CHAPTER 1

Introduction

1.1 WHAT IS BUSINESS ANALYTICS?

Business Analytics (BA) is the practice and art of bringing quantitative data to bear on decision-making. The term means different things to different organizations.

Consider the role of analytics in helping newspapers survive the transition to a digital world. One tabloid newspaper with a working-class readership in Britain had launched a web version of the paper, and did tests on its home page to determine which images produced more hits: cats, dogs, or monkeys. This simple application, for this company, was considered analytics. By contrast, the *Washington Post* has a highly influential audience that is of interest to big defense contractors: it is perhaps the only newspaper where you routinely see advertisements for aircraft carriers. In the digital environment, the *Post* can track readers by time of day, location, and user subscription information. In this fashion, the display of the aircraft carrier advertisement in the online paper may be focused on a very small group of individuals—say, the members of the House and Senate Armed Services Committees who will be voting on the Pentagon's budget.

Business Analytics, or more generically, *analytics*, include a range of data analysis methods. Many powerful applications involve little more than counting, rule-checking, and basic arithmetic. For some organizations, this is what is meant by analytics.

The next level of business analytics, now termed *Business Intelligence* (BI), refers to data visualization and reporting for understanding "what happened and what is happening." This is done by use of charts, tables, and dashboards to display, examine, and explore data. BI, which earlier consisted mainly of generating static reports, has evolved into more user-friendly and effective tools and practices, such as creating interactive dashboards that allow the user not only to

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Statistics.com PA 3041219
may not be redistributed

access real-time data, but also to directly interact with it. Effective dashboards are those that tie directly into company data, and give managers a tool to quickly see what might not readily be apparent in a large complex database. One such tool for industrial operations managers displays customer orders in a single two-dimensional display, using color and bubble size as added variables, showing customer name, type of product, size of order, and length of time to produce.

Business Analytics now typically includes BI as well as sophisticated data analysis methods, such as statistical models and data mining algorithms used for exploring data, quantifying and explaining relationships between measurements, and predicting new records. Methods like regression models are used to describe and quantify "on average" relationships (e.g., between advertising and sales), to predict new records (e.g., whether a new patient will react positively to a medication), and to forecast future values (e.g., next week's web traffic).

Readers familiar with earlier editions of this book may have noticed that the book title has changed from *Data Mining for Business Intelligence* to *Data Mining for Business Analytics* in this edition. The change reflects the more recent term BA, which overtook the earlier term BI to denote advanced analytics. Today, BI is used to refer to data visualization and reporting.

WHO USES PREDICTIVE ANALYTICS?

The widespread adoption of predictive analytics, coupled with the accelerating availability of data, has increased organizations' capabilities throughout the economy. A few examples:

Credit scoring: One long-established use of predictive modeling techniques for business prediction is credit scoring. A credit score is not some arbitrary judgment of credit-worthiness; it is based mainly on a predictive model that uses prior data to predict repayment behavior.

Future purchases: A more recent (and controversial) example is Target's use of predictive modeling to classify sales prospects as "pregnant" or "not-pregnant." Those classified as pregnant could then be sent sales promotions at an early stage of pregnancy, giving Target a head start on a significant purchase stream.

Tax evasion: The US Internal Revenue Service found it was 25 times more likely to find tax evasion when enforcement activity was based on predictive models, allowing agents to focus on the most-likely tax cheats (Siegel, 2013).

The Business Analytics toolkit also includes statistical experiments, the most common of which is known to marketers as A-B testing. These are often used for pricing decisions:

- Orbitz, the travel site, found that it could price hotel options higher for Mac users than Windows users.
- Staples online store found it could charge more for staplers if a customer lived far from a Staples store.

Statistics.com FAS 04.12.19 may not be redistributed

Beware the organizational setting where analytics is a solution in search of a problem: A manager, knowing that business analytics and data mining are hot areas, decides that her organization must deploy them too, to capture that hidden value that must be lurking somewhere. Successful use of analytics and data mining requires both an understanding of the business context where value is to be captured, and an understanding of exactly what the data mining methods do.

1.2 What Is Data Mining?

In this book, data mining refers to business analytics methods that go beyond counts, descriptive techniques, reporting, and methods based on business rules. While we do introduce data visualization, which is commonly the first step into more advanced analytics, the book focuses mostly on the more advanced data analytics tools. Specifically, it includes statistical and machine-learning methods that inform decision-making, often in an automated fashion. Prediction is typically an important component, often at the individual level. Rather than "what is the relationship between advertising and sales," we might be interested in "what specific advertisement, or recommended product, should be shown to a given online shopper at this moment?" Or we might be interested in clustering customers into different "personas" that receive different marketing treatment, then assigning each new prospect to one of these personas.

The era of Big Data has accelerated the use of data mining. Data mining methods, with their power and automaticity, have the ability to cope with huge amounts of data and extract value.

1.3 Data Mining and Related Terms

The field of analytics is growing rapidly, both in terms of the breadth of applications, and in terms of the number of organizations using advanced analytics. As a result, there is considerable overlap and inconsistency of definitions.

The term *data mining* itself means different things to different people. To the general public, it may have a general, somewhat hazy and pejorative meaning of digging through vast stores of (often personal) data in search of something interesting. One major consulting firm has a "data mining department," but its responsibilities are in the area of studying and graphing past data in search of general trends. And, to confuse matters, their more advanced predictive models are the responsibility of an "advanced analytics department." Other terms that organizations use are *predictive analytics*, *predictive modeling*, and *machine learning.*

Data mining stands at the confluence of the fields of statistics and machine learning (also known as *artificial intelligence*). A variety of techniques for exploring data and building models have been around for a long time in the world of

Statistics.com PA 3041213 may not be redistributed

statistics: linear regression, logistic regression, discriminant analysis, and principal components analysis, for example. But the core tenets of classical statistics—computing is difficult and data are scarce—do not apply in data mining applications where both data and computing power are plentiful.

This gives rise to Daryl Pregibon's description of data mining as "statistics at scale and speed" (Pregibon, 1999). Another major difference between the fields of statistics and machine learning is the focus in statistics on inference from a sample to the population regarding an "average effect"—for example, "a $1 price increase will reduce average demand by 2 boxes." In contrast, the focus in machine learning is on predicting individual records—"the predicted demand for person i given a $1 price increase is 1 box, while for person j it is 3 boxes." The emphasis that classical statistics places on inference (determining whether a pattern or interesting result might have happened by chance in our sample) is absent from data mining.

In comparison to statistics, data mining deals with large datasets in an open-ended fashion, making it impossible to put the strict limits around the question being addressed that inference would require. As a result, the general approach to data mining is vulnerable to the danger of *overfitting*, where a model is fit so closely to the available sample of data that it describes not merely structural characteristics of the data, but random peculiarities as well. In engineering terms, the model is fitting the noise, not just the signal.

