Manhattan measure and Jaccard coefficient can be used. The Euclidean distance between two data points X $(x_1, x_2, \ldots, x_n)$ and C $(c_1, c_2, \ldots, c_n)$ with n attributes is given by (7.1)

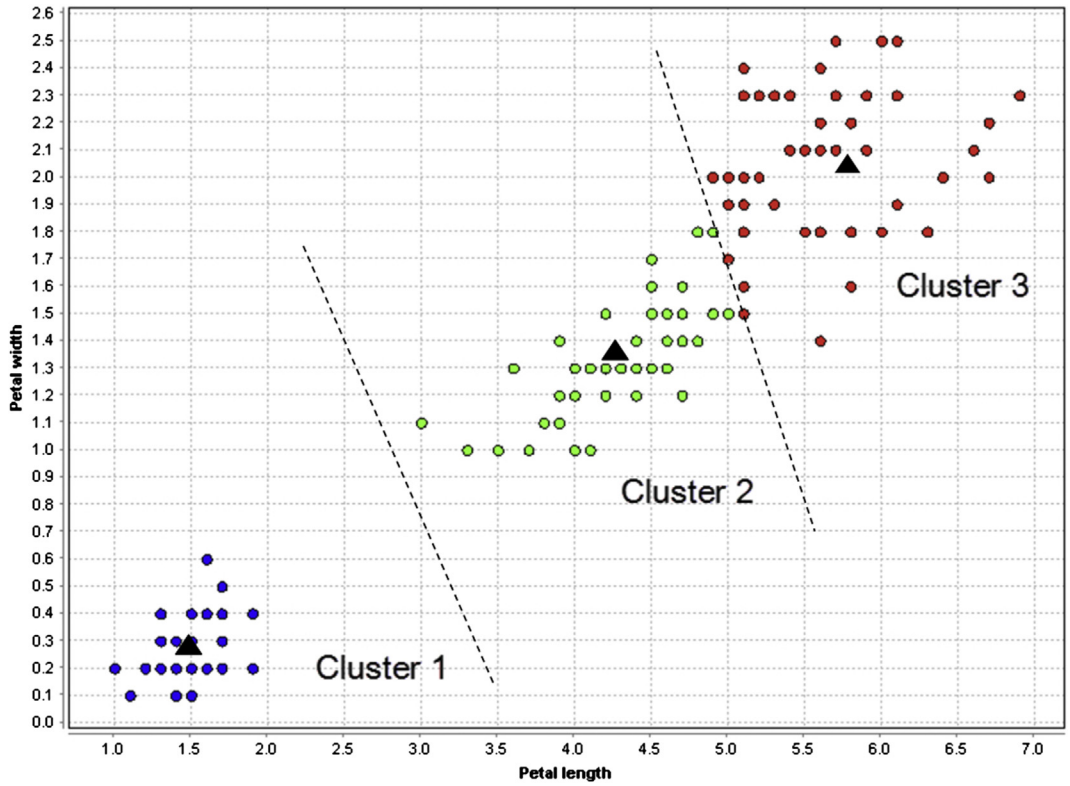


FIGURE 7.3
Prototype-based clustering and boundaries.

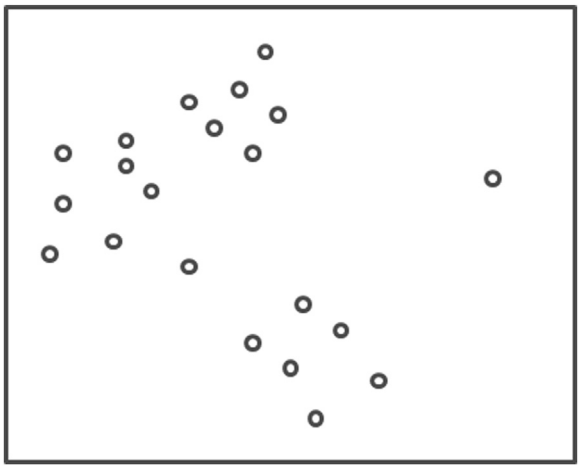


FIGURE 7.4
Dataset with two dimensions.

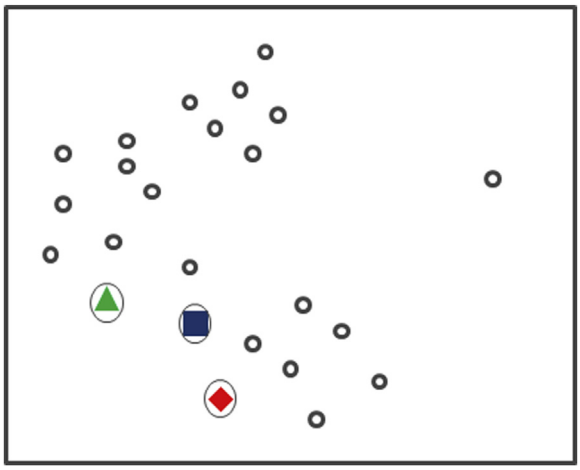


FIGURE 7.5
Initial random centroids.

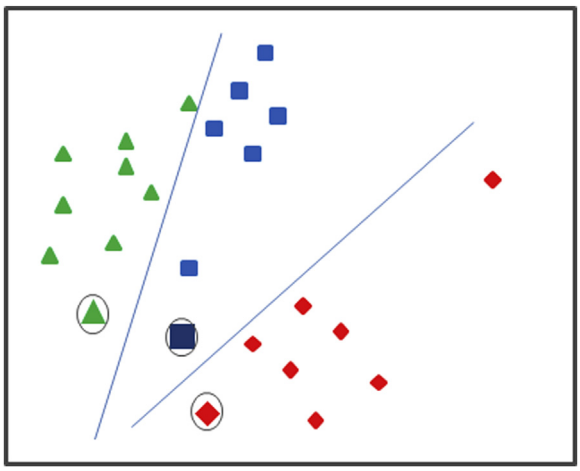


FIGURE 7.6
Assignment of data points to nearest centroids.

$$\text{Distance d} = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + \ldots + (x_n - c_n)^2} \tag{7.1}$$

All the data points associated to a centroid now have the same shape as their corresponding centroid as shown in Fig. 7.6. This step also leads to partitioning of data space into Voronoi partitions, with lines shown as boundaries.

Step 3: Calculate New Centroids

For each cluster, a new centroid can now be calculated, which is also the prototype of each cluster group. This new centroid is the most representative data point of all data points in the cluster. Mathematically, this step can be expressed as minimizing the sum of squared errors (SSEs) of all data points in a cluster to the centroid of the cluster. The overall objective of the step is to minimize the SSEs of individual clusters. The SSE of a cluster can be calculated using Eq. (7.2).

$$\text{SSE} = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - \mu_i \right\|^2 \tag{7.2}$$

where C_i is the i^{th} cluster, j are the data points in a given cluster, μ_i is the centroid for i^{th} cluster, and x_j is a specific data object. The centroid with minimal SSE for the given cluster i is the new mean of the cluster. The mean of the cluster can be calculated using (7.3)

$$\mu_i = \frac{1}{j_i} \sum_{x \in c_i} X \tag{7.3}$$

where X is the data object vector $(x_1, x_2, \ldots, x_n)$. In the case of k-means clustering, the new centroid will be the mean of all the data points. k-Medoid clustering is a variation of k-means clustering, where the median is calculated instead of the mean. Fig. 7.7 shows the location of the new centroids.

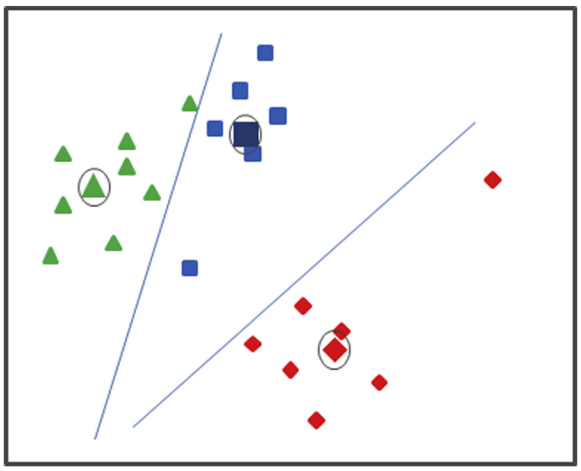


FIGURE 7.7
New centroids.

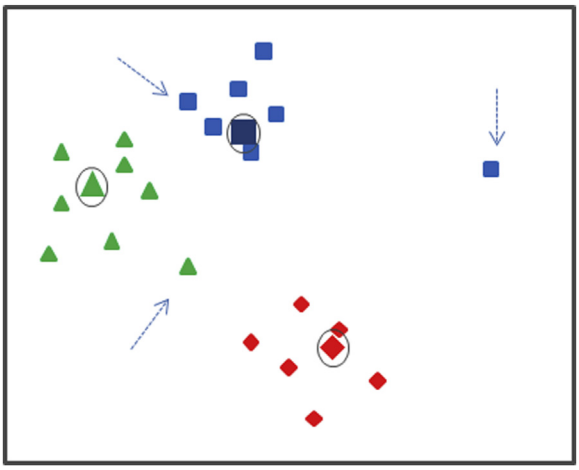


FIGURE 7.8
Assignment of data points to new centroids.

Step 4: Repeat Assignment and Calculate New Centroids

Once the new centroids have been identified, assigning data points to the nearest centroid is repeated until all the data points are reassigned to new centroids. In Fig. 7.8, note the change in assignment of three data points that belonged to different clusters in the previous step.

Step 5: Termination

Step 3—calculating new centroids, and step 4—assigning data points to new centroids, are repeated until no further change in assignment of data points happens. In other words, no significant change in centroids are noted. The final centroids are declared the prototypes of the clusters and they are used to describe the whole clustering model. Each data point in the dataset is now tied with a new clustering ID attribute that identifies the cluster.

Special Cases

Even though *k*-means clustering is simple and easy to implement, one of the key drawbacks of *k*-means clustering is that the algorithm seeks to find a *local optimum*, which may not yield globally optimal clustering. In this approach, the algorithm starts with an initial configuration (centroids) and continuously improves to find the best solution possible for that initial configuration. Since the solution is optimal to the initial configuration (locally optimal), there might be a better optimal solution if the initial configuration changes. The locally optimal solution may not be the most optimal solution for the given clustering problem. Hence, the success of a *k*-means algorithm much depends on the initiation of centroids. This limitation can be

addressed by having multiple random initiations; in each run one could measure the cohesiveness of the clusters by a *performance criterion*. The clustering run with the best performance metric can be chosen as the final run. Evaluation of clustering is discussed in the next section. Some key issues to be considered in *k*-means clustering are:

- ***Initiation:*** The final clustering grouping depends on the random initiator and the nature of the dataset. When random initiation is used, one can run the entire clustering process (also called "runs") with a different set of random initiators and find the clustering process that has minimal total SSE. Another technique is hierarchical clustering, where each cluster is in turn split into multiple clusters and, thereby, minimal SSE is achieved. Hierarchical clustering is further divided into agglomerative or bottom-up clustering and divisive or top-down clustering, depending on how the clustering is initiated. Agglomerative clustering starts with each data point as an individual cluster and proceeds to combine data points into clusters. Divisive clustering starts with the whole dataset as one big cluster and proceeds to split that into multiple clusters.
- ***Empty clusters:*** One possibility in *k*-means clustering is the formation of empty clusters in which no data objects are associated. If empty clusters are formed, a new centroid can be introduced in the cluster that has the highest SSE, thereby, splitting the cluster that contributes to the highest SSE or selecting a new centroid that is at the farthest point away from any other centroid.
- ***Outliers:*** Since SSE is used as an objective function, *k*-means clustering is susceptible to outliers; they drift the centroid away from the representative data points in a cluster. Hence, the prototype is no longer the best representative of the clusters it represents. While outliers can be eliminated with preprocessing techniques, in some applications finding outliers or a group of outliers is the objective of clustering, similar to identifying fraudulent transactions.
- ***Post-processing:*** Since *k*-means clustering seeks to be locally optimal, a few post-processing techniques can be introduced to force a new solution that has less SSE. One could always increase the number of clusters, *k*, and reduce SSE. But, this technique can start overfitting the dataset and yield less useful information. There are a few approaches that can be deployed, such as bisecting the cluster that has the highest SSE and merging two clusters into one even if SSE increases slightly.

Evaluation of Clusters

Evaluation of *k*-means clustering is different from regression and classification algorithms because in clustering there are no known external labels for

comparison. The evaluation parameter will have to be developed from the very dataset that is being evaluated. This is called unsupervised or internal evaluation. Evaluation of clustering can be as simple as computing total SSE. Good models will have low SSE within the cluster and low overall SSE among all clusters. SSE can also be referred to as the average within-cluster distance and can be calculated for each cluster and then averaged for all the clusters.

Another commonly used evaluation measure is the Davies–Bouldin index (Davies & Bouldin, 1979). The Davies–Bouldin index is a measure of uniqueness of the clusters and takes into consideration both cohesiveness of the cluster (distance between the data points and center of the cluster) and separation between the clusters. It is the function of the ratio of within-cluster separation to the separation between the clusters. The lower the value of the Davies–Bouldin index, the better the clustering. However, both SSE and the Davies–Bouldin index have the limitation of not guaranteeing better clustering when they have lower scores.

7.1.2 How to Implement

k-Means clustering implementation in RapidMiner is simple and straightforward with one operator for modeling and one for unsupervised evaluation. In the modeling step, the parameter for the number of clusters, *k*, is specified as desired. The output model is a list of centroids for each cluster and a new attribute is attached to the original input dataset with the cluster ID. The cluster label is appended to the original dataset for each data point and can be visually evaluated after the clustering. A model evaluation step is required to calculate the average cluster distance and Davies–Bouldin index.

For this implementation, the Iris dataset will be used with four attributes and 150 data objects (Fisher, 1936). Even though a class label is not needed for clustering, it was kept for later explanation to see if identified clusters from an unlabeled dataset are similar to natural clusters of species in the dataset.

Step 1: Data Preparation

k-Means clustering accepts both numeric and polynominal data types; however, the distance measures are more effective with numeric data types. The number of attributes increases the dimension space for clustering. In this example the number of attributes has been limited to two by selecting petal width (a3) and petal length (a4) using the *Select attribute* operator as shown in Fig. 7.9. It is easy to visualize the mechanics of *k*-means algorithm by looking at two-dimensional plots for clustering. In practical implementations, clustering datasets will have more attributes.

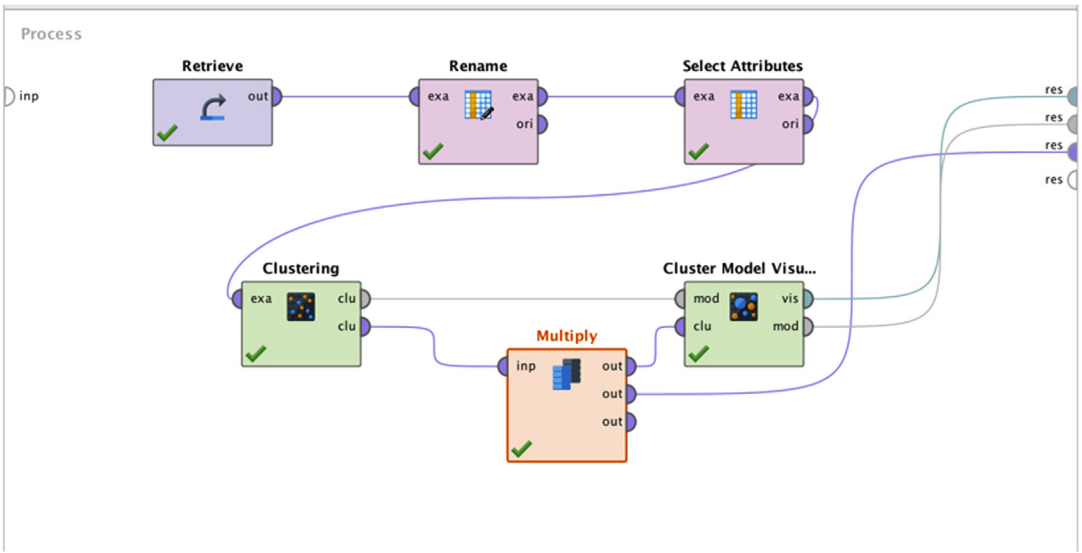


FIGURE 7.9
Data science process for *k*-means clustering.

Step 2: Clustering Operator and Parameters

The *k*-means modeling operator is available in the Modeling > Clustering and Segmentation folder of RapidMiner. These parameters can be configured in the model operator:

- *k*: *k* is the desired number of clusters.
- *Add cluster as attribute*: Append cluster labels (IDs) into the original dataset. Turning on this option is recommended for later analysis.
- *Max runs*: Since the effectiveness of *k*-means clustering is dependent on random initial centroids, multiple runs are required to select the clustering with the lowest SSE. The number of such runs can be specified here.
- *Measure type*: The proximity measure can be specified in this parameter. The default and most common measurement is Euclidean distance (L2). Other options here are Manhattan distance (L1), Jaccard coefficient, and cosine similarity for document data. Please refer to Chapter 4, Classification section *k*-NN for description on distance measures.
- *Max optimization steps*: This parameter specifies the number of iterations of assigning data objects to centroids and calculating new centroids.

The output of the modeling step includes the cluster model with *k* centroid data objects and the initial dataset appended with cluster labels. Cluster labels are named generically such as cluster_0, cluster_1,. . ., cluster_$k-1$.