In this book, we use the term *machine learning* to refer to algorithms that learn directly from data, especially local patterns, often in layered or iterative fashion. In contrast, we use *statistical models* to refer to methods that apply global structure to the data. A simple example is a linear regression model (statistical) vs. a k-nearest-neighbors algorithm (machine learning). A given record would be treated by linear regression in accord with an overall linear equation that applies to *all* the records. In k-nearest neighbors, that record would be classified in accord with the values of a small number of nearby records.

Lastly, many practitioners, particularly those from the IT and computer science communities, use the term *machine learning* to refer to all the methods discussed in this book.

1.4 Big Data

Data mining and Big Data go hand in hand. *Big Data* is a relative term—data today are big by reference to the past, and to the methods and devices available to deal with them. The challenge Big Data presents is often characterized by the four V's—volume, velocity, variety, and veracity. *Volume* refers to the amount of data. *Velocity* refers to the flow rate—the speed at which it is being generated and changed. *Variety* refers to the different types of data being generated (currency,

Statistics.com LLC 2019 may not be distributed

dates, numbers, text, etc.). *Veracity* refers to the fact that data is being generated by organic distributed processes (e.g., millions of people signing up for services or free downloads) and not subject to the controls or quality checks that apply to data collected for a study.

Most large organizations face both the challenge and the opportunity of Big Data because most routine data processes now generate data that can be stored and, possibly, analyzed. The scale can be visualized by comparing the data in a traditional statistical analysis (say, 15 variables and 5000 records) to the Walmart database. If you consider the traditional statistical study to be the size of a period at the end of a sentence, then the Walmart database is the size of a football field. And that probably does not include other data associated with Walmart—social media data, for example, which comes in the form of unstructured text.

If the analytical challenge is substantial, so can be the reward:

- OKCupid, the online dating site, uses statistical models with their data to predict what forms of message content are most likely to produce a response.
- Telenor, a Norwegian mobile phone service company, was able to reduce subscriber turnover 37% by using models to predict which customers were most likely to leave, and then lavishing attention on them.
- Allstate, the insurance company, tripled the accuracy of predicting injury liability in auto claims by incorporating more information about vehicle type.

The above examples are from Eric Siegel's book *Predictive Analytics* (2013, Wiley).

Some extremely valuable tasks were not even feasible before the era of Big Data. Consider web searches, the technology on which Google was built. In early days, a search for "Ricky Ricardo Little Red Riding Hood" would have yielded various links to the *I Love Lucy* TV show, other links to Ricardo's career as a band leader, and links to the children's story of Little Red Riding Hood. Only once the Google database had accumulated sufficient data (including records of what users clicked on) would the search yield, in the top position, links to the specific *I Love Lucy* episode in which Ricky enacts, in a comic mixture of Spanish and English, Little Red Riding Hood for his infant son.

1.5 Data Science

The ubiquity, size, value, and importance of Big Data has given rise to a new profession: the *data scientist*. *Data science* is a mix of skills in the areas of statistics, machine learning, math, programming, business, and IT. The term itself is thus broader than the other concepts we discussed above, and it is a rare individual who combines deep skills in all the constituent areas. In their book *Analyzing*

Statistics.com PA 5/04/2019 may not be redistributed

the Analyzers (Harris et al., 2013), the authors describe the skill sets of most data scientists as resembling a 'T'—deep in one area (the vertical bar of the T), and shallower in other areas (the top of the T).

At a large data science conference session (Strata+Hadoop World, October 2014), most attendees felt that programming was an essential skill, though there was a sizable minority who felt otherwise. And, although Big Data is the motivating power behind the growth of data science, most data scientists do not actually spend most of their time working with terabyte-size or larger data.

Data of the terabyte or larger size would be involved at the deployment stage of a model. There are manifold challenges at that stage, most of them IT and programming issues related to data-handling and tying together different components of a system. Much work must precede that phase. It is that earlier piloting and prototyping phase on which this book focuses—developing the statistical and machine learning models that will eventually be plugged into a deployed system. What methods do you use with what sorts of data and problems? How do the methods work? What are their requirements, their strengths, their weaknesses? How do you assess their performance?

1.6 WHY ARE THERE SO MANY DIFFERENT METHODS?

As can be seen in this book or any other resource on data mining, there are many different methods for prediction and classification. You might ask yourself why they coexist, and whether some are better than others. The answer is that each method has advantages and disadvantages. The usefulness of a method can depend on factors such as the size of the dataset, the types of patterns that exist in the data, whether the data meet some underlying assumptions of the method, how noisy the data are, and the particular goal of the analysis. A small illustration is shown in Figure 1.1, where the goal is to find a combination of *household income level* and *household lot size* that separates buyers (blue solid circles)

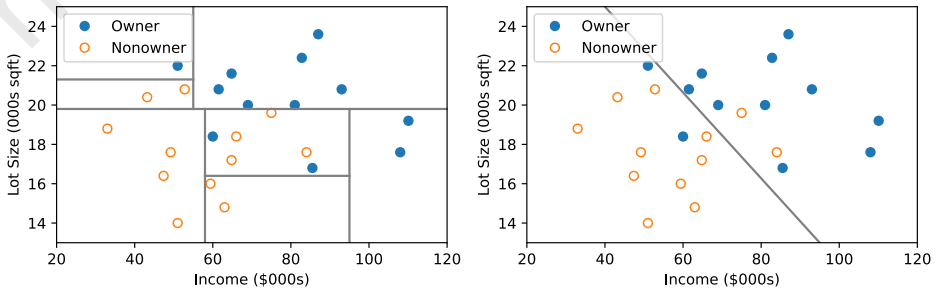

FIGURE 1.1 TWO METHODS FOR SEPARATING OWNERS FROM NONOWNERS

Statistics.com PA 3042219 may not be redistributed

from nonbuyers (orange hollow circles) of riding mowers. The first method (left panel) looks only for horizontal and vertical lines to separate buyers from nonbuyers, whereas the second method (right panel) looks for a single diagonal line.

Different methods can lead to different results, and their performance can vary. It is therefore customary in data mining to apply several different methods and select the one that appears most useful for the goal at hand.

1.7 Terminology and Notation

Because of the hybrid parentry of data mining, its practitioners often use multiple terms to refer to the same thing. For example, in the machine learning (artificial intelligence) field, the variable being predicted is the output variable or target variable. To a statistician, it is the dependent variable or the response. Here is a summary of terms used:

Algorithm A specific procedure used to implement a particular data mining technique: classification tree, discriminant analysis, and the like.

Attribute see **Predictor**.

Case see **Observation**.

Categorical Variable A variable that takes on one of several fixed values, e.g. a flight could be on-time, delayed or canceled.

Confidence A performance measure in association rules of the type "IF A and B are purchased, THEN C is also purchased." Confidence is the conditional probability that C will be purchased IF A and B are purchased.

Confidence also has a broader meaning in statistics (*confidence interval*), concerning the degree of error in an estimate that results from selecting one sample as opposed to another.

Dependent Variable see **Response**.

Estimation see **Prediction**.

Factor Variable see **Categorical variable**

Feature see **Predictor**.

Holdout Data (or **holdout set**) A sample of data not used in fitting a model, but instead used to assess the performance of that model. This book uses the terms *validation set* and *test set* instead of *holdout set*.

Input Variable see **Predictor**.

Model An algorithm as applied to a dataset, complete with its settings (many of the algorithms have parameters that the user can adjust).

Statistics.com PA 3 04 12 19
may not be redistributed

Observation The unit of analysis on which the measurements are taken (a customer, a transaction, etc.), also called *instance*, *sample*, *example*, *case*, *record*, *pattern*, or *row*. In spreadsheets, each row typically represents a record; each column, a variable. Note that the term "sample" here is different from its usual meaning in statistics, where it refers to a collection of observations.

Outcome Variable see **Response**.

Output Variable see **Response**.

***P* (*A* | *B*)** The conditional probability of event A occurring given that event B has occurred, read as "the probability that A will occur given that B has occurred."

Prediction The prediction of the numerical value of a continuous output variable; also called *estimation*.

Predictor A variable, usually denoted by X, used as an input into a predictive model, also called a *feature*, *input variable*, *independent variable*, or from a database perspective, a *field*.

Profile A set of measurements on an observation (e.g., the height, weight, and age of a person).

Record see **Observation**.

Response A variable, usually denoted by Y, which is the variable being predicted in supervised learning, also called *dependent variable*, *output variable*, *target variable*, or *outcome variable*.

Sample In the statistical community, "sample" means a collection of observations. In the machine learning community, "sample" means a single observation.

Score A predicted value or class. *Scoring new data* means using a model developed with training data to predict output values in new data.

Success Class The class of interest in a binary outcome (e.g., *purchasers* in the outcome *purchase/no purchase*).

Supervised Learning The process of providing an algorithm (logistic regression, regression tree, etc.) with records in which an output variable of interest is known and the algorithm "learns" how to predict this value with new records where the output is unknown.

Target see **Response**.

Test Data (or **test set**) The portion of the data used only at the end of the model building and selection process to assess how well the final model might perform on new data.

Training Data (or **training set**) The portion of the data used to fit a model.

Unsupervised Learning An analysis in which one attempts to learn patterns in the data other than predicting an output value of interest.

Statistics.com P/A 3 041219 may not be redistributed

Validation Data (or **validation set**) The portion of the data used to assess how well the model fits, to adjust models, and to select the best model from among those that have been tried.

Variable Any measurement on the records, including both the input (X) variables and the output (Y) variable.

1.8 Road Maps to This Book

The book covers many of the widely used predictive and classification methods as well as other data mining tools. Figure 1.2 outlines data mining from a process perspective and where the topics in this book fit in. Chapter numbers are indicated beside the topic. Table 1.1 provides a different perspective: it organizes data mining procedures according to the type and structure of the data.

Order of Topics

The book is divided into five parts: Part I (Chapters 1–2) gives a general overview of data mining and its components. Part II (Chapters 3–4) focuses on the early stages of data exploration and dimension reduction.

Part III (Chapter 5) discusses performance evaluation. Although it contains only one chapter, we discuss a variety of topics, from predictive performance metrics to misclassification costs. The principles covered in this part are crucial for the proper evaluation and comparison of supervised learning methods.

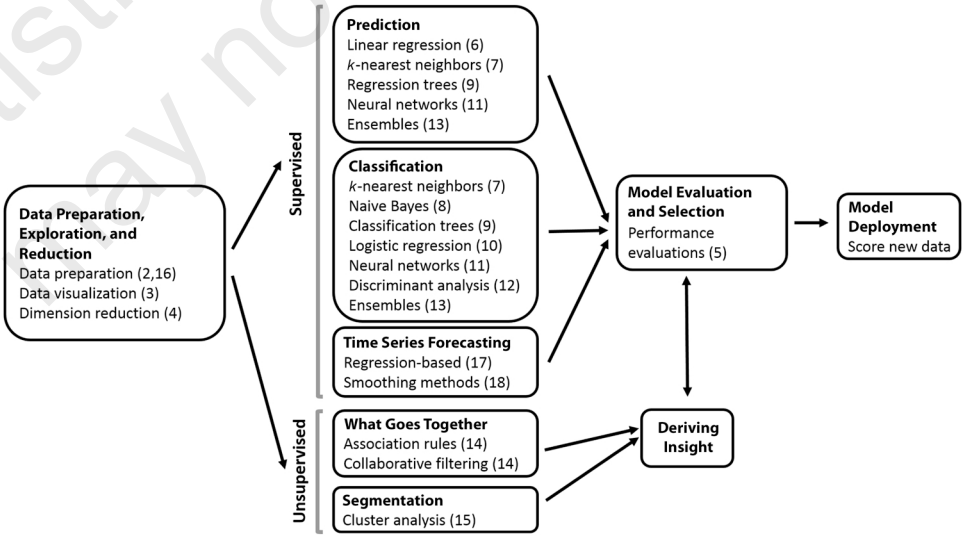

FIGURE 1.2 DATA MINING FROM A PROCESS PERSPECTIVE. NUMBERS IN PARENTHESES INDICATE CHAPTER NUMBERS

Statistics.com PA 3041219 may not be redistributed

TABLE 1.1 ORGANIZATION OF DATA MINING METHODS IN THIS BOOK, ACCORDING TO THE NATURE OF THE DATA*

	Supervised		Unsupervised
	Continuous Response	Categorical Response	No Response
Continuous predictors	Linear regression (6) k-Nearest neighbors (7) Neural nets (11) Ensembles (13)	k-Nearest neighbors (7) Logistic regression (10) Neural nets (11) Discriminant analysis (12) Ensembles (13)	Principal components (4) Collaborative filtering (14) Cluster analysis (15)
Categorical predictors	Linear regression (6) Regression trees (9) Neural nets (11) Ensembles (13)	Naïve Bayes (8) Classification trees (9) Logistic regression (10) Neural nets (11) Ensembles (13)	Association rules (14) Collaborative filtering (14)

***Numbers in parentheses indicate chapter number.**

Part IV includes eight chapters (Chapters 6–13), covering a variety of popular supervised learning methods (for classification and/or prediction). Within this part, the topics are generally organized according to the level of sophistication of the algorithms, their popularity, and ease of understanding. The final chapter introduces ensembles and combinations of methods.