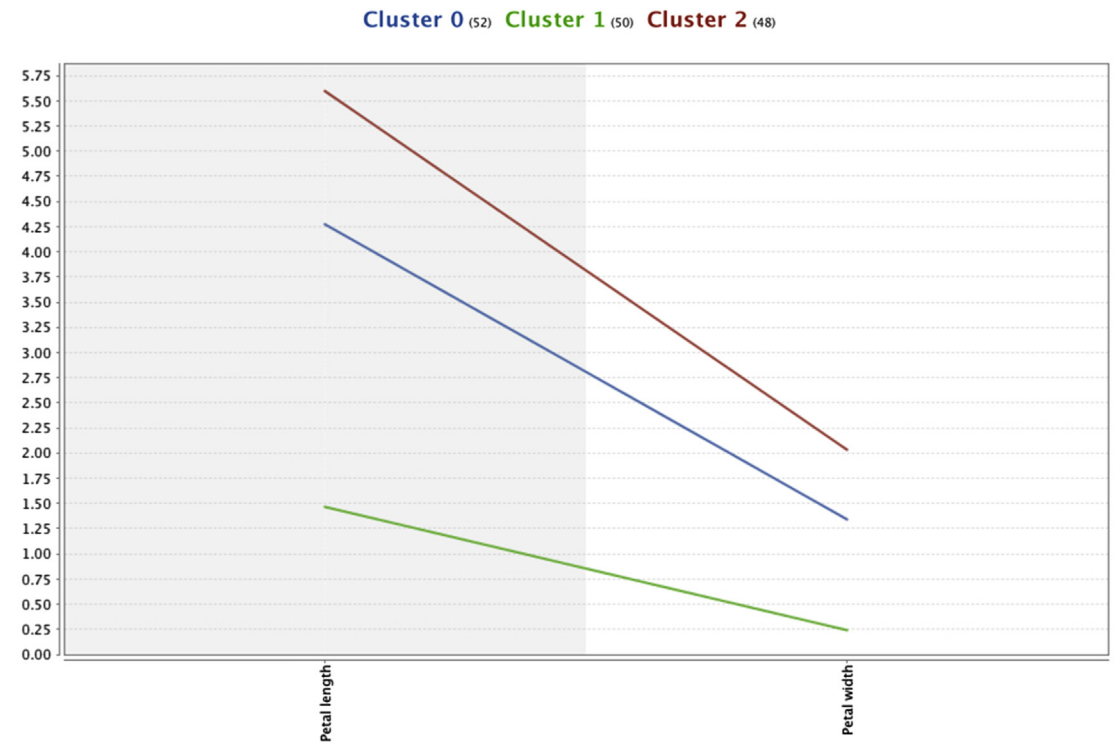
Step 3: Evaluation

Since the attributes used in the dataset are numeric, the effectiveness of clustering groups need to be evaluated using SSE and the Davies–Bouldin index. In RapidMiner, the *Cluster Model Visualizer* operator under Modeling > Segmentation is available for a performance evaluation of cluster groups and visualization. *Cluster Model Visualizer* operator needs both inputs from the modeling step: cluster centroid vector (model) and the labeled dataset. The two measurement outputs of the evaluation are average cluster distance and the Davies–Bouldin index.

Step 4: Execution and Interpretation

After the outputs from the *performance* operator have been connected to the result ports, the data science process can be executed. These outputs can be observed from results window:

- ***Cluster Model (Clustering):*** The model output contains the centroid for each of the *k*-clusters, along with their attribute values. As shown in Fig. 7.10, in the text view and folder view sections, all the data objects associated with the each cluster can be seen. The centroid plot view provides the parallel chart view (Chapter 3: Data Exploration) of the centroids. A large separation between centroids is desirable, because well-separated clusters divide the dataset cleanly.
- ***Labeled example set:*** The appended dataset has some of the most important information on clustering. The generic Iris dataset of 150 observations is clustered in three groups. The cluster value is appended as a new special polynominal attribute and takes a generic label format. In the scatterplot view of this output dataset, the *x*- and *y*-axes can be configured to be attributes of the original dataset, petal length and petal width. In the plot in Fig. 7.11, it can be noted how the algorithm identified clusters. This output can be compared against the original label (Iris species). and only five data points in the border of I. *versicolor* and I. *virginica* are mis-clustered! The *k*-means clustering process identified the different species in the dataset almost exactly.
- ***Visualizer and Performance vector:*** The output of the Cluster Model Visualizer shows the centroid charts, table, scatter plots, heatmaps, and performance evaluation metrics like average distance measured and the Davies–Bouldin index (Fig. 7.12). This step can be used to compare multiple clustering processes with different parameters. In advanced implementations, it is possible to determine the value of "*k*" using *Optimize Parameters* operator, based on the number of looped clustering runs with various values of *k*. Amongst multiple clustering runs each with a distinct value for *k*, the *k* value with the lowest average-within-centroid distance or Davies–Bouldin index is selected as the most optimal.



Cluster	Petal length	Petal width
Cluster 0	4.269	1.342
Cluster 1	1.464	0.244
Cluster 2	5.596	2.037

FIGURE 7.10
k-Means clustering centroids output.

The *k*-means clustering algorithm is simple, easy to implement, and easy to interpret. Although the algorithm can effectively handle an *n*-dimensional dataset, the operation will be expensive with a higher number of iterations and runs. One of the key limitations of *k*-means is that it relies on the user to assign the value of *k* (Berry & Linoff, 2000a,2000b). The number of clusters in the dataset will be unknown to begin with and an arbitrary number can limit the ability to find the right number of natural clusters in the dataset. There are a variety of methods to estimate the right number for *k*, ranging from the Bayesian Information Criterion to hierarchical methods that increase the value of *k* until the data points assigned to the cluster are Gaussian (Hamerly & Elkan, 2003). For a start, it is recommended to use a

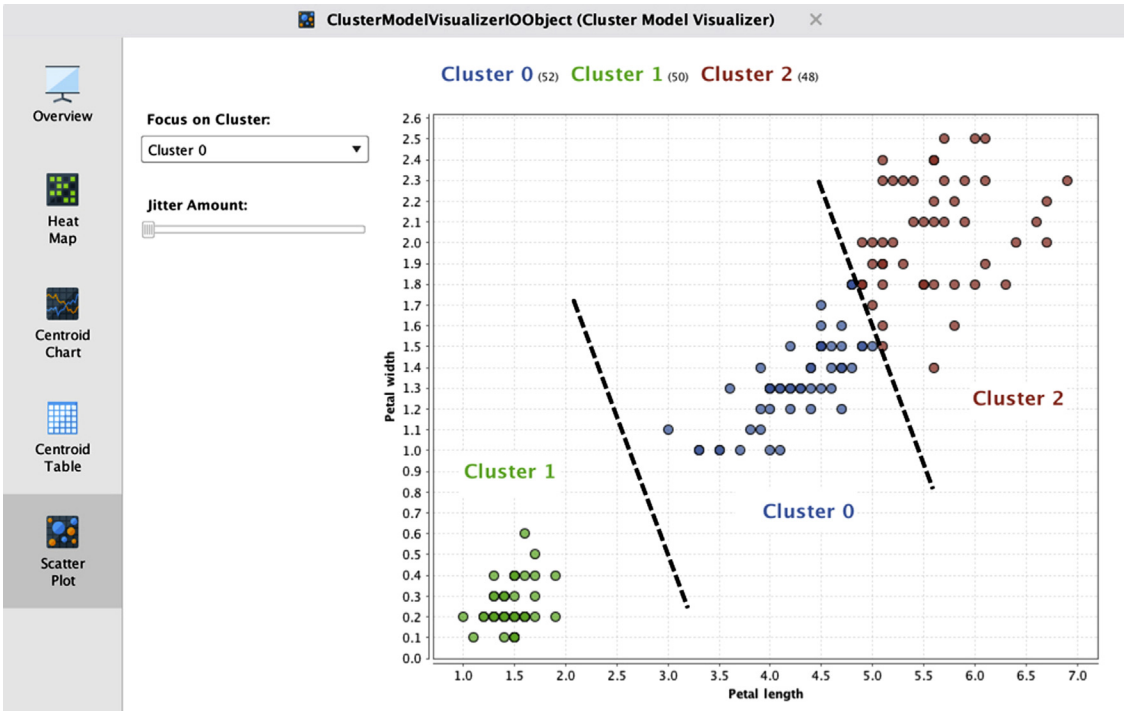


FIGURE 7.11
k-Means clustering visual output.

value of k in the low single digits and increasing it until it fits. Clustering using density methods will help provide an idea into the number of clusters and could be used as a value of k in k-means clustering.

Since the centroid prototype approach is used, k-means tends to find globular clusters in the dataset. However, natural clusters can be of all shapes and sizes. The presence of outliers possesses a challenge in the modeling of k-means clustering. The simplicity of the k-means clustering technique makes it a great choice for quick evaluation of globular clusters and as a preprocessing technique for data science modeling and for dimensionality reduction.

7.2 DBSCAN CLUSTERING

A cluster can also be defined as an area of high concentration (or density) of data objects surrounded by areas of low concentration (or density) of data objects. A density-clustering algorithm identifies clusters in the data based on the measurement of the density distribution in n-dimensional space. Unlike

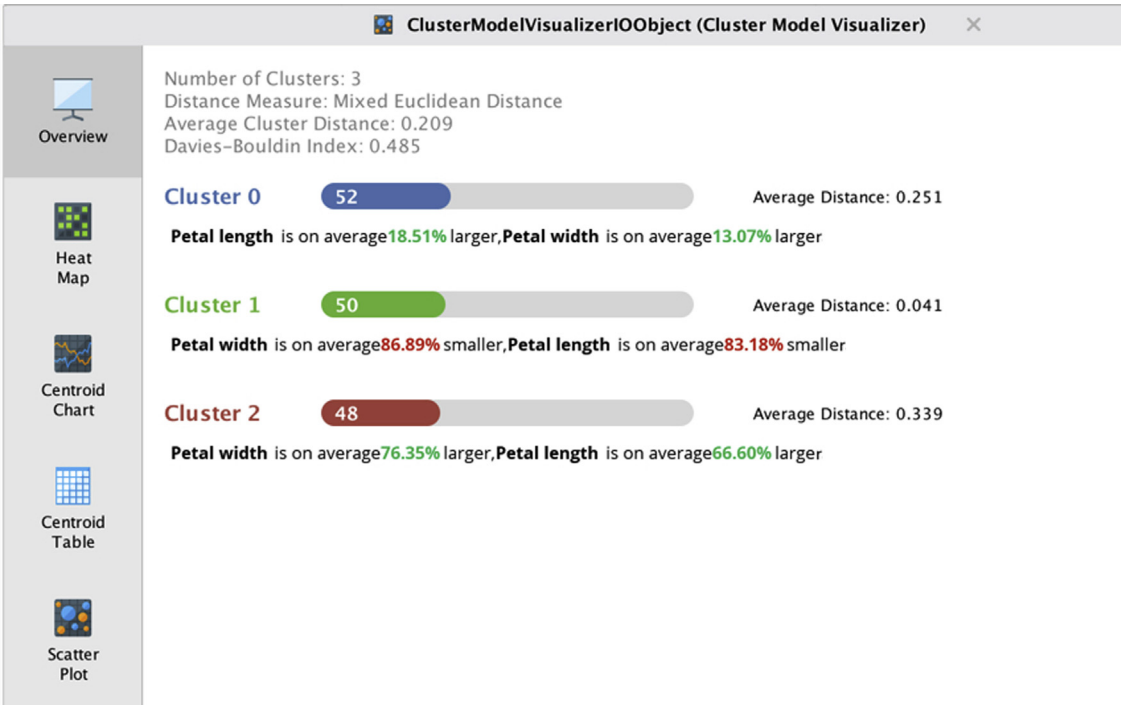


FIGURE 7.12
Performance measures of *k*-means clustering.

centroid methods, specifying the number of the cluster parameters (k) is not necessary for density-based algorithms. Thus, density-based clustering can serve as an important data exploration technique. DBSCAN is one of the most commonly used density-clustering algorithms (Ester, Kriegel, Sander, & Xu,1996). To understand how the algorithm works, the concept of density in a data space first needs to be defined.

Density can be defined as the number of data points in a unit n-dimensional space. The number of dimensions n is the number of attributes in a dataset. To simplify the visualization and to further understand how the model works, consider a two-dimensional space or a dataset with two numeric attributes. From looking at the dataset represented in Fig. 7.13, it can be visually concluded that the density in the top-left section is higher than the density in top-right, bottom-left, and bottom-right sections. Technically, density relates to the number of points in unit space, in this case a quadrant. Wherever there is high-density space amongst relatively low-density spaces, there is a cluster.

One can also measure density within a circular space around a point as in Fig. 7.14. The number of points within a circular space with radius ε

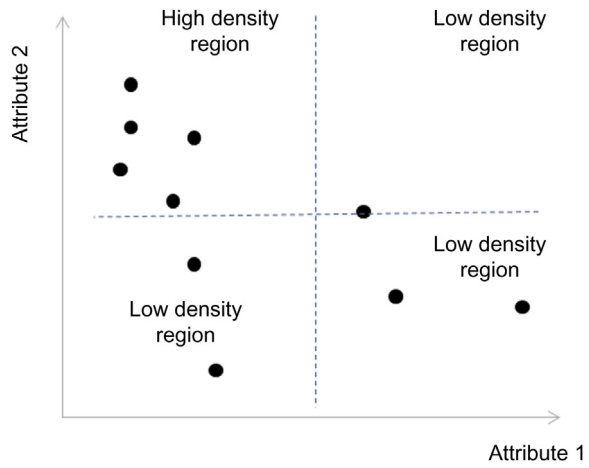


FIGURE 7.13
Dataset with two attributes.

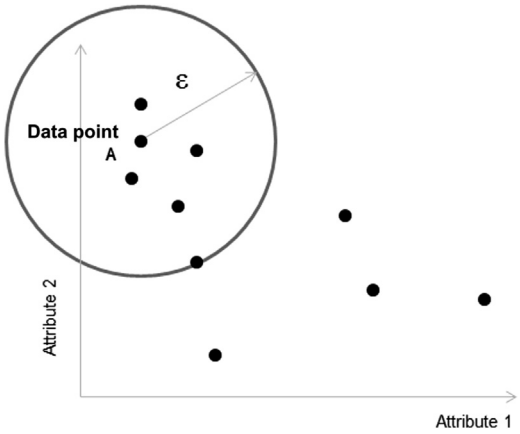


FIGURE 7.14
Density of a data point within radius ε.

(epsilon) around a data point A is six. This measure is called center-based density since the space considered is globular with the center being the point that is considered.

7.2.1 How It Works

The DBSCAN algorithm creates clusters by identifying high-density and low-density space within the dataset. Similar to *k*-means clustering, it is preferred

that the attributes are numeric because distance calculation is still used. The algorithm can be reduced to three steps: defining threshold density, classification of data points, and clustering (Tan et al., 2005).

Step 1: Defining Epsilon and MinPoints

The DBSCAN algorithm starts with calculation of a density for all data points in a dataset, with a given fixed radius ε (epsilon). To determine whether a neighborhood is high-density or low-density, a threshold of data points (MinPoints) will have to be defined, above which the neighborhood is considered high-density. In Fig. 7.14, the number of data points inside the space is defined by radius ε. If MinPoints is defined as 5, the space ε surrounding data point A is considered a high-density region. Both ε and MinPoints are user-defined parameters and can be altered for a dataset.

Step 2: Classification of Data Points

In a dataset, with a given ε and MinPoints, all data points can be defined into three buckets (Fig. 7.15):

- *Core points*: All the data points inside the high-density region of at least one data point are considered a core point. A high-density region is a space where there are at least *MinPoints* data points within a radius of ε for any data point.
- *Border points*: Border points sit on the circumference of radius ε from a data point. A border point is the boundary between high-density and

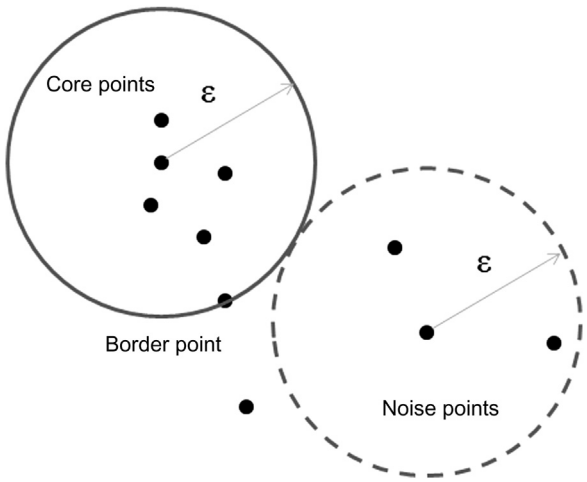


FIGURE 7.15
Core, border, and density points.

low-density space. Border points are counted within the high-density space calculation.

- *Noise points*: Any point that is neither a core point nor border point is called a noise point. They form a low-density region around the high-density region.

Step 3: Clustering

Once all data points in the dataset are classified into density points, clustering is a straightforward task. Groups of core points form distinct clusters. If two core points are within ε of each other, then both core points are within the same cluster. All these clustered core points form a cluster, which is surrounded by low-density noise points. All noise points form low-density regions around the high-density cluster, and noise points are not classified in any cluster. Since DBSCAN is a partial clustering algorithm, a few data points are left unlabeled or associated to a default noise cluster.

Optimizing Parameters

One of the key advantages of using a density algorithm is that there is no need for specifying the number of clusters (k). Clusters are automatically found in the dataset. However, there is an issue with selecting the distance parameter ε and a minimum threshold (MinPoints) to identify the dense region. One technique used to estimate optimal parameters for the DBSCAN clustering algorithm relates to the k-NN algorithm. The initial values of the parameter can be estimated by building a k-distribution graph. For a user-specified value of k (say, four data points), the distance of the k-th NN can be calculated for a data point. If the data point is a core point in a high-density region, then the distance of the k-th NN will be smaller. For a noise point, the distance will be larger. Similarly, the k-distance can be calculated for all data points in a dataset. A k-distance distribution graph can be built by arranging all the k-distance values of individual data points in descending order, as shown in Fig. 7.16. This arrangement is similar to Pareto charts. Points on the right-hand side of the chart will belong to data points inside a cluster, because the distance is smaller. In most datasets, the value of k-distance sharply rises after a particular value. The distance at which the chart rises will be the optimal value ε (epsilon) and the value of k can be used for MinPoints.