Part V focuses on unsupervised mining of relationships. It presents association rules and collaborative filtering (Chapter 14) and cluster analysis (Chapter 15).

Part VI includes three chapters (Chapters 16–18), with the focus on forecasting time series. The first chapter covers general issues related to handling and understanding time series. The next two chapters present two popular forecasting approaches: regression-based forecasting and smoothing methods.

Part VII (Chapters 19–20) presents two broad data analytics topics: social network analysis and text mining. These methods apply data mining to specialized data structures: social networks and text.

Finally, part VIII includes a set of cases.

Although the topics in the book can be covered in the order of the chapters, each chapter stands alone. We advise, however, to read parts I–III before proceeding to chapters in parts IV–V. Similarly, Chapter 16 should precede other chapters in part VI.

Statistics.com PA 3 04 Fall 19
may not be redistributed

USING PYTHON AND JUPYTER NOTEBOOKS

Python is a powerful, general purpose programming language that can be used for many applications ranging from short scripts to enterprise applications. There is a large and growing number of free, open-source libraries and tools for scientific computing. For more information about Python and its use visit www.python.org.

To facilitate a hands-on data mining experience, this book uses Jupyter notebooks. The provide an interactive computational environment, in which you can easily combine code execution, rich text, and plots.

The Python language is widely used in academia and industry and had gained wide popularity for data mining and data analysis due to the availability of well maintained packages for data analysis, data visualization, and machine learning. It has extensive coverage of statistical and data mining techniques for classification, prediction, mining associations and text, forecasting, and data exploration and reduction. It offers a variety of supervised data mining tools: neural nets, classification and regression trees, k-nearest-neighbor classification, naive Bayes, logistic regression, linear regression, and discriminant analysis, all for predictive modeling. Python's packages also cover unsupervised algorithms: association rules, collaborative filtering, principal components analysis, k-means clustering, and hierarchical clustering, as well as visualization tools and data-handling utilities. Often, the same method is implemented in multiple packages, as we will discuss throughout the book. The illustrations, exercises, and cases in this book are written in relation to Python.

Download: To download Python and Jupyter notebooks, we recommend that you use *anaconda*. Visit www.anaconda.com and follow the instructions there.

Installation: Install anaconda and the anaconda-navigator. The anaconda-navigator can also be used to install and update individual packages.

Use: You can start Jupyter notebooks from the anaconda-navigator or from the command line.

For up-to-date and more comprehensive instructions see www.dataminingbook.com

Statistics.com PREDICT 2019
may not be reproduced

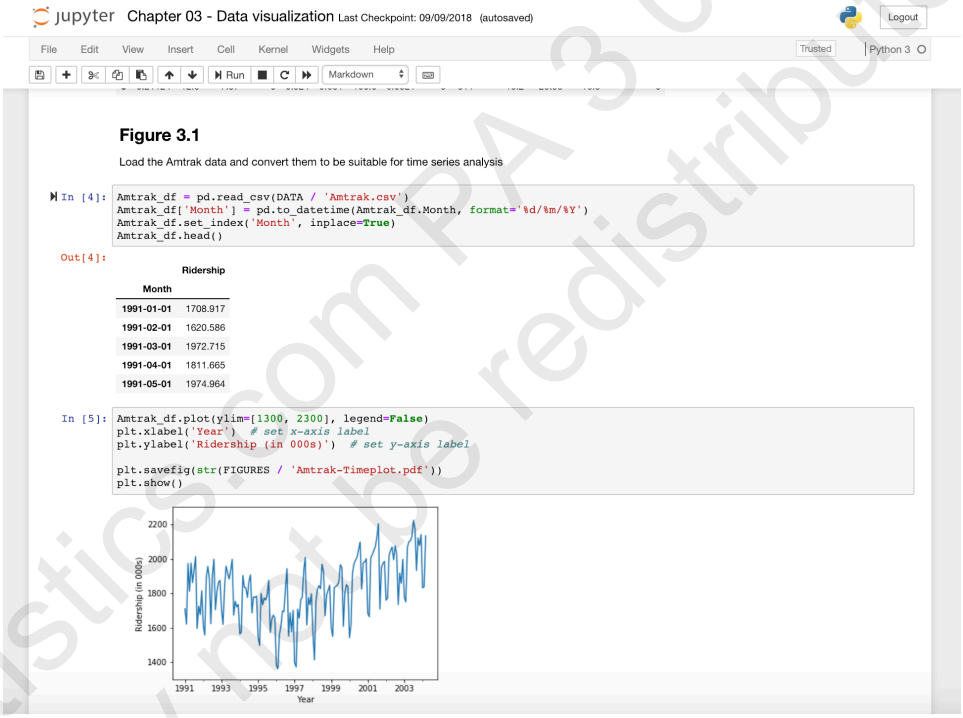

FIGURE 1.3 JUPYTER NOTEBOOK

Statistics.com PA 3 041219 may not be redistributed

CHAPTER 2

Overview of the Data Mining Process

In this chapter, we give an overview of the steps involved in data mining, starting from a clear goal definition and ending with model deployment. The general steps are shown schematically in Figure 2.1. We also discuss issues related to data collection, cleaning, and preprocessing. We introduce the notion of data partitioning, where methods are trained on a set of training data and then their performance is evaluated on a separate set of validation data, as well as explain how this practice helps avoid overfitting. Finally, we illustrate the steps of model building by applying them to data.

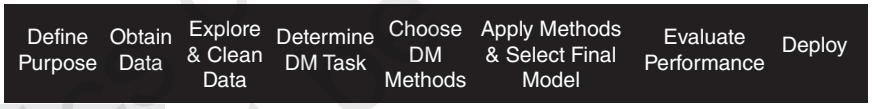

FIGURE 2.1 SCHEMATIC OF THE DATA MODELING PROCESS

2.1 INTRODUCTION

In Chapter 1, we saw some very general definitions of data mining. In this chapter, we introduce the variety of methods sometimes referred to as *data mining*. The core of this book focuses on what has come to be called *predictive analytics*, the tasks of classification and prediction as well as pattern discovery, which have become key elements of a "business analytics" function in most large firms. These terms are described and illustrated below.

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Statistics.com PA 3 041219 may not be redistributed

Not covered in this book to any great extent are two simpler database methods that are sometimes considered to be data mining techniques: (1) OLAP (online analytical processing) and (2) SQL (structured query language). OLAP and SQL searches on databases are descriptive in nature and are based on business rules set by the user (e.g., "find all credit card customers in a certain zip code with annual charges > $20,000, who own their home and who pay the entire amount of their monthly bill at least 95% of the time.") Although SQL queries are often used to obtain the data in data mining, they do not involve statistical modeling or automated algorithmic methods.