Special Cases: Varying Densities

The DBSCAN algorithm partitions data based on a certain threshold density. This approach creates an issue when a dataset contains areas of varying data density. The dataset in Fig. 7.17 has four distinct regions numbered from 1−4. Region 1 is the high-density area A, regions 2 and 4 are of medium-density B, and between them is region 3, which is extremely low-density C. If

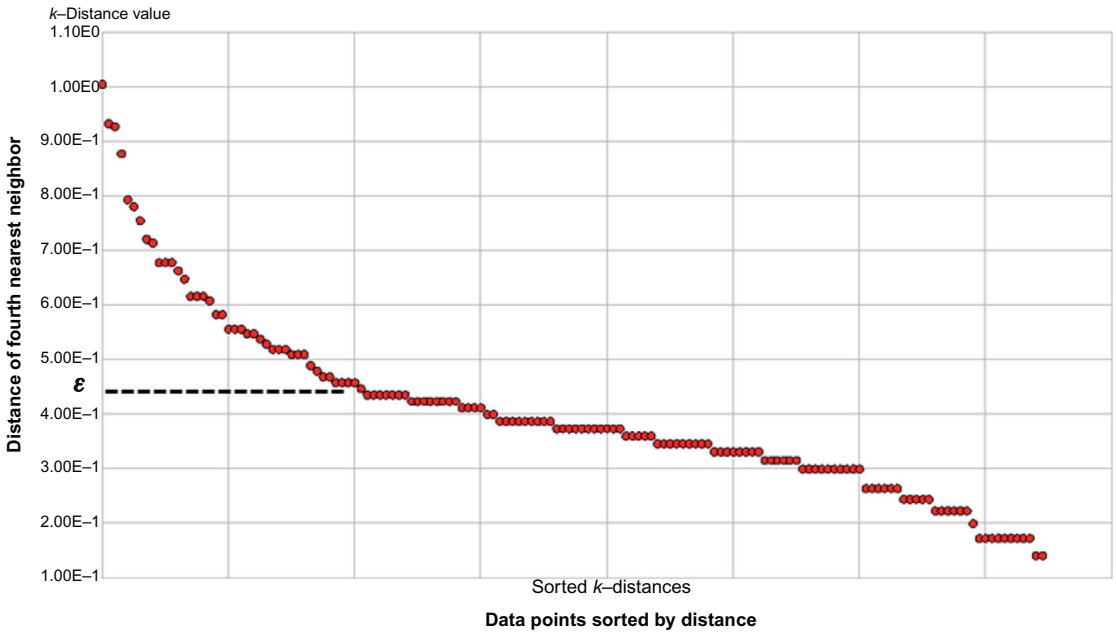


FIGURE 7.16
k-Distribution chart for the Iris dataset with $k = 4$.

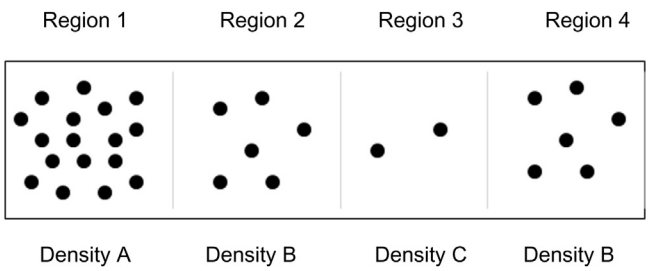


FIGURE 7.17
Dataset with varying densities.

the density threshold parameters are tuned in such a way as to partition and identify region 1, then regions 2 and 4 (with density B) will be considered noise, along with region 3. Even though region 4 with density B is next to an extremely low-density area and clearly identifiable visually, the DBSCAN algorithm will classify regions 2 through 4 as noise. The *k*-means clustering algorithm is better at partitioning datasets with varying densities.

7.2.2 How to Implement

The implementation of the DBSCAN algorithm is supported in RapidMiner through the DBSCAN modeling operator. The DBSCAN operator accepts numeric and polynominal datasets with provisions for user-specified ε (epsilon) and MinPoints parameters. Here are the implementation steps.

Step 1: Data Preparation

As with the *k*-means section, the number of attributes in the dataset will be limited to a3 and a4 (petal length and petal width) using the *Select Attribute* operator, so that the cluster can be visualized and the clustering process better understood.

Step 2: Clustering Operator and Parameters

The modeling operator is available in the Modeling > Clustering and Segmentation folder and is labeled DBSCAN. These parameters can be configured in the model operator:

- *Epsilon (ε)*: Size of the high-density neighborhood. The default value is 1.
- *MinPoints*: Minimum number of data objects within the epsilon neighborhood to qualify as a cluster.
- *Distance measure*: The proximity measure can be specified in this parameter. The default and most common measurement is Euclidean distance. Other options here are Manhattan distance, Jaccard coefficient, and cosine similarity for document data. Please refer to Chapter 4, Classification for a summary of different distance measures.
- *Add cluster as attributes*: To append cluster labels into the original dataset. This option is recommended for later analysis.

Step 3: Evaluation

Similar to *k*-means clustering implementation, the effectiveness of clustering groups can be evaluated using average within-cluster distance. In RapidMiner, the *Cluster Density Performance* operator under Evaluation > Clustering is available for performance evaluation of cluster groups generated by Density algorithms. The clustering model and labeled dataset are connected to *performance* operator for cluster evaluation. Additionally, to aid the calculation, *performance* operator expects Similarity Measure objects. A similarity measure vector is a distance measure of every example data object with the other data object. The similarity measure can be calculated by using *Data to Similarity* Operator on the example dataset.

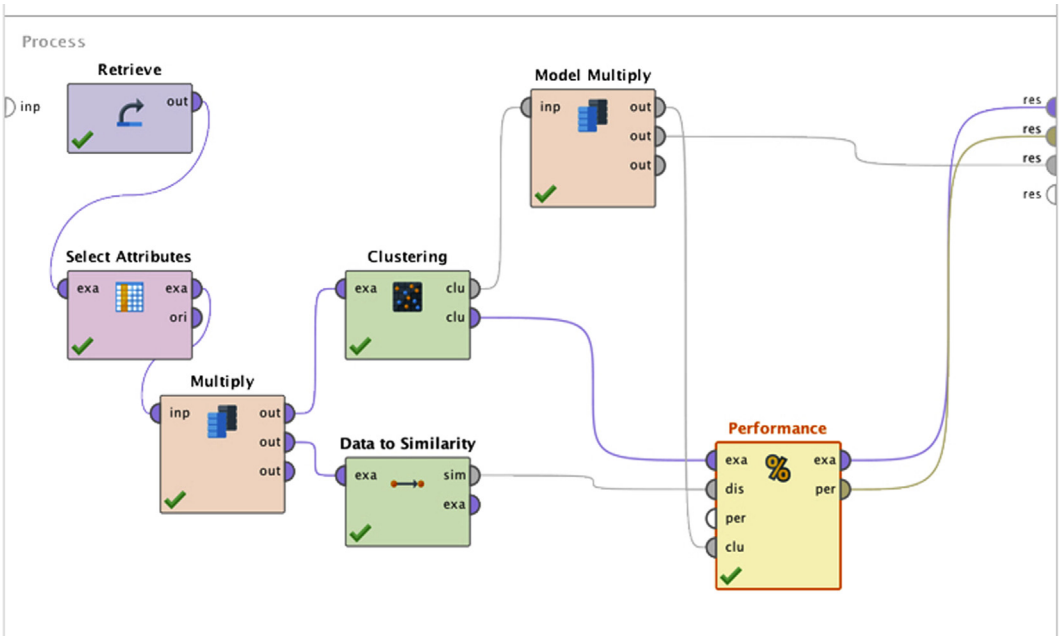


FIGURE 7.18
Data science process with density clustering.

Step 4: Execution and Interpretation

After the outputs from the *performance* operator have been connected to the result ports, as shown in Fig. 7.18, the model can be executed. The result output can be observed as:

1. *Model*: The cluster model output contains information on the number of clusters found in the dataset (Cluster 1, Cluster 2, etc.) and data objects identified as noise points (Cluster 0). If no noise points are found, then Cluster 0 is an empty cluster. As shown in Fig. 7.19, the Folder view and Graph view from the output window provide the visualization of data points classified under different clusters.
2. *Clustered example set*: The example set now has a clustering label that can be used for further analysis and visualization. In the scatterplot view of this dataset (Fig. 7.20), *x*- and *y*-axes can be configured to be attributes of the original dataset, petal length and petal width. The Color Column can be configured to be the new cluster label. In the plot, it can be noted how the algorithm found two clusters within the example set. The I. *setosa* species data objects have clear high-density areas but there is a density bridge between the I. *versicolor* and I. *virginica* species data points. There is no clear low-density area to

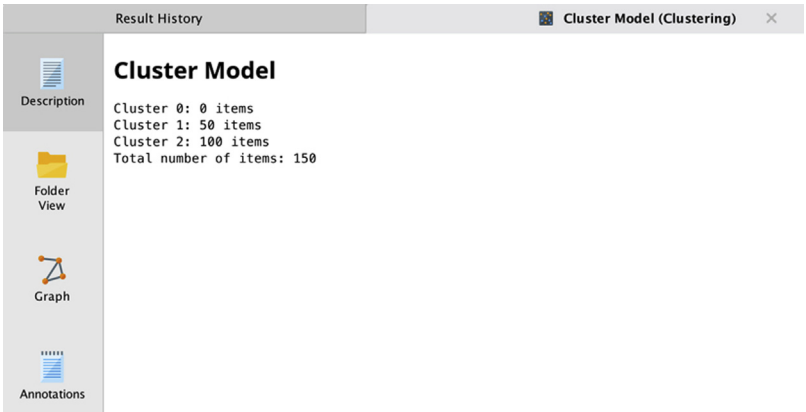


FIGURE 7.19
Density-clustering model output.

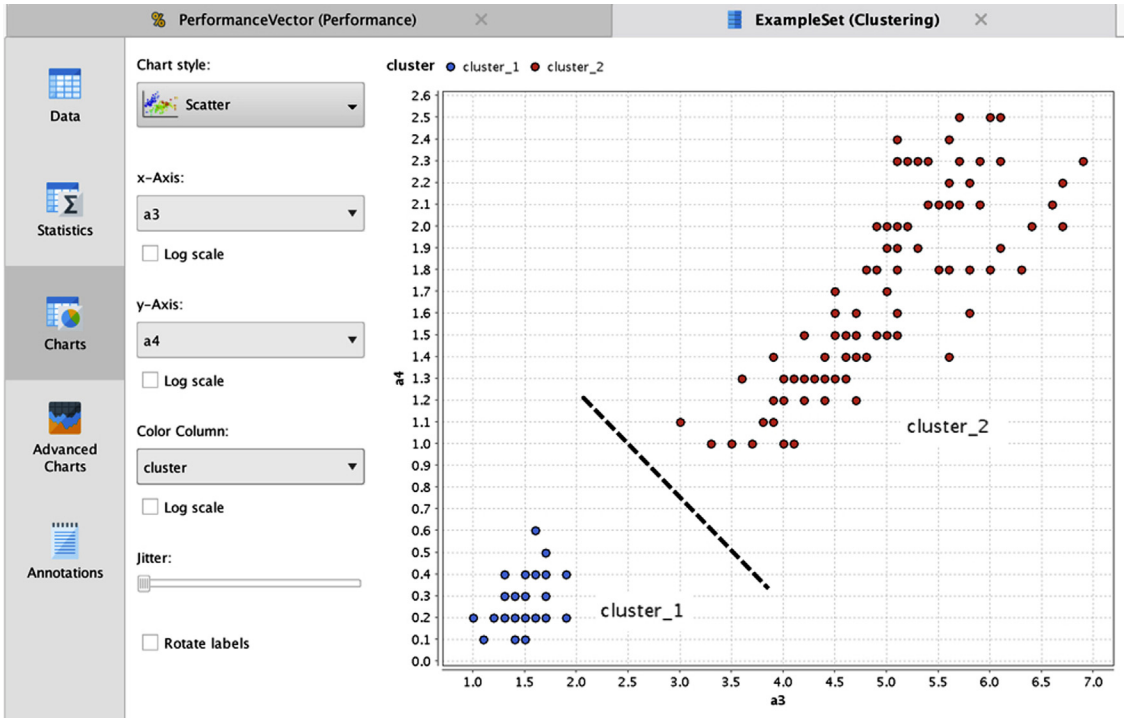


FIGURE 7.20
Density-clustering visual output.

partition these two species of data points. Hence, I. *versicolor* and I. *virginica* natural clusters are combined to one artificial predicted cluster. The epsilon and MinPoints parameters can be adjusted to find different results for the clustering.

3. *Performance vector*: The performance vector window shows the average distance within each cluster and the average of all clusters. The average distance is the distance between all the data points within the cluster divided by number of data points. These measures can be used to compare the performance of multiple model runs.

The main attraction of using DBSCAN clustering is that one does not have to specify the value of k, the number of clusters to be identified. In many practical applications, the number of clusters to be discovered will be unknown, like finding unique customers or electoral segments. DBSCAN uses variations in the density of the data distribution to find the concentration of structures in the data. These clusters can be of any shape and they are not confined to globular structures as in the k-means approach. But the density algorithms run into the risk of finding bridges between two natural clusters and merging them into one cluster.

Since the density-clustering technique yields partial clustering, DBSCAN ignores noise and outlier data points and they are not clustered in the final results. The inability to identify varying densities within a dataset is one of the major limitations of the DBSCAN clustering technique. Centroid methods are more successful at finding varying density patterns in a dataset. A dataset with a high number of attributes will have processing challenges with density-clustering methods. Given the complementary pros and cons of the k-means and DBSCAN methods, it is advisable to cluster the dataset by both methods and understand the patterns of both result sets.

7.3 SELF-ORGANIZING MAPS

A self-organizing map (SOM) is a powerful visual clustering technique that evolved from a combination of neural networks and prototype-based clustering. A SOM is a form of neural network where the output is an organized visual matrix, usually a two-dimensional grid with rows and columns. The objective of this neural network is to transfer all input data objects with n attributes (n dimensions) to the output lattice in such a way that objects next to each other are closely related to each other. Two examples of SOM layouts are provided in Fig. 7.21. This two-dimensional matrix becomes an exploration tool in identifying the clusters of objects related to each other by visual examination. A key distinction in this neural network is the absence of an output target function to optimize or predict, hence, it is an unsupervised learning algorithm. SOMs effectively arrange the data points in a lower

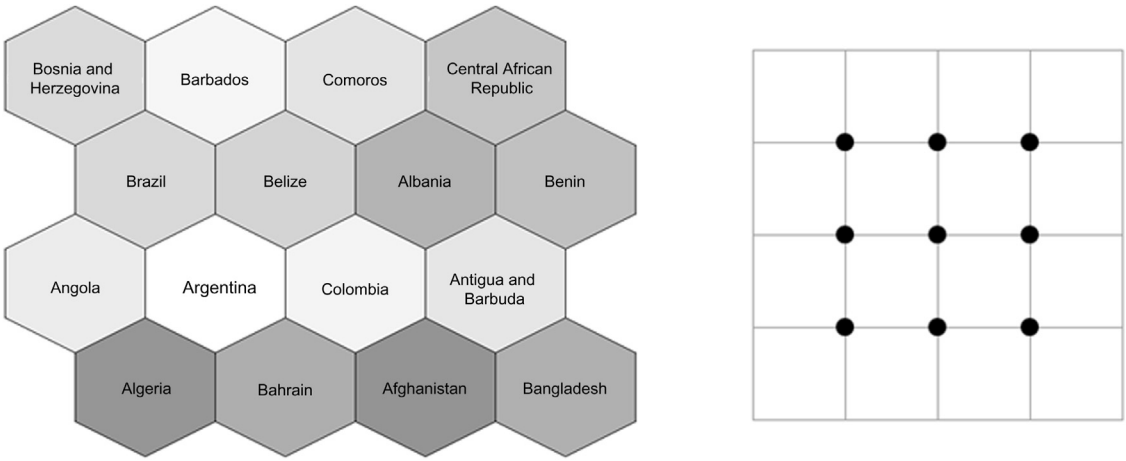


FIGURE 7.21
Self-organizing maps of countries by GDP data using a (A) hexagonal grid and (B) rectangular lattice. *GDP*, Gross domestic product.

dimensional space, thereby, aiding in the visualization of high-dimensional data through a low-dimensional space.

SOMs are relevant to clustering because the most common SOM output is a two-dimensional grid with data objects placed next to each other based on their similarity to one another. Objects related to each other are placed in close proximity. SOMs differ from other clustering techniques because there is no explicit clustering labels assigned to data objects. Data objects are arranged based on their attribute proximity and the task of clustering is left to visual analysis by the user. Hence, SOM is used as a *visual* clustering and data exploration technique (Germano, 1999).

SOMs were first proposed by Kohonen (1982) and, hence, this technique is also known as Kohonen networks; it is sometimes also referred to by a more specific name, self-organizing feature maps. SOM methodology is used to project data objects from *data space*, mostly in *n* dimensions, to *grid space*, usually resulting in two dimensions. Though other output formats are possible, the most common output formats for SOMs are: (1) a hexagonal lattice or a (2) rectangular grid as shown in Fig. 7.21. Each data point from the dataset occupies a cell or a node in the output lattice, with arrangement constraints depending on the similarity of data points. Each cell in the SOM grid, called a *neuron*, corresponds to one or a group of data points. In a hexagonal grid, each neuron has six neighbors while a rectangular lattice has four neighbors.

SOMs are commonly used in comparing data points with a large number of numeric attributes. The objective of this kind of analysis is to compare the

relative features of data objects in a simple two-dimensional setting where the placement of objects is related to each other. In Fig. 7.21A, the SOM compares relative gross domestic product (GDP) data from different countries where countries with similar GDP profiles are placed either in the same cells or next to each other. All similar countries around a particular cell can be considered a grouping. Although the individual data objects (countries) do not have a cluster membership, the placement of objects together aides in visual data analysis. This application is also called competitive SOMs.

7.3.1 How It Works

The algorithm for a SOM is similar to centroid-based clustering but with a neural network foundation. Since a SOM is essentially a neural network, the model accepts only numerical attributes. However, there is no target variable in SOM because it is an unsupervised learning model. The objective of the algorithm is to find a set of centroids (neurons) to represent the cluster but with topological constraints. The topology refers to an arrangement of centroids in the output grid. All the data objects from the dataset are assigned to each centroid. Centroids closer to each other in the grid are more closely "related" to each other than to centroids further away in the grid. A SOM converts numbers from the data space to a grid space with additional inter-topology relationships.

Step 1: Topology Specification

The first step for a SOM is specifying the topology of the output. Even though multi-dimensional output is possible, two-dimensional rows and columns with either a rectangular lattice or a hexagonal lattice are commonly used in SOMs to aid in the visual discovery of clustering. One advantage in using a hexagonal lattice is that each node or centroid can have six neighbors, two more than in a rectangular lattice. Hence, in a hexagonal lattice, the association of a data point with another data point can be more precise than for a rectangular grid. The number of centroids can be specified by providing the number of rows and columns in the grid. The number of centroids is the product of the number of rows and columns in the grid. Fig. 7.22 shows a hexagonal lattice SOM.