2.2 Core Ideas in Data Mining

Classification

Classification is perhaps the most basic form of data analysis. The recipient of an offer can respond or not respond. An applicant for a loan can repay on time, repay late, or declare bankruptcy. A credit card transaction can be normal or fraudulent. A packet of data traveling on a network can be benign or threatening. A bus in a fleet can be available for service or unavailable. The victim of an illness can be recovered, still be ill, or be deceased.

A common task in data mining is to examine data where the classification is unknown or will occur in the future, with the goal of predicting what that classification is or will be. Similar data where the classification is known are used to develop rules, which are then applied to the data with the unknown classification.

Prediction

Prediction is similar to classification, except that we are trying to predict the value of a numerical variable (e.g., amount of purchase) rather than a class (e.g., purchaser or nonpurchaser). Of course, in classification we are trying to predict a class, but the term *prediction* in this book refers to the prediction of the value of a continuous variable. (Sometimes in the data mining literature, the terms *estimation* and *regression* are used to refer to the prediction of the value of a continuous variable, and *prediction* may be used for both continuous and categorical data.)

Association Rules and Recommendation Systems

Large databases of customer transactions lend themselves naturally to the analysis of associations among items purchased, or "what goes with what." *Association rules*, or *affinity analysis*, is designed to find such general associations patterns between items in large databases. The rules can then be used in a variety of ways. For example, grocery stores can use such information for product placement.

Statistics.com PA 3 04 2019 may not be redistributed

They can use the rules for weekly promotional offers or for bundling products. Association rules derived from a hospital database on patients' symptoms during consecutive hospitalizations can help find "which symptom is followed by what other symptom" to help predict future symptoms for returning patients.

Online recommendation systems, such as those used on Amazon.com and Netflix.com, use *collaborative filtering*, a method that uses individual users' preferences and tastes given their historic purchase, rating, browsing, or any other measurable behavior indicative of preference, as well as other users' history. In contrast to *association rules* that generate rules general to an entire population, collaborative filtering generates "what goes with what" at the individual user level. Hence, collaborative filtering is used in many recommendation systems that aim to deliver personalized recommendations to users with a wide range of preferences.

Predictive Analytics

Classification, prediction, and to some extent, association rules and collaborative filtering constitute the analytical methods employed in *predictive analytics*. The term predictive analytics is sometimes used to also include data pattern identification methods such as clustering.

Data Reduction and Dimension Reduction

The performance of data mining algorithms is often improved when the number of variables is limited, and when large numbers of records can be grouped into homogeneous groups. For example, rather than dealing with thousands of product types, an analyst might wish to group them into a smaller number of groups and build separate models for each group. Or a marketer might want to classify customers into different "personas," and must therefore group customers into homogeneous groups to define the personas. This process of consolidating a large number of records (or cases) into a smaller set is termed *data reduction*. Methods for reducing the number of cases are often called *clustering*.

Reducing the number of variables is typically called *dimension reduction*. Dimension reduction is a common initial step before deploying data mining methods, intended to improve predictive power, manageability, and interpretability.

Data Exploration and Visualization

One of the earliest stages of engaging with a dataset is exploring it. Exploration is aimed at understanding the global landscape of the data, and detecting unusual values. Exploration is used for data cleaning and manipulation as well as for visual discovery and "hypothesis generation."

Statistics.com PA 3/04/2019 may not be redistributed

Methods for exploring data include looking at various data aggregations and summaries, both numerically and graphically. This includes looking at each variable separately as well as looking at relationships among variables. The purpose is to discover patterns and exceptions. Exploration by creating charts and dashboards is called *Data Visualization* or *Visual Analytics*. For numerical variables, we use histograms and boxplots to learn about the distribution of their values, to detect outliers (extreme observations), and to find other information that is relevant to the analysis task. Similarly, for categorical variables, we use bar charts. We can also look at scatter plots of pairs of numerical variables to learn about possible relationships, the type of relationship, and again, to detect outliers. Visualization can be greatly enhanced by adding features such as color and interactive navigation.

Supervised and Unsupervised Learning

A fundamental distinction among data mining techniques is between supervised and unsupervised methods. *Supervised learning algorithms* are those used in classification and prediction. We must have data available in which the value of the outcome of interest (e.g., purchase or no purchase) is known. Such data are also called "labeled data," since they contain the label (outcome value) for each record. These *training data* are the data from which the classification or prediction algorithm "learns," or is "trained," about the relationship between predictor variables and the outcome variable. Once the algorithm has learned from the training data, it is then applied to another sample of labeled data (the *validation data*) where the outcome is known but initially hidden, to see how well it does in comparison to other models. If many different models are being tried out, it is prudent to save a third sample, which also includes known outcomes (the *test data*) to use with the model finally selected to predict how well it will do. The model can then be used to classify or predict the outcome of interest in new cases where the outcome is unknown.

Simple linear regression is an example of a supervised learning algorithm (although rarely called that in the introductory statistics course where you probably first encountered it). The Y variable is the (known) outcome variable and the X variable is a predictor variable. A regression line is drawn to minimize the sum of squared deviations between the actual Y values and the values predicted by this line. The regression line can now be used to predict Y values for new values of X for which we do not know the Y value.

Unsupervised learning algorithms are those used where there is no outcome variable to predict or classify. Hence, there is no "learning" from cases where such an outcome variable is known. Association rules, dimension reduction methods, and clustering techniques are all unsupervised learning methods.

Statistics.com PA 3 04/12/19 may not be redistributed

Supervised and unsupervised methods are sometimes used in conjunction. For example, unsupervised clustering methods are used to separate loan applicants into several risk-level groups. Then, supervised algorithms are applied separately to each risk-level group for predicting propensity of loan default.

SUPERVISED LEARNING REQUIRES GOOD SUPERVISION

In some cases, the value of the outcome variable (the 'label') is known because it is an inherent component of the data. Web logs will show whether a person clicked on a link or not. Bank records will show whether a loan was paid on time or not. In other cases, the value of the known outcome must be supplied by a human labeling process to accumulate enough data to train a model. E-mail must be labeled as spam or legitimate, documents in legal discovery must be labeled as relevant or irrelevant. In either case, the data mining algorithm can be led astray if the quality of the supervision is poor.

Gene Weingarten reported in the January 5, 2014 *Washington Post* magazine how the strange phrase "defiantly recommend" is making its way into English via auto-correction. "Defiantly" is closer to the common misspelling *definatly* than is *definitely*, so Google.com, in the early days, offered it as a correction when users typed the misspelled word "definatly." In the ideal supervised learning model, humans guide the auto-correction process by rejecting *defiantly* and substituting *definitely*. Google's algorithm would then learn that this is the best first-choice correction of "definatly." The problem was that too many people were lazy, just accepting the first correction that Google presented. All these acceptances then cemented "defiantly" as the proper correction.