Step 2: Initialize Centroids

A SOM starts the process by initializing the centroids. The initial centroids are values of random data objects from the dataset. This is similar to initializing centroids in *k*-means clustering.

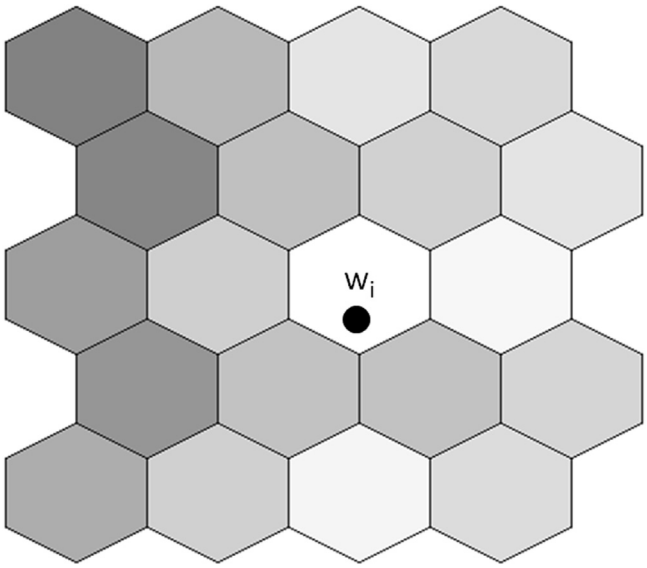


FIGURE 7.22
Weight of the centroid is updated.

Step 3: Assignment of Data Objects

After centroids are selected and placed on the grid in the intersection of rows and columns, data objects are selected one by one and assigned to the nearest centroid. The nearest centroid can be calculated using a distance function like Euclidean distance for numeric data or a cosine measure for document or binary data.

Step 4: Centroid Update

The centroid update is the most significant and distinct step in the SOM algorithm and is repeated for every data object. The centroid update has two related sub-steps.

The first sub-step is to update the closest centroid. The objective of the method is to update the data values of the nearest centroid of the data object, proportional to the difference between the centroid and the data object. This is similar to updating weights in the back-propagation algorithm of neural networks. In the neural network section of Chapter 4, Classification, how the weights of neurons are updated based on the error difference between the predicted and actual values was discussed. Similarly, in the context of a SOM, the values of centroids are updated. In fact, centroids are considered neurons in a SOM. Through this update, the closest centroid moves closer to the data object in the data space.

The centroid update step is repeated for a number of iterations, usually in the thousands. Denote t as the t^{th} iteration of the update where the data point $d(t)$ is picked up. Let w_1, w_2, w_3, …, w_k represent all the centroids in the grid space. Fig. 7.22 shows the lattice with centroid weight. Let r and c be the number of rows and columns in the grid. Then k will be equal to $r * c$. Let w_i be the nearest centroid for a data object $d(t)$. During iteration t, the nearest centroid w_i is updated using Eq. (7.4).

$$w_i(t+1) = w_i(t) + f_i(t) \times [d(t) - w_i(t)] \tag{7.4}$$

The effect of the update is determined by the difference between the centroid and the data point in the data space and the neighborhood function $f_i(t)$. The neighborhood function decreases for every iteration so that there are no drastic changes made in the final iteration. In addition to updating the nearest primary centroid, all other centroids in the grid space neighborhood of the primary centroid are updated as well. This will be reviewed in more detail in the next sub-step.

The second sub-step is to update all centroids in the grid space neighborhood as shown in Fig. 7.23. The neighborhood update step has to be proportional to the distance from the closest centroid to the centroid that is being updated. The update function has to be stronger when the distance is closer. Taking into account time decay and distance between neighborhood centroids, a Gaussian function is commonly used for:

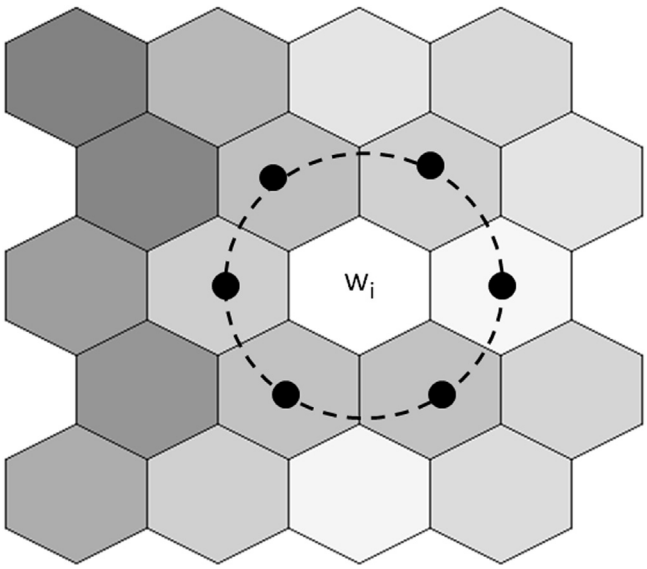


FIGURE 7.23
The weight of the neighborhood centroids are updated.

$$f_i(t) = \lambda_i(t) e^{\left(-\frac{(g_i - g_j)^2}{2\sigma^2}\right)} \tag{7.5}$$

where $\lambda_i(t)$ is the learning rate function that takes a value between 0 and 1 and decays for every iteration. Usually it is either a linear rate function or an inverse of the time function. The variable in the exponential parameter $g_i - g_j$ is the distance between the centroid being updated and the nearest centroid of the data point in the grid space. The variable σ determines the radius of the centroid update or the neighborhood effect. By updating the entire neighborhood of centroids in the grid, the SOM self-organizes the centroid lattice. Effectively, the SOM converts data from the data space to a location-constrained grid space.

Step 5: Termination

The entire algorithm is continued until no significant centroid updates take place in each run or until the specified number of run count is reached. The selection of the data object can be repeated if the dataset size is small. Like with many data science algorithms, a SOM tends to converge to a solution in most cases but doesn't guarantee an optimal solution. To tackle this problem, it is necessary to have multiple runs with various initiation measures and to compare the results.

Step 6: Mapping a New Data Object

A SOM model itself is a valuable visualization tool that can describe the relationship between data objects and be used for visual clustering. After the grids with the desired number of centroids have been built, any new data object can be quickly given a location on the grid space, based on its proximity to the centroids. The characteristics of new data objects can be further understood by studying the neighbors.

7.3.2 How to Implement

SOM can be implemented in a few different ways in RapidMiner, with varied functionality and resulting output.

- *Data exploration chart*: In Chapter 3, Data Exploration, the SOM chart was reviewed as one of the data exploration techniques. In RapidMiner, any dataset connected to a result port has a SOM chart feature under the Chart tab. This is a quick and easy method where the number of rows and columns can be specified, and a SOM chart can be rendered.
- *Data Transformation > Attribute set reduction > SOM Operator*: The *SOM* operator available under the Data Transformation folder is used to reduce the number of dimensions in the dataset. It is similar to the

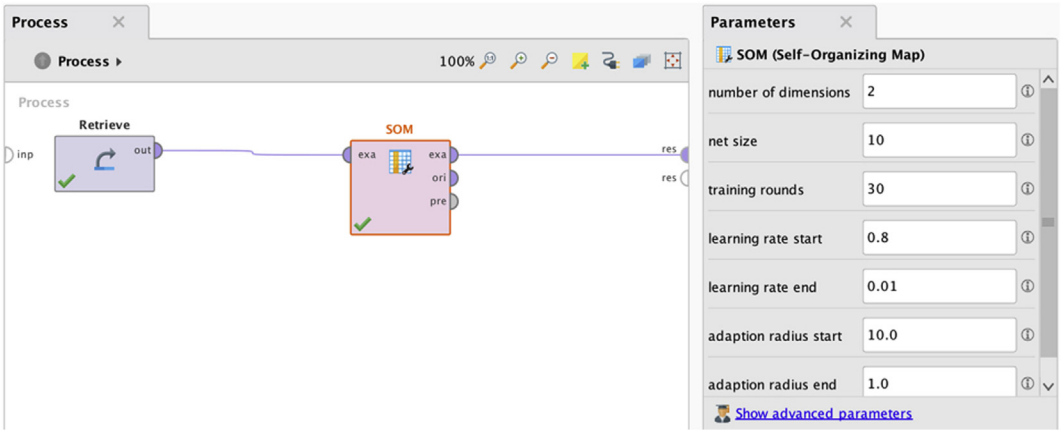


FIGURE 7.24
SOM in data transformation. *SOM*, Self-organizing map.

application of principal component analysis where the dataset is reduced to a lower dimensionality. In theory, a SOM can help reduce the data to any number of dimensions below the dimension count of the dataset. In this operator, the number of desired output dimensions can be specified in the parameters, as shown in Fig. 7.24. The net size parameter indicates the unique values in each of the SOM dimensions. There is no visual output for this operator and in two dimensions, only a square topography can be achieved through the *SOM data transformation* operator.

- *RapidMiner Extension > SOM Modeling Operator*: The *SOM modeling* operator is available in the SOM extension (Motl, 2012) and offers rich SOM visuals. SOM extensions will be used for the rest of this implementation section, so installing the SOM extension is recommended before proceeding further.

RapidMiner provides a marketplace platform called Extensions for specialized data science tasks which has operators, data science algorithms, data transformation operators, and visual exploration tools. The extensions can be installed and uninstalled easily from Help > Updates and Extensions. SOM is one of the extensions, along with text mining, the R extension, Recommenders, Weka extension, etc. Extensions will be used in the upcoming chapters.

The dataset used in this section is the relative GDP information by country (IMF, 2012) from the *World Economic Outlook Database October 2012* by the International Monetary Fund. The dataset has 186 records, one for each country, and four attributes in percentage of GDP: relative GDP invested, relative GDP saved, government revenue, and current account balance. Fig. 7.25 shows the actual raw data for a few rows and Fig. 7.26 shows the quartile plot for all the attributes.

Row No.	Country	Current account balance	General government revenue	Gross national savings	Total investment
1	Afghanistan	3.877	21.977	30.398	26.521
2	Albania	−11.372	25.835	14.509	25.886
3	Algeria	7.489	36.458	48.947	41.428
4	Angola	9.024	43.479	21.692	12.668
5	Antigua and...	−13.109	22.430	16.194	29.303
6	Argentina	0.658	37.199	22.595	24.451
7	Armenia	−14.653	20.970	16.660	31.313
8	Australia	−2.870	31.846	23.925	26.794
9	Austria	3.009	48.105	24.611	21.602
10	Azerbaijan	28.423	45.652	46.955	18.532

FIGURE 7.25
GDP by country dataset. *GDP*, Gross domestic product.

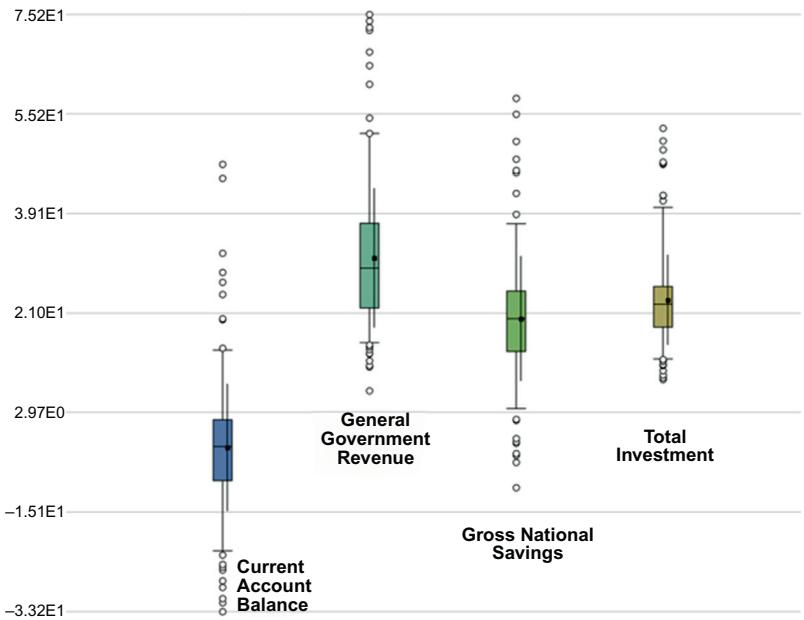


FIGURE 7.26
GDP by country: box-whisker (quartile) plot for all four attributes. *GDP*, Gross domestic product.

The objective of the clustering is to compare and contrast countries based on their percentage of GDP invested and saved, government revenue, and current account balance. Note that they are not compared using the size of the economy through absolute GDP but by the size of investment, national savings, current account, and the size of government relative to the country's GDP. The goal of this modeling exercise is to arrange countries in a grid so that countries with similar characteristics of investing, savings, size of government, and current accounts are placed next to each other. The four-dimensional information is being compressed into a two-dimensional map or grid. The dataset and RapidMiner process can be accessed from the companion site of the book at www.IntroDataScience.com.

Step 1: Data Preparation

As a neural network, a SOM cannot accept polynominal or categorical attributes because centroid updates and distance calculations work only with numeric values. Polynominal data can be either ignored with information loss or converted to a numeric attribute using the *Nominal to Numerical* type conversion operator available in RapidMiner. The country attribute is set as a label using Set role operator. In this dataset, there are records (each record is a country) where there is no data in few attributes. Neural networks cannot handle missing values and, hence, they need to be replaced by either zero or the minimum or average value for the attribute using the *Replace Missing Value* operator. In this example the average was chosen as the default missing value.

Step 2: SOM Modeling Operator and Parameters

The *SOM modeling extension* operator is available in the Self-Organizing Map folder, labeled with the same name. Please note that the SOM folder is visible only when the SOM extension has been installed. The coming parameters can be configured in the model operator. The modeling operator accepts the example set with numeric data and a label attribute if applicable. In this example set, the country name is a label attribute. Fig. 7.27 shows the RapidMiner process for developing a SOM model.

- *Training Rounds*: Defaults to 1000. This value indicates the number of training rounds for the data object selection process.
- *Net Size*: Indicates whether the grid size should be specified by the user or automatically approximated by the system. In this exercise select user input for *X* and *Y*.
- *Net Size X*: Specifies the number of columns in the grid (horizontal direction). This is the same as the possible values for the SOM_0 attribute in the output. In this example this value will be set to 10.

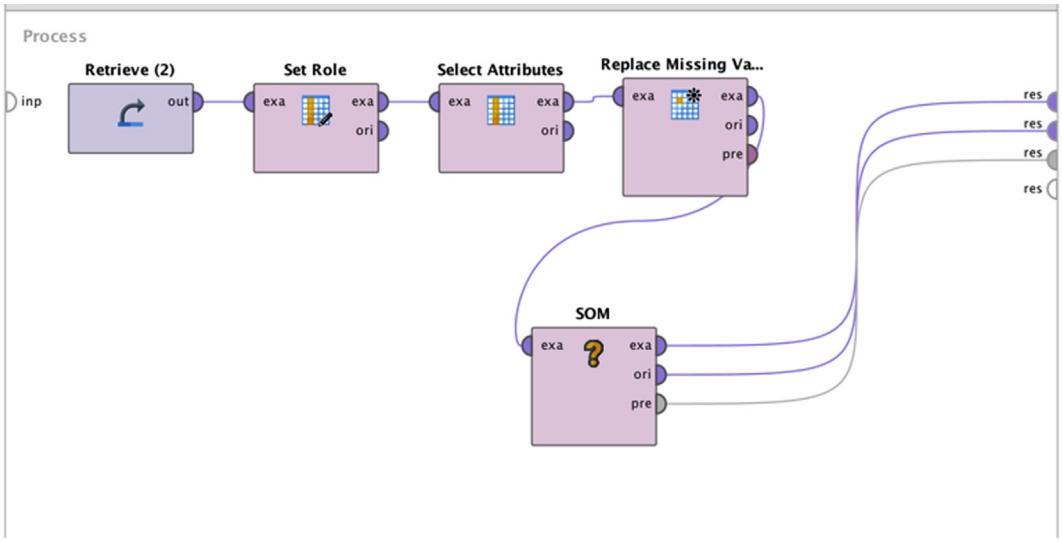


FIGURE 7.27
Clustering with SOM. *SOM*, Self-organizing map.

- *Net Size Y*: Specifies the number of rows in the grid (vertical direction). This also indicates the values for the SOM_1 attribute in the output grid. In this example this value will be set to 10.
- *Learning Rate*: The neural network learning parameter (λ), which takes a value from 0 to 1. The value of λ determines how sensitive the change in weight is to the previous weights. A value closer to 0 means the new weight would be mostly based on previous weight and an λ closer to 1 means that the weight would be mainly based on error correction. The initial λ will be assigned as 0.9 (see Section 4.5 on Neural Networks).
- *Learning Rate function*: The learning rate function in a neural network is a time decay function of the learning process. The default and most commonly used time decay function is the inverse of time.

Step 3: Execution and Interpretation

The RapidMiner process can be saved and executed. The output of the *SOM modeling* operator consists of a visual model and a grid dataset. The visual model is a lattice with centroids and mapped data points. The grid dataset output is the example set labeled with location coordinates for each record in the grid lattice.

Visual Model

The visual model of the SOM displays the most recognizable form of SOM in a hexagonal grid format. The size of the grid is configured by the input parameters that set the net size of *X* and *Y*. There are several advanced visualization styles available in the Visualization results window. The SOM visual output can be customized using these parameters:

- *Visualization Style*: This selection controls the visual layout and background color of the SOM hexagons. The value of the selected measure is represented as a background gradient color. The default, U-Matrix, presents a background gradient color proportional to the distance of the central data points in adjacent hexagons. The P-Matrix option shows the number of example data points through the background gradient color. The selection of an individual attribute name for the visualization style renders the background gradient proportional to the value of the selected attribute. The visualization style selection does not rearrange the data points assigned to hexagons.
- *Label*: Selection shows the attribute value selected in the hexagons.
- *Color Schema*: Selection of monochrome or color scheme.

Fig. 7.28 shows a SOM with the default selection of the label as Country and the visualization style as U-Matrix. It can be observed how countries are

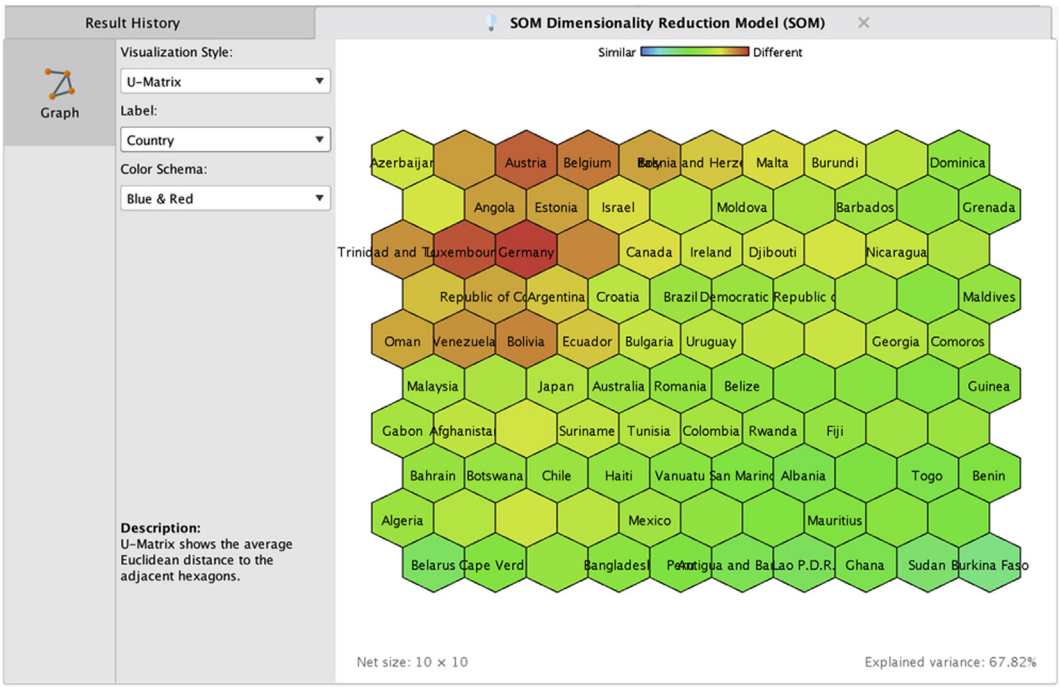


FIGURE 7.28
SOM output in the hexagonal grid. *SOM*, Self-organizing map.

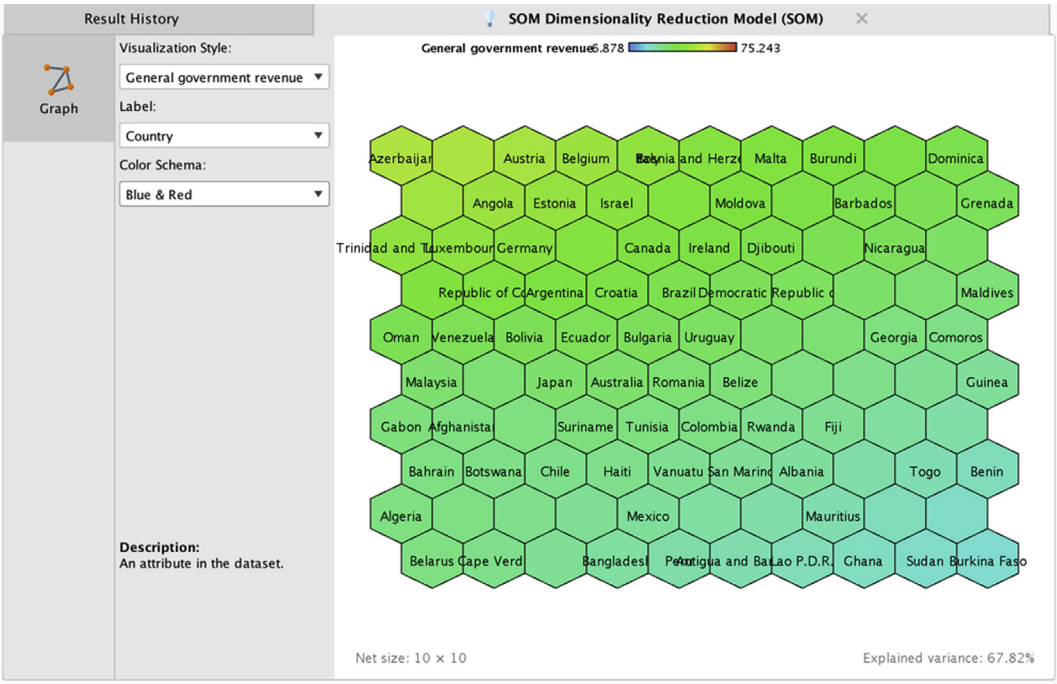


FIGURE 7.29
SOM output with color overlay related to government revenue. *SOM*, Self-organizing map.

placed on the grid based on their relationship with each other, as evaluated by the four economic metrics in relation to GDP. Countries with similar characteristics are placed closer to each other than to others in the grid. If more than one country belongs to a centroid (hexagon), then the label of one country closer to the centroid is displayed on the grid. The grid locations of all the counties are listed in the location coordinates section of the results window.

A few interesting patterns in the data can be observed by changing the visualization style as a metric in the dataset. In Fig. 7.29, government revenue as percentage of GDP is used and visually, countries with high government revenue as a percentage of GDP is displayed on the top-left side of the grid (e.g., Belgium with 48%) and countries with low government revenue are at bottom of the grid (Bangladesh 11%).

Fig. 7.30 shows the national savings rate visualization in the SOM; countries with a high savings rate (Algeria 49%) are concentrated on the left side and countries with a low savings rate (Maldives 2%) are concentrated on the right side of the SOM.

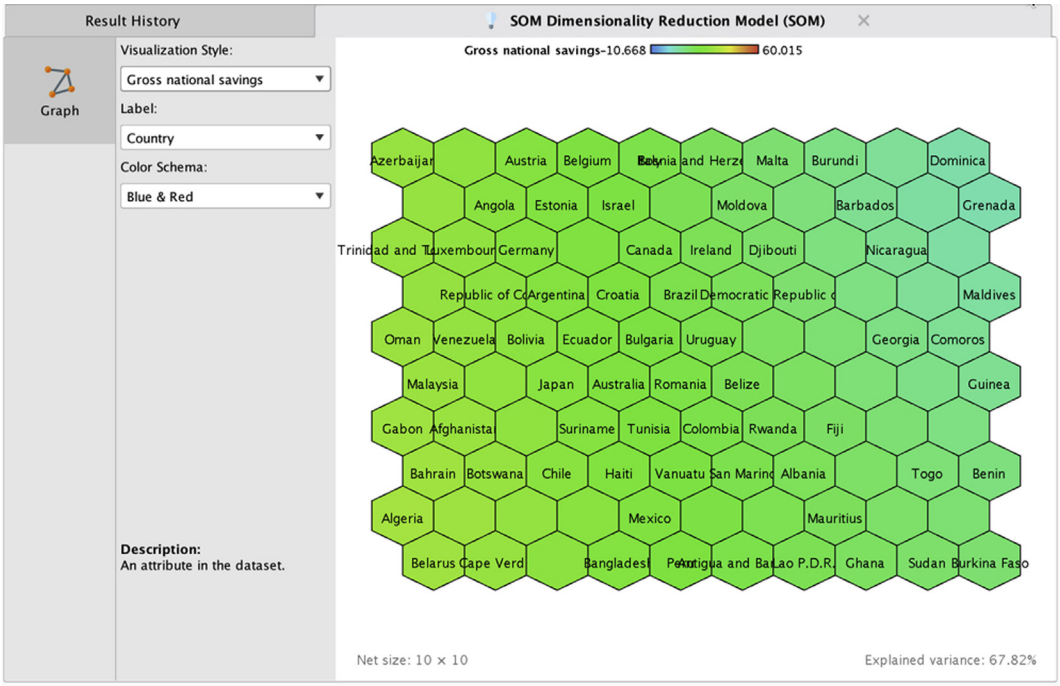


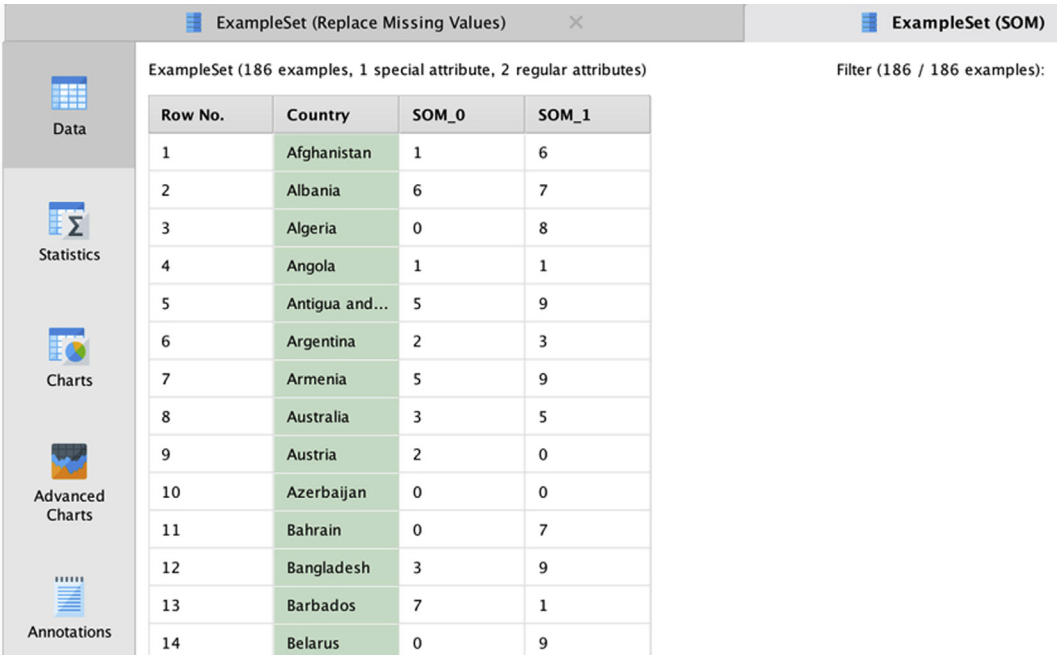
FIGURE 7.30
SOM output with color overlay related to national savings rate. *SOM*, Self-organizing map.

Location Coordinates

The second output of the *SOM* operator contains the location coordinates of the *x*- and *y*-axes of grid with labels SOM_0 and SOM_1. The coordinate values of location range from 0 to net size—1, as specified in the model parameters, since all data objects, in this case countries, are assigned to a specific location in the grid. This output, as shown in Fig. 7.31, can be further used for post-processing such as distance calculation of locations between countries in the grid space.

Conclusion

The methodology of SOMs is derived from the foundations of both neural network and prototype-clustering approaches. SOMs are an effective visual clustering tool to understand high-dimensional numeric data. They reduce the features of the dataset to two or three features, which is used to specify the topology of the layout. Hence, SOMs are predominantly used as a visual discovery and data exploration technique. Some of the applications of SOMs include the methods that are used in conjunction with other data science techniques such as graph mining (Resta, 2012), text mining (Liu, Liu, & Wang, 2012), speech recognition (Kohonen, 1988), etc.



Row No.	Country	SOM_0	SOM_1
1	Afghanistan	1	6
2	Albania	6	7
3	Algeria	0	8
4	Angola	1	1
5	Antigua and...	5	9
6	Argentina	2	3
7	Armenia	5	9
8	Australia	3	5
9	Austria	2	0
10	Azerbaijan	0	0
11	Bahrain	0	7
12	Bangladesh	3	9
13	Barbados	7	1
14	Belarus	0	9

FIGURE 7.31
SOM output with location coordinates. *SOM*, Self-organizing map.

References

Bache, K., & Lichman, M. (2013). *UCI machine learning repository*. University of California, School of Information and Computer Science. Retrieved from <http://archive.ics.uci.edu/ml>.

Berry, M. J., & Linoff, G. (2000a). Converging on the customer: Understanding the customer behavior in the telecommunications industry. In M. J. Berry, & G. Linoff (Eds.), *Mastering data science: The art and science of customer relationship management* (pp. 357–394). John Wiley & Sons, Inc.

Berry, M. J., & Linoff, G. (2000b). Data science techniques and algorithms. In M. J. Berry, & G. Linoff (Eds.), *Mastering data science: The art and science of customer relationship management* (pp. 103–107). John Wiley & Sons, Inc.

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 1*(2), 224–227.

Ester, M., Kriegel, H. -P., Sander, J., & Xu X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *AAAI Press proceedings of 2nd international conference on knowledge discovery and data science KDD-96* (Vol. 96, pp. 226–231). AAAI Press.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Human Genetics, 7*, 179–188. Retrieved from <https://doi.org/10.1111/j.1469-1809.1936.tb02137.x>.

Germano, T. (March 23, 1999) Self-organizing maps. Retrieved from <http://davis.wpi.edu/~matt/courses/soms/> Accessed 10.12.13.

CHAPTER 9

Text Mining

This chapter explores and formalizes how to extract patterns and discover new knowledge by applying many of the techniques learned so far, not on ordered data, but on unstructured natural language. This constitutes the vast area of text and web mining. For all the techniques described up to this point, a cleaned and organized table consisting of rows and columns of data was fed as input to an algorithm. The output from the algorithm was a model that could then be used to predict outcomes from a new data set or to find patterns in data. But are the same techniques applicable to extract patterns and predict outcomes when the input data looks like normal written communication? This might seem baffling at first, but as shall be seen in this chapter, there are ways of presenting text data to the same algorithms that process normal data.

To start out a brief historical introduction to the field of text mining is given to establish some context. In the following section, techniques that can convert common text into a semi-structured format will be described that we, and the algorithms introduced so far, can recognize. Finally, the text mining concepts will be implemented using two case studies: one involving an unsupervised (clustering) model and another involving a supervised support vector machine (SVM) model. The chapter will be closed with some key considerations to keep in mind while implementing text mining.

Unstructured data (including text, audio, images, videos, etc.) is the new frontier of data science. Eric Siegel in his book *Predictive Analytics* (Siegel, 2013) provides an interesting analogy: if all the data in the world was equivalent to the water on earth, then textual data is like the ocean, making up a majority of the volume. Text analytics is driven by the need to process natural human language, but unlike numeric or categorical data, natural language does not exist in a structured format consisting of rows (of examples) and columns (of attributes). Text mining is, therefore, the domain of unstructured data science.

Data Science. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00009-5

Some of the first applications of text mining came about when people were trying to organize documents (Cutting, 1992). Hearst (1999) recognized that text analysis does not require artificial intelligence but "... a mixture of computationally-driven and user-guided analysis," which is at the heart of the supervised models used in predictive analytics that have been discussed so far.

IT IS NLP, MY DEAR WATSON!

Perhaps the most famous application of text mining is IBM's Watson program, which performed spectacularly when competing against humans on the nightly game show Jeopardy! How does Watson use text mining? Watson has instant access to hundreds of millions of structured and unstructured documents, including the full content of Wikipedia entries.

When a Jeopardy! question is transcribed to Watson, it searches for and identifies candidate documents that score a close match to the words of the question. The search and comparison methods it uses are similar to those used by search engines, and include many of the techniques, such as *n*-grams and stemming, which will be discussed in this chapter. Once it identifies candidate documents, it again uses other text mining (also known as natural language processing or NLP) methods to rank them. For example, if the answer is, *regarding this device, Archimedes said, "give me a place to stand on, and I will move the earth,"* a Watson search for this sentence in its databases might reveal among its candidate documents several with the term "lever." Watson might insert the word "lever" inside the answer text and rerun a new search to see if there are other documents with the new combination of terms. If the search result has many matches to the terms in the sentence—as it most likely would in this case—a high score is assigned to the inserted term.

If a broad and non-domain-focused program like Watson, which relies heavily on text mining and NLP, can answer open-ended quiz show questions with nearly 100% accuracy, one can imagine how successful specialized NLP tools would be. In fact IBM has successfully deployed a Watson-type program to help in decision making at health care centers (Upbin, 2013).

Text mining also finds applications in numerous business activities such as email spam filtering, consumer sentiment analysis, and patent mining to name a few. A couple of these will be explored in this chapter.

People in the data management domains can appreciate text mining in a slightly different context. Here, the objective is not so much discovering new trends or patterns, but cleaning data stored in business databases. For example, when people make manual entries into a customer relationship management software, there is a lot of scope for typographic errors: a salesperson's name may be spelled "Osterman" in several instances (which is perhaps the correct spelling) and "Ostrerman" in a few instances, which is a misspelling. Text mining could be used in such situations to identify the right spelling and suggest it to the entry operator to ensure that data consistency is maintained. Similar application logic could be used in identifying and streamlining call center service data (McKnight, 2005).

Text mining, more than any other technique within data mining, fits the *mining* metaphor. Traditionally, mining refers to the process of separating dirt from valuable metal and in the case of text mining, it is an attempt to separate valuable keywords from a mass of other words (or relevant documents

from a sea of documents) and use them to identify meaningful patterns or make predictions.

9.1 HOW IT WORKS

The fundamental step in text mining involves converting text into semi-structured data. Once the unstructured text is converted into semi-structured data, there is nothing to stop one from applying any of the analytics techniques to classify, cluster, and predict. The unstructured text needs to be converted into a semi-structured data set so that patterns can be found and even better, models could be trained to detect patterns in new and unseen text. The chart in Fig. 9.1 identifies the main steps in this process at a high level.

Each of the main processes will now be examined in detail and some necessary terminology and concepts will be introduced. But before these processes are described, a few core ideas essential to text analytics will need to be defined.

9.1.1 Term Frequency–Inverse Document Frequency

Consider a web search problem where the user types in some keywords and the search engine extracts all the documents (essentially, web pages) that contain these keywords. How does the search engine know which web pages to serve up? In addition to using network rank or page rank, the search engine also runs some form of text mining to identify the most relevant web pages. Suppose for example, that the user types in the following keywords: "RapidMiner books that describe text mining." In this case, the search engines run on the following basic logic:

1. Give a high weightage to those keywords that are relatively rare.
2. Give a high weightage to those web pages that contain a large number of instances of the rare keywords.

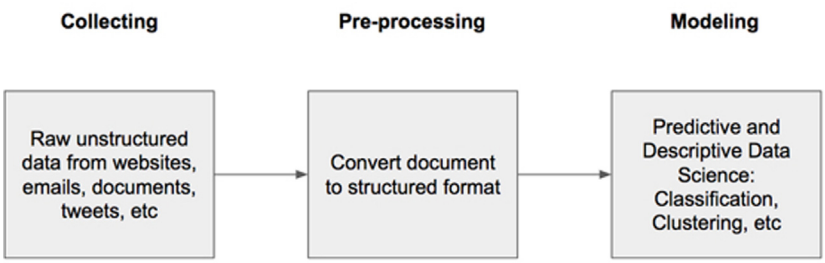


FIGURE 9.1
A high-level process for text mining.