2.3 The Steps in Data Mining

This book focuses on understanding and using data mining algorithms (Steps 4 to 7 below). However, some of the most serious errors in analytics projects result from a poor understanding of the problem—an understanding that must be developed before we get into the details of algorithms to be used. Here is a list of steps to be taken in a typical data mining effort:

1. *Develop an understanding of the purpose of the data mining project.* How will the stakeholder use the results? Who will be affected by the results? Will the analysis be a one-shot effort or an ongoing procedure?
2. *Obtain the dataset to be used in the analysis.* This often involves sampling from a large database to capture records to be used in an analysis. How well this sample reflects the records of interest affects the ability of the data mining results to generalize to records outside of this sample. It may also involve pulling together data from different databases or sources.

Statistics.com FAS 04.12.19 may not be reproduced

The databases could be internal (e.g., past purchases made by customers) or external (credit ratings). While data mining deals with very large databases, usually the analysis to be done requires only thousands or tens of thousands of records.

3. *Explore, clean, and preprocess the data.* This step involves verifying that the data are in reasonable condition. How should missing data be handled? Are the values in a reasonable range, given what you would expect for each variable? Are there obvious outliers? The data are reviewed graphically: for example, a matrix of scatterplots showing the relationship of each variable with every other variable. We also need to ensure consistency in the definitions of fields, units of measurement, time periods, and so on. In this step, new variables are also typically created from existing ones. For example, "duration" can be computed from start and end dates.
4. *Reduce the data dimension, if necessary.* Dimension reduction can involve operations such as eliminating unneeded variables, transforming variables (e.g., turning "money spent" into "spent $>$ \$100" vs. "spent $\leq$ \$100"), and creating new variables (e.g., a variable that records whether at least one of several products was purchased). Make sure that you know what each variable means and whether it is sensible to include it in the model.
5. *Determine the data mining task.* (classification, prediction, clustering, etc.). This involves translating the general question or problem of Step 1 into a more specific data mining question.
6. *Partition the data (for supervised tasks).* If the task is supervised (classification or prediction), randomly partition the dataset into three parts: training, validation, and test datasets.
7. *Choose the data mining techniques to be used.* (regression, neural nets, hierarchical clustering, etc.).
8. *Use algorithms to perform the task.* This is typically an iterative process—trying multiple variants, and often using multiple variants of the same algorithm (choosing different variables or settings within the algorithm). Where appropriate, feedback from the algorithm's performance on validation data is used to refine the settings.
9. *Interpret the results of the algorithms.* This involves making a choice as to the best algorithm to deploy, and where possible, testing the final choice on the test data to get an idea as to how well it will perform. (Recall that each algorithm may also be tested on the validation data for tuning purposes; in this way, the validation data become a part of the fitting process and are likely to underestimate the error in the deployment of the model that is finally chosen.)

Statistics.com PA 5/04/2019 may not be redistributed

10. *Deploy the model.* This step involves integrating the model into operational systems and running it on real records to produce decisions or actions. For example, the model might be applied to a purchased list of possible customers, and the action might be "include in the mailing if the predicted amount of purchase is $> \$10$." A key step here is "scoring" the new records, or using the chosen model to predict the outcome value ("score") for each new record.

The foregoing steps encompass the steps in SEMMA, a methodology developed by the software company SAS:

Sample Take a sample from the dataset; partition into training, validation, and test datasets.

Explore Examine the dataset statistically and graphically.

Modify Transform the variables and impute missing values.

Model Fit predictive models (e.g., regression tree, neural network).

Assess Compare models using a validation dataset.

IBM SPSS Modeler (previously SPSS-Clementine) has a similar methodology, termed CRISP-DM (CRoss-Industry Standard Process for Data Mining). All these frameworks include the same main steps involved in predictive modeling.

2.4 Preliminary Steps

Organization of Datasets

Datasets are nearly always constructed and displayed so that variables are in columns and records are in rows. We will illustrate this with home values in West Roxbury, Boston, in 2014. 14 variables are recorded for over 5000 homes. The spreadsheet is organized so that each row represents a home—the first home's assessed value was \$344,200, its tax was \$4430, its size was 9965 ft^2, it was built in 1880, and so on. In supervised learning situations, one of these variables will be the outcome variable, typically listed in the first or last column (in this case it is TOTAL VALUE, in the first column).

Predicting Home Values in the West Roxbury Neighborhood

The Internet has revolutionized the real estate industry. Realtors now list houses and their prices on the web, and estimates of house and condominium prices have become widely available, even for units not on the market. At this time of

Statistics.com PA 3 041219 may not be redistributed

writing, Zillow (www.zillow.com) is the most popular online real estate information site in the United States[1], and in 2014 they purchased their major rival, Trulia. By 2015, Zillow had become the dominant platform for checking house prices and, as such, the dominant online advertising venue for realtors. What used to be a comfortable 6% commission structure for realtors, affording them a handsome surplus (and an oversupply of realtors), was being rapidly eroded by an increasing need to pay for advertising on Zillow. (This, in fact, is the key to Zillow's business model—redirecting the 6% commission away from realtors and to itself.)

Zillow gets much of the data for its "Zestimates" of home values directly from publicly available city housing data, used to estimate property values for tax assessment. A competitor seeking to get into the market would likely take the same approach. So might realtors seeking to develop an alternative to Zillow.

A simple approach would be a naive, model-less method—just use the assessed values as determined by the city. Those values, however, do not necessarily include all properties, and they might not include changes warranted by remodeling, additions, etc. Moreover, the assessment methods used by cities may not be transparent or always reflect true market values. However, the city property data can be used as a starting point to build a model, to which additional data (such as that collected by large realtors) can be added later.

Let's look at how Boston property assessment data, available from the city of Boston, might be used to predict home values. The data in *WestRoxbury.csv* includes information on single family owner-occupied homes in West Roxbury, a neighborhood in southwest Boston, MA, in 2014. The data include values for various predictor variables, and for an outcome—assessed home value ("total value"). This dataset has 14 variables and includes 5802 homes. A sample of the data[2] is shown in Table 2.2, and the "data dictionary" describing each variable[3] is in Table 2.1.

As we saw earlier, below the header row, each row in the data represents a home. For example, the first home was assessed at a total value of $344.2 thousand (TOTAL VALUE). Its tax bill was $4330. It has a lot size of 9965 square feet (ft^2), was built in the year 1880, has two floors, six rooms, and so on.