In this context, what is a rare keyword? Clearly, English words like "that," "books," "describe," and "text" possibly appear in a large number of web pages, whereas "RapidMiner" and "mining" may appear in a relatively smaller number of web pages. (A quick web search returned 7.7 billion results for the word "books," whereas only 584,000 results were returned for "RapidMiner" at the time of this writing.) Therefore, these rarer keywords would receive a higher rating to begin with according to logic 1. Next, among all those pages that contain the rare keywords, only those pages that contain the largest number of instances of the rare keywords are likely to be the most relevant for the user and will receive high weightage according to logic 2. Thus, the highest-weighted web pages are the ones for which the *product* of these two weights is the highest. Therefore, only those pages that not only contain the rare keywords, but also have a high number of instances of the rare keywords should appear at the top of the search results.

The technique of calculating this weighting is called term *TF–IDF*, which stands for term frequency–inverse document frequency.

Calculating TF is extremely easy: it is simply the ratio of the number of times a keyword appears in a given document, n_k (where k is the keyword), to the total number of terms in the document, n:

$$\mathrm{TF} = \frac{n_k}{n} \tag{9.1}$$

Considering the mentioned example, a common English word such as "that" will have a fairly high TF score and a word such as "RapidMiner" will have a much lower TF score.

IDF is defined as follows:

$$\mathrm{IDF} = \log_2\left(\frac{N}{N_k}\right) \tag{9.2}$$

where N is the number of documents under consideration (in a search engine context, N is the number of all the indexed web pages). For most text mining problems, N is the number of documents that one is trying to mine, and N_k is the number of documents that contain the keyword, k. Again, a word such as "that" would arguably appear in every document and, thus, the ratio (N/N_k) would be close to 1, and the IDF score would be close to zero for. However, a word like "RapidMiner" would possibly appear in a relatively fewer number of documents and so the ratio (N/N_k) would be much greater than 1. Thus, the IDF score would be high for this less common keyword.

Finally, TF–IDF is expressed as the simple product as shown:

$$\mathrm{TF-IDF} = \frac{n_k}{n} \times \log_2\left(\frac{N}{N_k}\right) \tag{9.3}$$

Going back to the mentioned example, when the high TF for "that" is multiplied by its corresponding low IDF, a low (or zero) TF−IDF will be reached, whereas when the low TF for "RapidMiner" is multiplied by its corresponding fairly high IDF, a relatively higher TF−IDF would be obtained.

Typically, TF−IDF scores for every word in the set of documents is calculated in the preprocessing step of the three-step process described earlier. Performing this calculation will help in applying any of the standard data science techniques that have been discussed so far in this book. In the following sections additional concepts that are commonly employed in text mining will be described.

9.1.2 Terminology

Consider the following two sentences: "This is a book on data mining" and "This book describes data mining and text mining using RapidMiner." Suppose the objective is to perform a comparison between them, or a similarity mapping. For this purpose, each sentence is one unit of text that needs to be analyzed.

These two sentences could be embedded in an email message, in two separate web pages, in two different text files, or else they could be two sentences in the same text file. In the text mining context, each sentence is considered a distinct *document*. Furthermore, in the simplest case, words are separated by a special character: a blank space. Each word is called a *token*, and the process of discretizing words within a document is called *tokenization*. For the purpose here, each sentence can be considered a separate document, although what is considered an individual document may depend upon the context. For now, a document here is simply a sequential collection of tokens.

Document 1	This is a book on data mining
Document 2	This book describes data mining and text mining using RapidMiner

Some form of structure can be imposed on this raw data by creating a matrix where the columns consist of all the tokens found in the two documents and the cells of the matrix are the counts of the number of times a token appears, as shown in Table 9.1.

Each token is now an attribute in standard data science parlance and each document is an example. One, therefore, has a structured *example set*, to use standard terminology. Basically, unstructured raw data is now transformed into a format that is recognized, not only by the human users as a data table, but more importantly by all the machine learning algorithms which require

Table 9.1 Building a Matrix of Terms From Unstructured Raw Text

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1	1	1	1	1	1	1	0	0	0	0	0
Document 2	1	0	0	1	0	1	2	1	1	1	1	1

such tables for training. This table is called a *document vector* or *term document matrix (TDM)* and is the cornerstone of the preprocessing required for text mining. Suppose a third statement is added, "RapidMiner is offered as an open source software program." This new document will increase the number of rows of the matrix by one (Document 3); however, it increases the number of columns by seven (seven new words or tokens were introduced). This results in zeroes being recorded in nine other columns for row 3. As more new statements are added that have little in common, one would end up with a very sparse matrix.

Note that one could have also chosen to use the term frequencies for each token instead of simply counting the number of occurrences and it would still be a sparse matrix. TF can be obtained by dividing each row of Table 9.1 by number of words in the row (document). This is shown in Table 9.2.[1]

Similarly, one could have also chosen to use the TF−IDF scores for each term to create the document vector. This is also shown in Fig. 9.2.

One thing to note in the two sample text documents was the occurrence of common words such as "a," "this," "and," and other similar terms. Clearly in larger documents a higher number of such terms would be expected that do not really convey specific meaning. Most grammatical necessities such as articles, conjunctions, prepositions, and pronouns may need to be filtered before additional analysis is performed. Such terms are called *stop words* and usually include most articles, conjunctions, pronouns, and prepositions. *Stop word filtering* is usually the second step that follows immediately after *tokenization*. Notice that the document vector has a significantly reduced size after applying standard English stop word filtering (see Fig. 9.3).

In addition to filtering standard stop words, some specific terms might also need to be filtered out. For example, in analyzing text documents that pertain to the automotive industry, one may want to filter away terms that are common to this industry such as "car," "automobile," "vehicle," and so on. This

[1] RapidMiner does a double normalization while calculating TF scores. For example, in case of Document 1, the TF score for the term "data" would be $(0.1498)/\sqrt{(0.1498^2 + 0.1498^2 + \ldots + 0.1498^2)} = 0.1498/\sqrt{7*(0.1498^2)} = 0.3779$. Similarly for all other terms, double normalization makes it easier to apply algorithms such as SVM. This change in TF calculation is reflected in the TF-IDF scores.

Table 9.2 Using Term Frequencies Instead of Term Counts in a TDM

	This	is	a	book	on	data	mining	describes	text	rapidminer	and	using
Document 1	1/7 = 0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0.1428	0	0	0	0	0
Document 2	1/10 = 0.1	0	0	0.1	0	0.1	0.2	0.1	0.1	0.1	0.1	0.1

TDM, *Term document matrix.*

ExampleSet (2 examples, 0 special attributes, 12 regular attributes)

Row No.	RapidMiner	This	a	and	book	data	describes	is	mining	on	text	using
1	0	0	0.577	0	0	0	0	0.577	0	0.577	0	0
2	0.447	0	0	0.447	0	0	0.447	0	0	0	0.447	0.447

FIGURE 9.2

Calculating TF—IDF scores for the sample TDM. *TF—IDF*, Term Frequency—Inverse Document Frequency; *TDM*, term document matrix.

Row No.	RapidMiner	book	data	describes	mining	text	using
1	0	1	1	0	1	0	0
2	1	1	1	1	2	1	1

FIGURE 9.3

Stop word filtering reduces the size of the TDM significantly.

is generally achieved by creating a separate dictionary where these context-specific terms can be defined and then *term filtering* can be applied to remove them from the data. (*Lexical substitution* is the process of finding an alternative for a word in the context of a clause and is used to align all the terms to the same term based on the field or subject which is being analyzed—this is especially important in areas with specific jargon, e.g., in clinical settings.)

Words such as "recognized," "recognizable," or "recognition" may be encountered in different usages, but contextually they may all imply the same meaning. For example, "Einstein is a *well-recognized* name in physics" or "The physicist went by the easily *recognizable* name of Einstein" or "Few other physicists have the kind of name *recognition* that Einstein has." The so-called root of all these highlighted words is "recognize." By reducing terms in a document to their basic stems, the conversion of unstructured text to structured data can be simplified because now only the occurrence of the root terms has to be taken into account. This process is called *stemming*. The most common stemming technique for text mining in English is the Porter method (Porter, 1980). Porter stemming works on a bunch of rules where the basic idea is to remove and/or replace the suffix of words. For example, one rule would be: *Replace all terms which end in 'ies' by 'y,'* such as replacing the term "anomalies" with "anomaly." Similarly, another rule would be to stem all terms ending in "s" by removing the "s," as in "algorithms" to "algorithm." While the Porter stemmer is extremely efficient, it can make mistakes that could prove costly. For example, "arms" and "army" would both be stemmed to "arm," which would result in somewhat different contextual meanings. There are other stemmers available; which one is chosen is usually guided by ones experience in various domains. Stemming is usually the next process step following term filtering. (A word of caution: stemming is

Row...	label	RapidMiner	book	book_data	book_descr...	data	data_mining	describes	describes_data	mining	mining_text	mining_usi...	text_0	text_mining	using	using_RapidMiner
1	text1	0	0.447	0.447	0	0.447	0.447	0	0	0.447	0	0	0	0	0	0
2	text2	0.243	0.243	0	0.243	0.243	0.243	0.243	0.243	0.485	0.243	0.243	0.243	0.243	0.243	0.243

FIGURE 9.4
Meaningful *n*-grams show higher TF—IDF scores. *TF—IDF*, Term Frequency—Inverse Document Frequency.

Table 9.3 A Typical Sequence of PreProcessing Steps to Use in Text Mining

Step	Action	Result
1	Tokenize	Convert each word or term in a document into a distinct attribute
2	Stop word removal	Remove highly common grammatical tokens/words
3	Filtering	Remove other very common tokens
4	Stemming	Trim each token to its most essential minimum
5	*n*-grams	Combine commonly occurring token pairs or tuples (more than 2)

completely dependent on the human language being processed as well as the period of the language being processed. Historical usage varies so widely that comparing text across generations—Shakespeare to present-day literature for instance—can raise concerns.)

There are families of words in the spoken and written language that typically go together. For example, the word "Good" is usually followed by either "Morning," "Afternoon," "Evening," "Night," or in Australia, "Day." Grouping such terms, called *n-grams*, and analyzing them statistically can present new insights. Search engines use word *n*-gram models for a variety of applications, such as automatic translation, identifying speech patterns, checking misspelling, entity detection, information extraction, among many other different uses. Google has processed more than a trillion words (1,024,908,267,229 words back as far back as 2006) of running text and has published the counts for all 1,176,470,663 five-word sequences that appear at least 40 times (Franz, 2006). While most text mining applications do not require 5-grams, bigrams and trigrams are quite useful. The final preprocessing step typically involves forming these *n*-grams and storing them in the document vector. Also, most algorithms providing *n*-grams become computationally expensive and the results become huge so in practice the amount of "*n*" will vary based on the size of the documents and the corpus.

Fig. 9.4 shows a TF-based document vector for bigrams ($n = 2$) from the examples and as can be seen, terms like "data mining" and "text mining" and "using RapidMiner" can be quite meaningful in this context. Table 9.3

summarizes a typical sequence of preprocessing steps that will convert unstructured data into a semi-structured format.

Usually there is a preprocessing step before tokenization such as removing special characters, changing the case (up-casing and down-casing), or sometimes even performing a simple spell check beforehand. Data quality in text mining is just as important as in other areas.

9.2 HOW TO IMPLEMENT

A few essential concepts have been introduced that would be needed for a basic text mining project. In the following sections, two case studies will be examined that apply text mining. In the first example, several documents (web pages) will be taken and keywords found in them will be grouped into similar *clusters*. In the second example, a *blog gender classification* will be attempted. To start with several blogs (documents) written by men and women authors, will be used as training data. Using the article keywords as features, several classification models will be trained, including a couple of SVMs, to recognize stylistic characteristics of authors and to classify new unseen blogs as belonging to one of the two author classes (male or female).

9.2.1 Implementation 1: Keyword Clustering

In this first example, some of the web mining features of RapidMiner will be introduced and then a clustering model will be created with keywords data mined from a website. The objective of this case study is to scan several pages from a given website and identify the most frequent words within these pages that also serve to characterize each page, and then to identify the most frequent words using a clustering model. This simple example can be easily extended to a more comprehensive document-clustering problem where the most common words occurring in a document would be used as flags to group multiple documents. The predictive objective of this exercise is to then use the process to identify any random webpage and determine if the page pertains to one of the two categories which the model has been trained to identify.

The site (http://www.detroitperforms.org) that is being looked into is hosted by a public television station and is meant to be used as a platform for reaching out to members of the local community who are interested in arts and culture. The site serves as a medium for the station to not only engage with community members, but also to eventually aid in targeted marketing campaigns meant to attract donors to public broadcasting. The site has pages for several related categories: Music, Dance, Theater, Film, and so on. Each of these pages contains articles and events related to that category. The goal here is to characterize each page on the site and identify the top keywords

that appear on each page. To that end, each category page will be crawled, the content extracted, and the information converted into a structured document vector consisting of keywords. Finally, a *k*-medoids clustering process will be run to sort the keywords and rank them. Medoid clustering is similar to the *k*-means clustering described in Chapter 7, Clustering. A medoid is the most centrally located object in a cluster (Park & Jun, 2009). *k*-Medoids are less susceptible to noise and outliers when compared to *k*-means. This is because *k*-medoids tries to minimize dis-similarities rather than Euclidean distances, which is what *k*-means does.

Before beginning with web page clustering in RapidMiner, make sure that the web mining and text mining extensions are installed. *(This is easily done by going to Help → Updates and Extensions on the main menu bar.)* RapidMiner provides three different ways to crawl and get content from websites. The *Crawl Web* operator will allow setting up of simple crawling rules and based on these rules will store the crawled pages in a directory for further processing. The *Get Page* operator retrieves a single page and stores the content as an example set. The *Get Pages* operator works similarly but can access multiple pages identified by their URLs contained in an input file. The *Get Pages* operator will be used in this example. Both of the *Get Page(s)* operators allow the choosing of either the GET or POST HTTP request methods for retrieving content.[2]

Step 1: Gather Unstructured Data

The first step in this process is to create an input text file containing a list of URLs to be scanned by the *Get Pages* operator. This is specified in the *Read CSV* (renamed in the process shown in Fig. 9.6 to *Read URL List*) operator, which initiates the whole process. The text file consists of three lines: a header line that is needed for the link attribute parameter for Get Pages and two lines containing the two URLs that are going to be crawled, as shown in Fig. 9.5.[3]

pages.txt

```
1 links_to_scan
2 http://www.detroitperforms.org/category/dance/
3 https://www.detroitperforms.org/category/film/
4
```

FIGURE 9.5
Creating a URL read list.

[2] For more information on the differences between the two methods, and when to use which type of request, refer to the tutorials on www.w3schools.com.
[3] Be aware that websites may frequently change their structure or content or be taken down altogether. The results shown here for this example were obtained when the website listed was crawled at the time of writing. Your results may differ depending upon when the process is executed.

The first URL is the Dance category page and the second one is the Film category page on the website. Save the text file as pages.txt as shown in the figure.

The output from the *Get Pages* operator consists of an example set that will contain two main attributes: the URL and extracted HTML content. Additionally, it also adds some metadata attributes that are not needed in this example, such as content length (characters), date, and so on. These extra attributes can be filtered out using the *Select Attributes* operator.

Step 2: Data Preparation

Next, connect the output from this to a *Process Documents from Data* operator. This is a nested operator, which means this operator contains an inner subprocess where all the preprocessing takes place. The first step in this preprocessing is removing all the HTML tags and only preserving the actual content. This is enabled by the *Extract Content* operator. Put this operator inside the *Process Documents from Data* operator and connect the different operators as shown in Figs. 9.6 and 9.7. Refer to Table 9.3 from earlier to see which operators to use. The inset shows the operators inside the nested *Process Documents from Data* operator. In this case, the word occurrences will need to be used for the clustering. So, select Term Occurrences for the vector creation parameter option when configuring the *Process Documents from Data* operator.

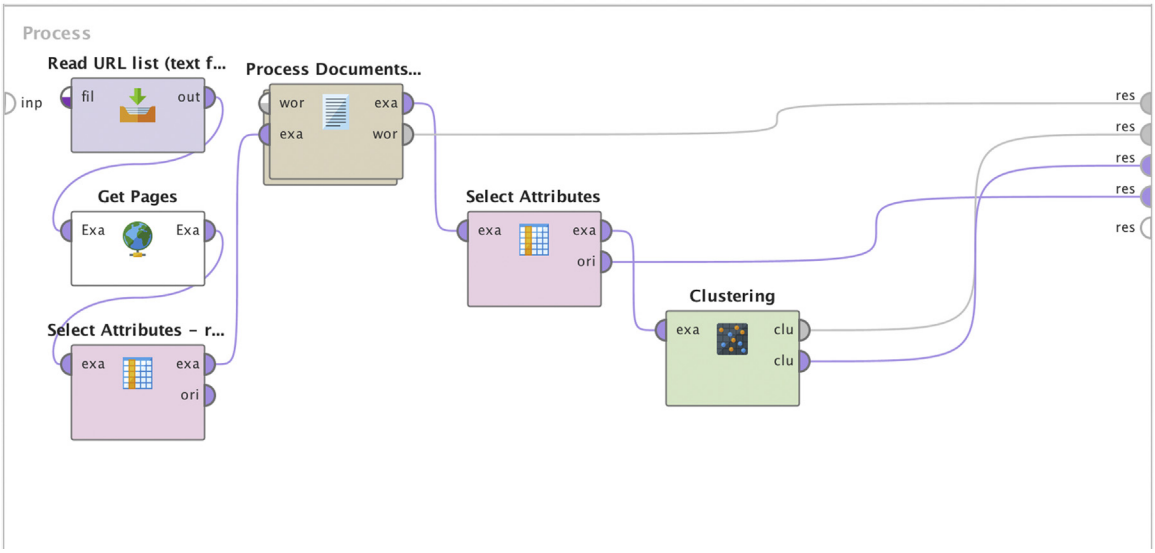


FIGURE 9.6
Overall process of creating keyword clustering from websites.