Loading and Looking at the Data in Python

The `pandas` package supports loading data in a large number of file formats. However, we will typically want to have the data available as a csv (comma

[1]Harney, K., "Zestimates may not be as right as you'd like", *Washington Post*, Feb. 7, 2015, p. T10.
[2]The data are a slightly cleaned version of the Property Assessment FY2014 data at https://data.boston.gov/dataset/property-assessment (accessed December 2017).
[3]The full data dictionary provided by the City of Boston is available at https://data.boston.gov/dataset/property-assessment; we have modified a few variable names.

Statistics.com PA 3 04 12 19 may not be distributed

TABLE 2.1 DESCRIPTION OF VARIABLES IN WEST ROXBURY (BOSTON) HOME VALUE DATASET

TOTAL VALUE	Total assessed value for property, in thousands of USD
TAX	Tax bill amount based on total assessed value multiplied by the tax rate, in USD
LOT SQ FT	Total lot size of parcel in square feet
YR BUILT	Year the property was built
GROSS AREA	Gross floor area
LIVING AREA	Total living area for residential properties (ft^2)
FLOORS	Number of floors
ROOMS	Total number of rooms
BEDROOMS	Total number of bedrooms
FULL BATH	Total number of full baths
HALF BATH	Total number of half baths
KITCHEN	Total number of kitchens
FIREPLACE	Total number of fireplaces
REMODEL	When the house was remodeled (Recent/Old/None)

separated values) file. If the data are in an xlsx (or xls) file, we can save that same file in Excel as a csv file: go to File > Save as > Save as type: CSV (Comma delimited) (*.csv) > Save.

Note: When dealing with .csv files in Excel, beware of two things:

- Opening a .csv file in Excel strips off leading 0's, which corrupts zipcode data.
- Saving a .csv file in Excel saves only the digits that are displayed; if you need precision to a certain number of decimals, you need to ensure they are displayed before saving.

Once we have Python and `pandas` installed on our machine and the *WestRoxbury.csv* file saved as a csv file, we can run the code in Table 2.3 to load the data into Python.

TABLE 2.2 FIRST 10 RECORDS IN THE WEST ROXBURY HOME VALUES DATASET

TOTAL VALUE	TAX	LOT SQ FT	YR BUILT	GROSS AREA	LIVING AREA	FLOORS	ROOMS	BED ROOMS	FULL BATH	HALF BATH	KIT CHEN	FIRE PLACE	REMODEL
344.2	4330	9965	1880	2436	1352	2	6	3	1	1	1	0	None
412.6	5190	6590	1945	3108	1976	2	10	4	2	1	1	0	Recent
330.1	4152	7500	1890	2294	1371	2	8	4	1	1	1	0	None
498.6	6272	13,773	1957	5032	2608	1	9	5	1	1	1	1	None
331.5	4170	5000	1910	2370	1438	2	7	3	2	0	1	0	None
337.4	4244	5142	1950	2124	1060	1	6	3	1	0	1	1	Old
359.4	4521	5000	1954	3220	1916	2	7	3	1	1	1	0	None
320.4	4030	10,000	1950	2208	1200	1	6	3	1	0	1	0	None
333.5	4195	6835	1958	2582	1092	1	5	3	1	0	1	1	Recent
409.4	5150	5093	1900	4818	2992	2	8	4	2	0	1	0	None

Statistics.com PA 3 041219 May not be redistributed

TABLE 2.3 **WORKING WITH FILES IN** `pandas`

To start, open Anaconda-Navigator and launch a 'jupyter' notebook. It opens a new browser window. Navigate to the directory where your csv file is saved and open a new Python notebook. You can change the name from 'Untitled' to a more descriptive title, e.g. WestRoxbury.
Paste the code into the input area and execute it using the *Run* button. The notebook will output the result of the last statement in each input field. If you like to see additional output use the *print* function. We will exclude it in most code samples to improve clarity.

code for loading and creating subsets from the data

```
# Import required packages
import pandas as pd

# Load data
housing_df = pd.read_csv('WestRoxbury.csv')
housing_df.shape  # find the dimension of data frame
housing_df.head()  # show the first five rows
print(housing_df)  # show all the data

# Rename columns: replace spaces with '_' to allow dot notation
housing_df = housing_df.rename(columns={'TOTAL_VALUE ': 'TOTAL_VALUE'}) # explicit
housing_df.columns = [s.strip().replace(' ', '_') for s in housing_df.columns] # all columns

# Practice showing the first four rows of the data
housing_df.loc[0:3]  # loc[a:b] gives rows a to b, inclusive
housing_df.iloc[0:4]  # iloc[a:b] gives rows a to b-1

# Different ways of showing the first 10 values in column TOTAL_VALUE
housing_df['TOTAL_VALUE'].iloc[0:10]
housing_df.iloc[0:10]['TOTAL_VALUE']
housing_df.iloc[0:10].TOTAL_VALUE  # use dot notation if the column name has no spaces

# Show the fifth row of the first 10 columns
housing_df.iloc[4][0:10]
housing_df.iloc[4, 0:10]
housing_df.iloc[4:5, 0:10]  # use a slice to return a data frame

# Use pd.concat to combine non-consecutive columns into a new data frame.
# The axis argument specifies the dimension along which the
# concatenation happens, 0=rows, 1=columns.
pd.concat([housing_df.iloc[4:6,0:2], housing_df.iloc[4:6,4:6]], axis=1)

# To specify a full column, use:
housing.iloc[:,0:1]
housing.TOTAL_VALUE
housing_df['TOTAL_VALUE'][0:10]  # show the first 10 rows of the first column

# Descriptive statistics
print('Number of rows ', len(housing_df['TOTAL_VALUE'])) # show length of first column
print('Mean of TOTAL_VALUE ', housing_df['TOTAL_VALUE'].mean()) # show mean of column
housing_df.describe() # show summary statistics for each column
```

Statistics.com LLC 3 04 2019
may not be redistributed

Data from a csv file is stored in `pandas` as a data frame (e.g., *housing_df*). If our csv file has column headers, these headers get automatically stored as the column names of our data. A data frame is the fundamental object which is particularly useful for data handling and manipulation. A data frame has rows and columns. The rows are the observations for each case (e.g., house), and the columns are the variables of interest (e.g., TOTAL VALUE, TAX). The code in Table 2.3 walks you through some basic steps you will want to perform prior to doing any analysis: finding the size and dimension of your data (number of rows and columns), viewing all the data, displaying only selected rows and columns, and computing summary statistics for variables of interest. Note that comments are preceded with the # symbol.