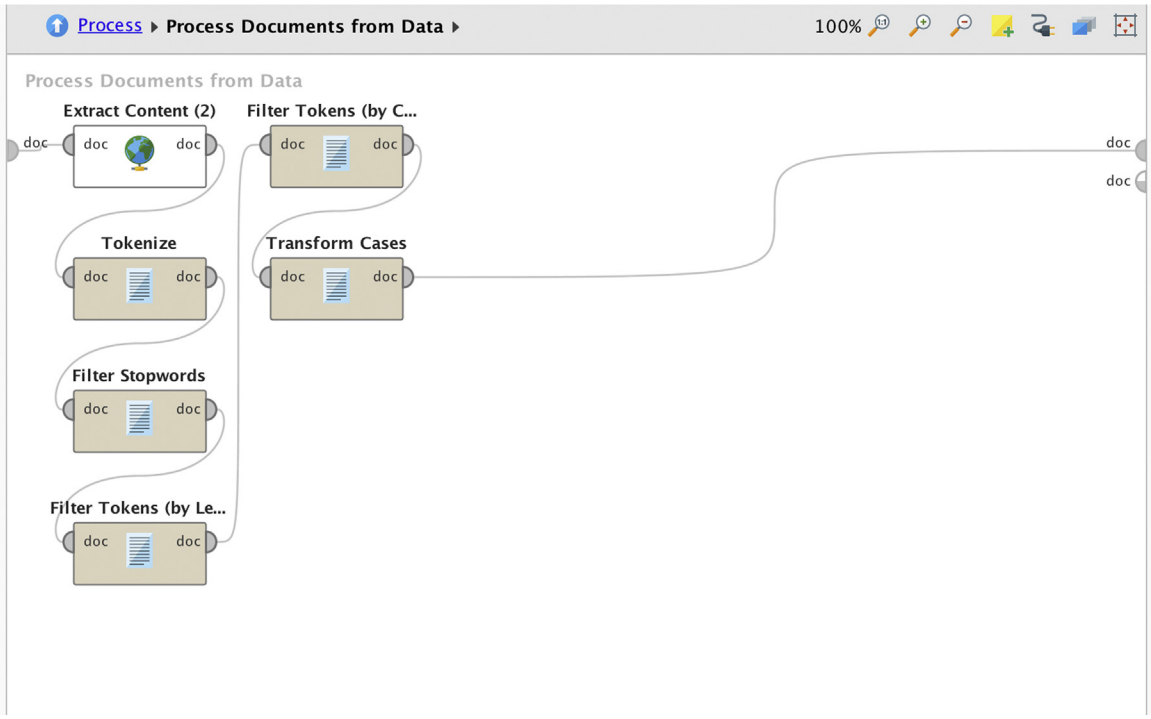


FIGURE 9.7
Configuring the nested preprocessing operator: process documents from data.

Step 3: Apply Clustering

The output from the *Process Documents from Data* operator consists of (1) a word list and (2) a document vector or TDM. The word list is not needed for clustering; however, the document vector is. Recall that the difference between the two is that in the document vector, each word is considered an attribute and each row or example is a separate document (in this case the web pages crawled). The values in the cells of the document vector can of course be word occurrences, word frequencies, or TF−IDF scores, but as noted in step 2, in this case the cells will have word occurrences. The output from the *Process Documents from Data* operator is filtered further to remove attributes that are less than 5 (that is all words that occur less than five times in *both* documents). Notice that RapidMiner will only remove those attributes (words) which occur less than five times in *both* documents—for example, the word "dance" appears only two times in the Film category but is the most common word in the Dance category; it is not and should not be removed! Finally, this cleaned output is fed into a *k*-medoids clustering operator, which is configured as shown in Fig. 9.8.

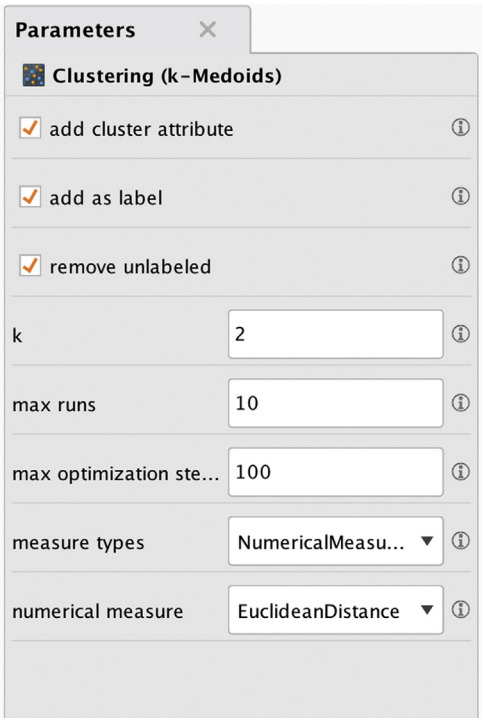


FIGURE 9.8
Configuring the *k*-medoids operator.

Upon running the process, RapidMiner will crawl the two URLs listed and execute the different operations to finally generate two clusters. To view these clustering outputs, select either the Centroid Table or Centroid Plot views in the Cluster Model (Clustering) results tab, which will clearly show the top keywords from each of the two pages crawled. In Fig. 9.9, see the top few keywords that characterize each cluster. One can then use this model to identify if the content of any random page would belong to either one of the categories.

9.2.2 Implementation 2: Predicting the Gender of Blog Authors

The objective of this case study is to attempt to predict the gender of blog authors based[4] on the content of the blog (should it be predicted? why? can it be predicted?).

[4] A compressed version of this data can be downloaded from the Opinion Mining, Sentiment Analysis, and Opinion Spam Detection website (http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html). The data set is called Blog Author Gender Classification dataset associated with the paper (Mukherjee, 2010). This site contains a lot of relevant information related to text mining and sentiment analysis, in addition to several other useful data sets.

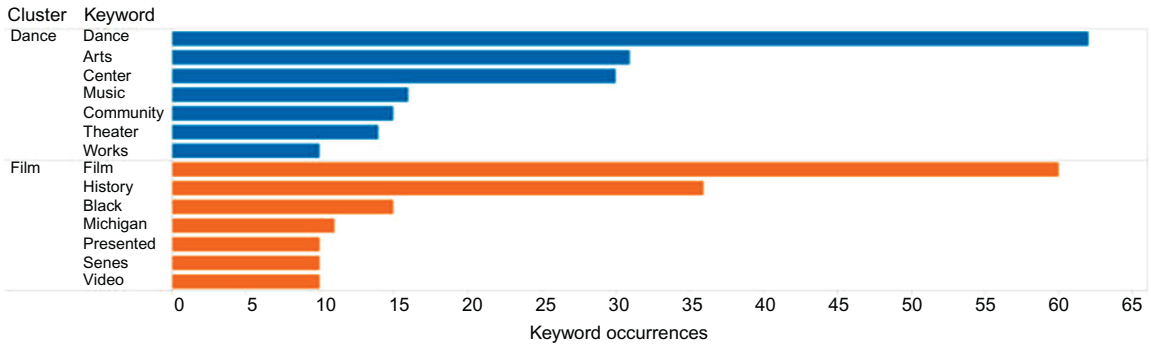


FIGURE 9.9
Results of the website keyword clustering process.

Step 1: Gather Unstructured Data

The data set for this case study consists of more than 3000 individual blog entries (articles) by men and women from around the world (Mukherjee, 2010). The data[4] is organized into a single spreadsheet consisting of 3227 rows and two columns as shown in the sample in Table 9.4. The first column is the actual blog content and the second column is the author's gender, which has been labeled.

For the purpose of this case study, the raw data will be split into two halves: the first 50% of the data is treated as training data with known labels and the remaining 50% is set aside to verify the performance of the training algorithm.

While developing models involving large amounts of data, which is common with unstructured text analysis, it is a good practice to divide the process into several distinct processes and store the intermediate data, models, and results from each process for recall at a later stage. RapidMiner facilitates this by providing special operators called *Store* and *Retrieve*. The *Store* operator stores an input-ouput (IO) Object in the data repository and *Retrieve* reads an object from the data repository. The use of these operators is introduced in the coming sections.

Step 2: Data Preparation

Once the data is downloaded and uncompressed, it yields a single MS Excel file, which can then be imported into the RapidMiner database using the *Read Excel* operator. The raw data consists of 290 examples that do not have a label and one example that has no blog content but has a label! This needs to be cleaned up. It is easier to delete this entry in the raw data—simply delete the row (#1523) in the spreadsheet that contains this missing entry and save the file before reading it into RapidMiner. Also

Table 9.4 Raw Data for the Blog Classification Study

Blog	Gender
This game was a blast. You (as Drake) start the game waking up in a train that is dangling over the side of a cliff. You have to climb up the train car, which is slowly teetering off the edge of the cliff, ready to plummet miles down into a snowy abyss. From the snowy beginning there are flashbacks to what led Drake to this predicament. The story unfolds in a very cinematic manner, and the scenes in between levels, while a bit clichéd by Hollywood standards, are still just as good if not better than your average brainless Mel Gibson or Bruce Willis action movie. In fact, the cheese is part of the fun and I would venture to say it's intentional	M
My mother was a contrarian, she was. For instance, she always wore orange on St. Patrick's Day, something that I of course did not understand at the time, nor, come to think of it do I understand today. Protestants wear orange in Ireland, not here, but I'm pretty sure my mother had nothing against the Catholics, so why did she do it? Maybe it had to do with the myth about Patrick driving the snakes, a.k.a. pagans, out of Ireland. Or maybe it was something political. I have no idea and since my mother is long gone from this earth, I guess I'll never know	F
LaLicious Sugar Soufflé body scrub has a devoted following and I now understand why. I received a sample of this body scrub in Tahitian Flower and after one shower with this tub of sugary goodness, I was hooked. The lush scent is deliciously intoxicating and it ended up inspiring compliments and extended sniffing from both loved ones and strangers alike. Furthermore, this scrub packs one heck of a punch when it comes to pampering dry skin. In fact, LaLicious promises that this body scrub is so rich that it will eliminate the need for applying your post-shower lotion. This claim is true—if you follow the directions	F
Stopped by the post office this morning to pick up a package on my way to the lab. I thought it would be as good a time as any to clean up my desk and at the very least make it appear that I am more organized than I really am (seriously, it's a mess). It's pretty nice here on the weekends, it's quiet, there's less worry of disturbing undergrad classes if I do any experiments in the daytime	M
Anyway, it turns out the T-shirt I ordered from Concrete Rocket arrived! Here's how the design looks	
See here's the thing: Men have their neat little boxes through which they compartmentalize their lives. Relationship over? Oh, I'll just close that box. It's not that easy for women. Our relationships are not just a section of our lives—they run through the entire fabric, a hot pink thread which adds to the mosaic composing who we are. Take out a relationship and you grab that thread and pull. Have you ever pulled a thread on a knit sweater? That's what it's like. The whole garment gets scrunched and disfigured just because that one piece was removed. And then you have to pull it back apart, smooth it out, fill in the gaps. See here's the thing: men have their neat little boxes through which they compartmentalize their lives. Relationship over? Oh, I'll just close that box. It's not that easy for women	F

make sure that the *Read Excel* operator is configured to recognize the data type in the first column as text and not polynominal (default) as shown in Fig. 9.10. Connect the output from *Read Excel* to a *Filter Examples* operator, where the entries with missing labels will then be removed. (If inclined, store these entries with missing labels for use as testing samples—this can be accomplished with another *Filter Examples*, but by checking *Invert Filter* box and then storing the output. In this case, however, examples with missing labels will simply be discarded.) The cleaned data can now be separated with a 50/50 split using a *Split Data* operator. Save the latter 50%

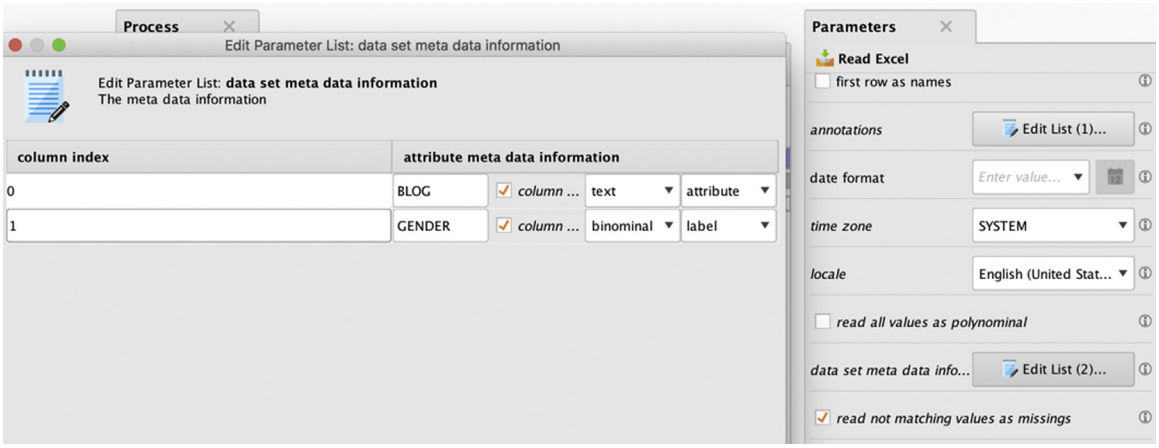


FIGURE 9.10
Properly configuring the Read Excel operator to accept text (not polynomial).

testing data (1468 samples) to a new file with a *Write Excel* operator and pass the remaining 50% training portion to a *Process Documents from Data* operator.

This is a nested operator where all the preprocessing happens. Recall that this is where the conversion of unstructured data into a structured format will take place. Connect the different operators within as shown in Fig. 9.13. The only point to note here is that a *Filter Stop word (Dictionary)* operator will be needed to remove any "nbsp" (" " is used to represent a nonbreaking space) terms that may have slipped into the content. Create a simple text file with this keyword inside it and let RapidMiner know that this dictionary exists by properly configuring the operator. To configure the *Process Documents from Data* operator, use the options as shown in Fig. 9.11.

The output from the process for step 2 consists of the document vector and a word list. While the word list may not be of immediate use in the subsequent steps, it is a good idea to store this along with the very important document vector. The final process is shown in Fig. 9.12.

Step 3.1: Identify Key Features

The document vector that is the result of the process in step 2 is a structured table consisting of 2055 rows—one for every blog entry in the training set—and 2815 attributes or columns—each token within an article that meets the filtering and stemming criteria defined by operators inside *Process Documents* is converted into an attribute. Training learning algorithms using 2815 features or variables is clearly an onerous task. The right approach is to further filter these attributes by using feature selection methods.

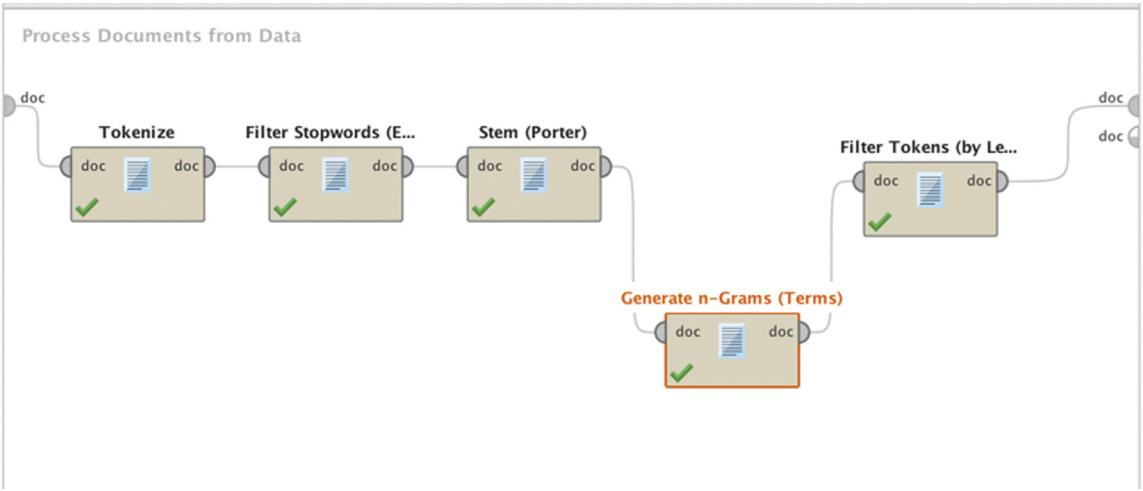


FIGURE 9.11
Configuring the preprocessing operator.

Two feature selection methods will be employed using the *Weight by Information Gain* and *Weight by SVM* operators that are available. *Weight by Information Gain* (more details in Chapter 14: Feature Selection, on this operator) will rank a feature or attribute by its relevance to the label attribute (in this case, gender) based on the information gain ratio and assign weights to them accordingly. *Weight by SVM* will set the coefficients of the SVM hyperplane as attribute weights. Once they are ranked using these techniques, only a handful of attributes can be selected (e.g., the top 20) to build the models. Doing so will result in a reasonable reduction in modeling costs.

The results of this intermediate process will generate two weight tables, one corresponding to each feature selection method. The process is started by retrieving the document vector saved in step 2 and then the process is ended by storing the weight tables for use in step 3.2 (see Fig. 9.14).

In the paper by Mukherjee and Liu (Mukherjee, 2010) from which this data set comes, they demonstrate the use of several other feature selection methods, including a novel one developed by the authors that is shown to yield a much higher prediction accuracy than the stock algorithms (such as the ones demonstrated here).

Step 3.2: Build Models

With the document vector and attribute weights now ready, one can experiment using several different machine learning algorithms to understand

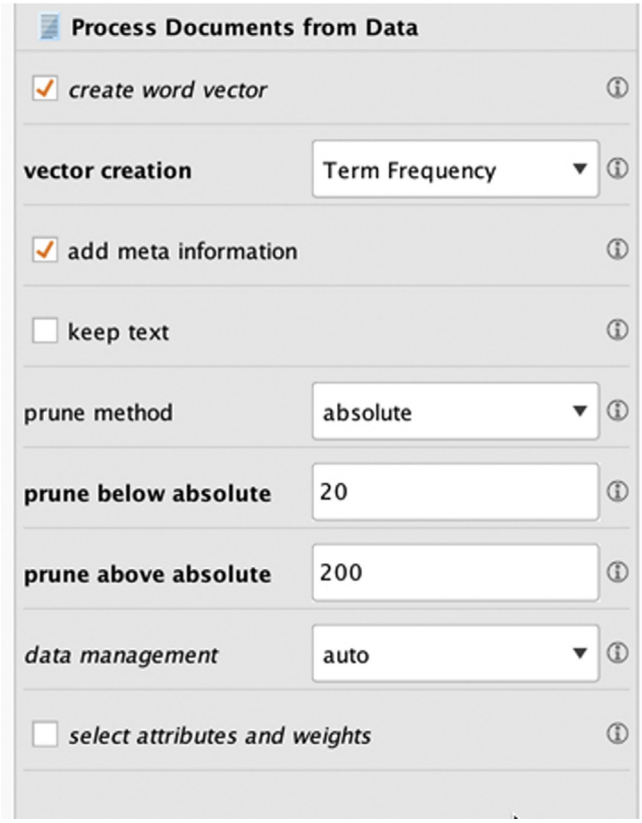


FIGURE 9.12
Overall process for blog gender classification.

which gives the best accuracy. The process illustrated in Figs. 9.15 and 9.16 will generate the models and store them (along with the corresponding performance results) for later application. This is a key strength of RapidMiner: once one has built up the necessary data for predictive modeling, switching back and forth between various algorithms requires nothing more than dragging and dropping the needed operators and making the connections. As seen in Fig. 9.16, there are four different algorithms nested inside the *X-Validation* operator and can conveniently be switched back and forth as needed. Table 9.5 shows that the *LibSVM (linear)* and *W-Logistic* operators (Available through Weka extension for RapidMiner) seem to give the best performance. Keep in mind that these accuracies are still not the highest and are in line with the performances reported by Mukherjee and Liu in their paper for generic algorithms.

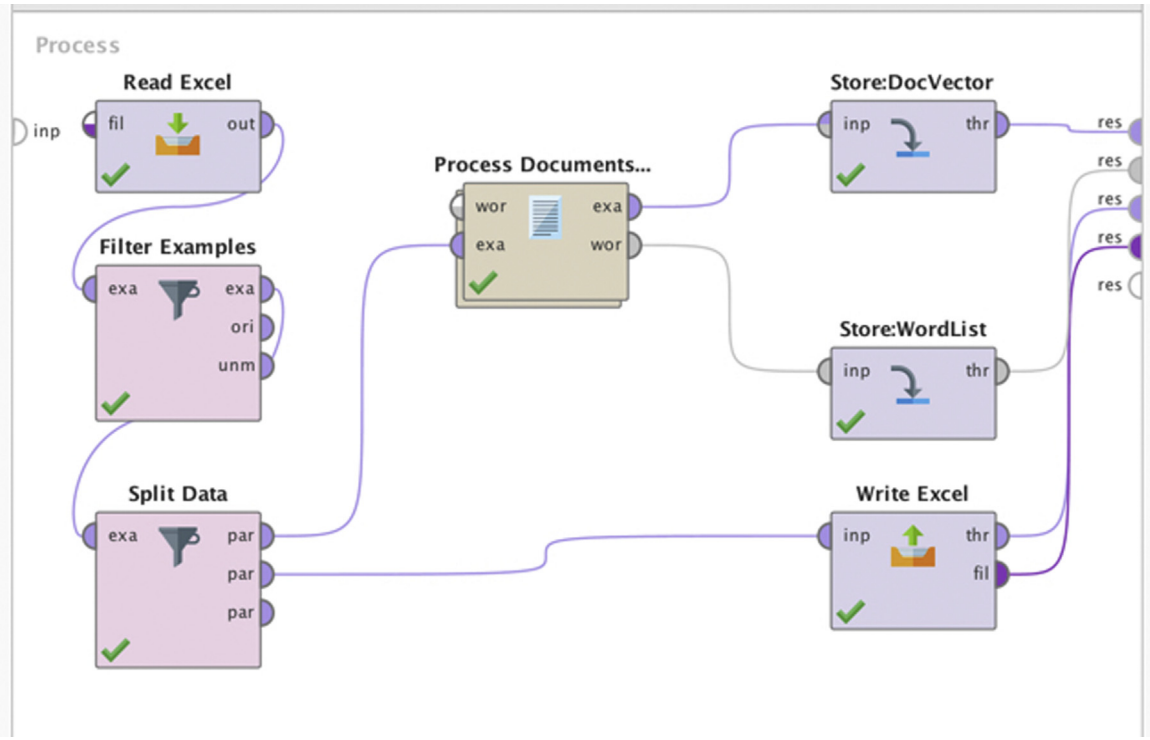


FIGURE 9.13
Preprocessing text data using the process documents from data operator.

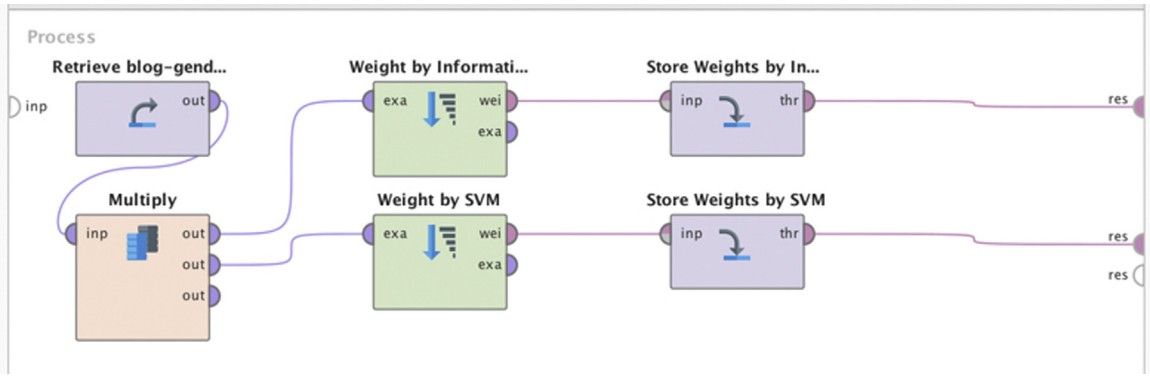


FIGURE 9.14
Using feature selection methods to filter attributes from the TDM. *TDM*, Term Document Matrix.

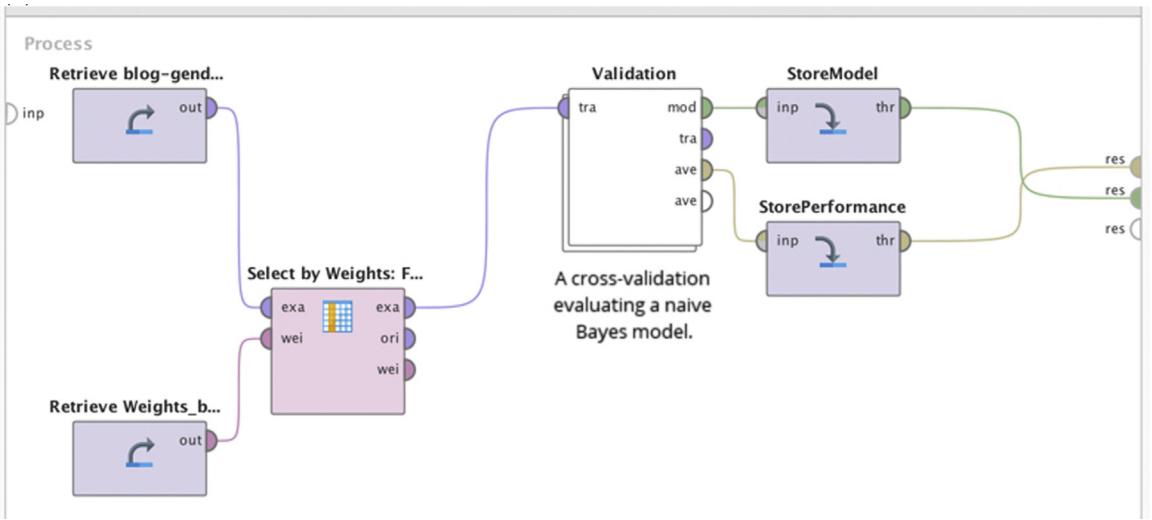


FIGURE 9.15
Training and testing predictive models for blog gender classification.

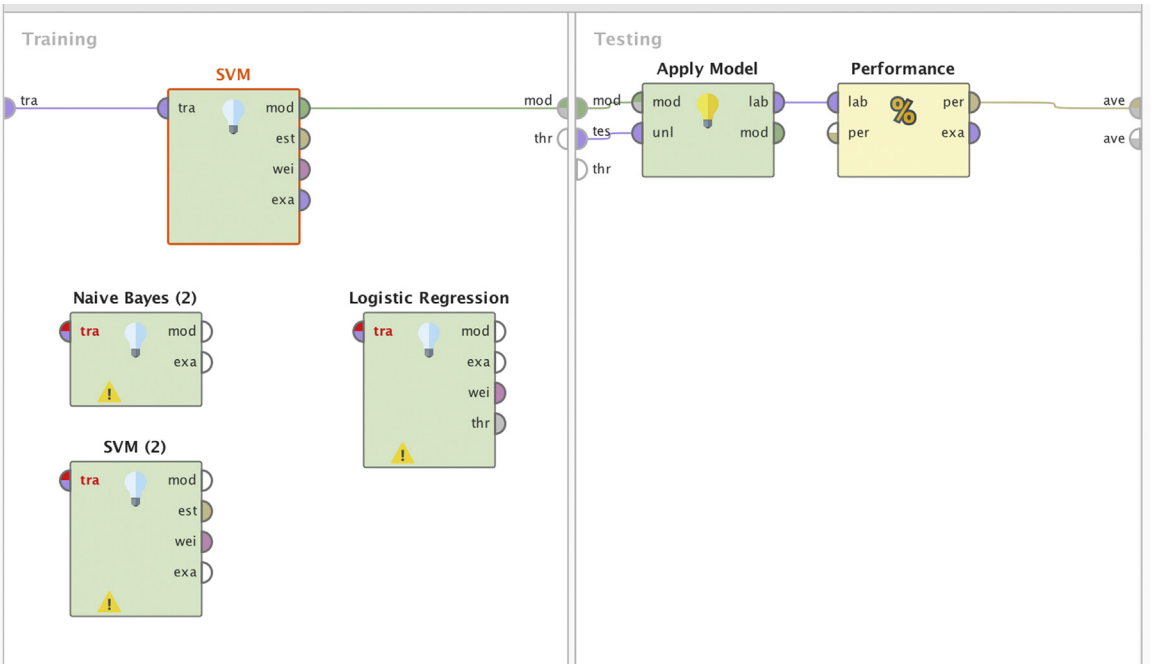


FIGURE 9.16
Switching between several algorithms.

Table 9.5 Comparing the Performance of Different Training Algorithms for Blog Gender Classification

Algorithm	Class Recall (M)	Class Recall (F)	Accuracy
LibSVM (linear)	87	53	72
W-Logistic	85	58	73
Naïve Bayes	86	55	72
SVM (polynomial)	82	42	63

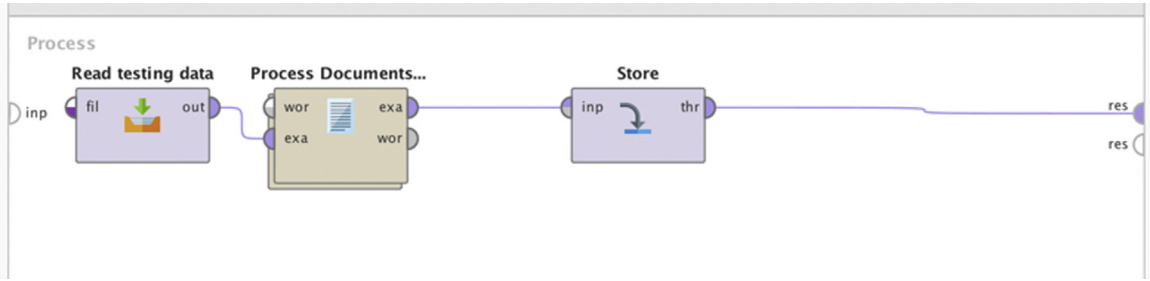


FIGURE 9.17
Preparing the unseen data for model deployment.

To improve upon these, we may need to further optimize the best performers so far by nesting the entire validation process within an optimization operator. This is described in the Chapter 15: Getting started with RapidMiner, in the section on optimization.

Step 4.1: Prepare Test Data for Model Application

Going back to the original 50% of the unseen data that was saved for testing purposes, the real-world performance of the best algorithm can actually be evaluated in classifying blogs by author gender. However, keep in mind that the raw data that was set aside as is, cannot be used (what would happen if one did?). This raw test data would also need to be converted into a document vector first. In other words, the step 2 process needs to be repeated (without the filtering and split data operators) on the 50% of the data that were set aside for testing. The *Process Documents from Data* operator can be simply copied and pasted from the process in step 2. (Alternatively, the entire data set could have been preprocessed before splitting!) The document vector is stored for use in the next step. This process is illustrated in Fig. 9.17.

Step 4.2: Applying the Trained Models to Testing Data

This is where the rubber hits the road! The last step will take any of the saved models created in step 3.2 and the newly created document vector from step

4.1 and apply the model on this test data. The process is shown in Figs. 9.18 and 9.19. One useful operator to add is the *Set Role* operator, which will be used to indicate to RapidMiner the label variable. Doing so will allow the results to be sorted from the Apply Model by Correct Predictions and Wrong Predictions using the View Filter in the Results perspective as shown here.

When this process is run, it can be observed that the LibSVM (linear) model can correctly predict only 828 of the 1468 examples, which translates to a poor 56% accuracy! The other models fare worse. Clearly the model and the process are in need of optimization and further refinement. Using RapidMiner's optimization operators, one can easily improve on this baseline accuracy. A discussion about how to use the optimization operators in general is provided in Chapter 15, Getting started with RapidMiner. The truly adventurous can implement the Mukherjee and Liu algorithm for feature selection in RapidMiner based on the instructions given in their paper! Although this implementation did not return a stellar predictive performance (and the motivation of this classification is digital advertisement, as specified in the paper), a word of caution is in order: algorithms have become increasingly more powerful and reckless application of machine learning may lead to undesirable consequences of discrimination (via gender, in this instance). Data scientists are responsible for ensuring that their products are used in an ethical and non-discriminatory manner.

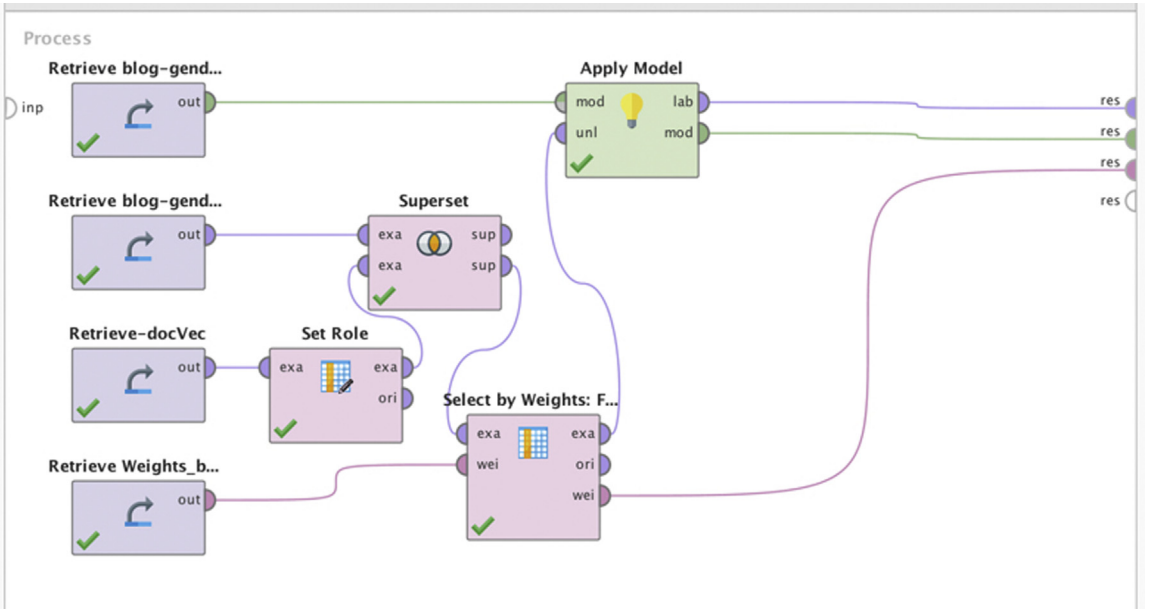


FIGURE 9.18
Applying the models built in step 3 on the unseen data.

Row No.	GENDER	predict...	confide...	confide...	abil	activ	adapt	admir	ador	adult	advanc	ahead	amount
1	M	M	1	0	0	0	0	0	0	0	0	0	0
2	M	M	1	0	0	0	0	0	0	0	0	0	0
3	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
4	M	M	1	0	0	0	0	0	0	0	0	0	0
5	M	M	0.500	0.500	0	0	0	0	0	0.186	0	0	0
6	M	M	1	0	0	0	0	0	0	0	0	0	0
7	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
8	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
9	M	M	1	0	0	0	0	0	0	0	0	0	0
10	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
11	M	M	0.500	0.500	0	0	0	0	0	0	0	0	0
12	M	M	1	0	0	0	0	0	0	0	0	0	0

FIGURE 9.19
The results view.

Bias in Machine Learning

Data Science is a powerful tool to extract value from data. Just like any other tool, it can be put to good use, inappropriate use, malicious use or use it in such a way that yield unintended consequences. Recently, a data science model that was developed to filter and sort the resumes of job applicants started to discriminate against women (Gershgorn, 2018). The data science modeling process would have started as a solution to a right business problem, which is to manage the influx of resumes and sorting the most relevant ones to the top. In doing so, the text mining model favoured one applicant class, leveraging some spurious pattern in the data. If the training data is biased, the machine learning model will be biased. Specifically, if the training data is derived from a biased process, the machine learning automation just amplifies the phenomenon. A loan approval model might not ask for the race of the applicant (which would be unethical and illegal). However, the location of an applicant might serve as a proxy for the race. It is important to test and audit if the model provides fair prediction for all the classes of users. These machine learning models have real world consequences. The responsibility lies with the data scientist to build a transparent model adhering to ethical principles. Creating robust tests to check for potential unfairness in the models is imperative to gain trust in the discipline of data science (Loukides et al., 2018).

9.3 CONCLUSION

Unstructured data, of which text data is a major portion, appears to be doubling in volume every three years (Mayer-Schonberger, 2013). The ability to