In Python, it is useful to use *slices* of data frames or lists. A *slice* returns an object usually containing a portion of a sequence, such as a subset of rows and columns from a data frame. For example, `TAX[0:4]` is a slice of variable TAX containing the first 5 records. Note that Python uses 0-indexing, which means that indices start at 0 and not at 1. `pandas` uses two methods[4] to access rows in a data frame: *loc* and *iloc*. The *loc* method is more general and allows accessing rows using labels. The *iloc* method on the other hand only allows using integer numbers. To specify a range of rows, use the slice notation, e.g. 0:9. In general, Python excludes the end index in the slice and the *iloc* method conforms with this convention. The *loc* method, however, includes it, so be careful when using these methods.

Python imports

In Table 2.3 we imported the Python package `pandas` to use it for data handling. We will use a few other packages in the remainder of this chapter. Use the following lines to import all the required packages.

import required functionality for this chapter

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
```

The abbreviations pd, np, and sm are commonly used in the data science community.

[4] *Method* and *function* are slightly different in object-oriented programming. For simplicity consider a method as a function that is closely associated with an object (e.g. a data frame) and has access to its data.

Statistics.com PA 3 04/12/19 may not be redistributed

Sampling from a Database

Typically, we perform data mining on less than the complete database. Data mining algorithms will have varying limitations on what they can handle in terms of the numbers of records and variables, limitations that may be specific to computing power and capacity as well as software limitations. Even within those limits, many algorithms will execute faster with smaller samples.

Accurate models can often be built with as few as several thousand records. Hence, we will want to sample a subset of records for model building. Table 2.4 provides code for sampling in `pandas`.

TABLE 2.4 SAMPLING IN `pandas`

code for sampling and over/under-sampling

```
# random sample of 5 observations
housing_df.sample(5)

# oversample houses with over 10 rooms
weights = [0.9 if rooms > 10 else 0.01 for rooms in housing_df.ROOMS]
housing_df.sample(5, weights=weights)
```

Oversampling Rare Events in Classification Tasks

If the event we are interested in classifying is rare, for example, customers purchasing a product in response to a mailing, or fraudulent credit card transactions, sampling a random subset of records may yield so few events (e.g., purchases) that we have little information on them. We would end up with lots of data on nonpurchasers and non-fraudulent transactions but little on which to base a model that distinguishes purchasers from nonpurchasers or fraudulent from non-fraudulent. In such cases, we would want our sampling procedure to overweight the rare class (purchasers or frauds) relative to the majority class (nonpurchasers, non-frauds) so that our sample would end up with a healthy complement of purchasers or frauds.

Assuring an adequate number of responder or "success" cases to train the model is just part of the picture. A more important factor is the costs of misclassification. Whenever the response rate is extremely low, we are likely to attach more importance to identifying a responder than to identifying a non-responder. In direct-response advertising (whether by traditional mail, e-mail, or web advertising), we may encounter only one or two responders for every hundred records—the value of finding such a customer far outweighs the costs of reaching him or her. In trying to identify fraudulent transactions, or customers

Statistics.com PA 3 04/2019 may not be redistributed

unlikely to repay debt, the costs of failing to find the fraud or the nonpaying customer are likely to exceed the cost of more detailed review of a legitimate transaction or customer.

If the costs of failing to locate responders are comparable to the costs of misidentifying responders as non-responders, our models would usually achieve highest overall accuracy if they identified everyone as a non-responder (or almost everyone, if it is easy to identify a few responders without catching many non-responders). In such a case, the misclassification rate is very low—equal to the rate of responders—but the model is of no value.

More generally, we want to train our model with the asymmetric costs in mind so that the algorithm will catch the more valuable responders, probably at the cost of "catching" and misclassifying more non-responders as responders than would be the case if we assume equal costs. This subject is discussed in detail in Chapter 5.

Statistics.com PA 3.04.219 may not be redistributed

Preprocessing and Cleaning the Data

Types of Variables There are several ways of classifying variables. Variables can be numerical or text (character/string). They can be continuous (able to assume any real numerical value, usually in a given range), integer (taking only integer values), categorical (assuming one of a limited number of values), or date. Categorical variables can be either coded as numerical $(1, 2, 3)$ or text (payments current, payments not current, bankrupt). Categorical variables can be unordered (called *nominal variables*) with categories such as North America, Europe, and Asia; or they can be ordered (called *ordinal variables*) with categories such as high value, low value, and nil value.

Continuous variables can be handled by most data mining routines with the exception of the naive Bayes classifier, which deals exclusively with categorical predictor variables. The machine learning roots of data mining grew out of problems with categorical outcomes; the roots of statistics lie in the analysis of continuous variables. Sometimes, it is desirable to convert continuous variables to categorical variables. This is done most typically in the case of outcome variables, where the numerical variable is mapped to a decision (e.g., credit scores above a certain threshold mean "grant credit," a medical test result above a certain threshold means "start treatment").

For the West Roxbury data, Table 2.5 presents some `pandas` statements to review the variables and determine what type (class) `pandas` thinks they are, and to determine the number of levels in a categorical variable.

Handling Categorical Variables Categorical variables can also be handled by most data mining routines, but often require special handling. If the categorical variable is ordered (age group, degree of creditworthiness, etc.), we can sometimes code the categories numerically (1, 2, 3, …) and treat the variable as if it were a continuous variable. The smaller the number of categories,

TABLE 2.5 REVIEWING VARIABLES IN pandas

code for reviewing variables

```
housing_df.columns  # print a list of variables

# REMODEL needs to be converted to a categorical variable
housing_df.REMODEL = housing_df.REMODEL.astype('category')
housing_df.REMODEL.cat.categories  # Show number of categories
housing_df.REMODEL.dtype  # Check type of converted variable
```

Partial Output

```
> housing_df.columns
Index(['TOTAL_VALUE', 'TAX', 'LOT_SQFT', 'YR_BUILT', 'GROSS_AREA',
       'LIVING_AREA', 'FLOORS', 'ROOMS', 'BEDROOMS', 'FULL_BATH', 'HALF_BATH',
       'KITCHEN', 'FIREPLACE', 'REMODEL'],
      dtype='object')

> housing_df.REMODEL.cat.categories
Index(['None', 'Old', 'Recent'], dtype='object')

> housing_df.REMODEL.dtype
category
```

and the less they represent equal increments of value, the more problematic this approach becomes, but it often works well enough.

Nominal categorical variables, however, often cannot be used as is. In many cases, they must be decomposed into a series of binary variables, called *dummy variables*. For example, a single categorical variable that can have possible values of "student," "unemployed," "employed," or "retired" would be split into four separate dummy variables:

Student—Yes/No
Unemployed—Yes/No
Employed—Yes/No
Retired—Yes/No

In many cases, only three of the dummy variables need to be used; if the values of three are known, the fourth is also known. For example, given that these four values are the only possible ones, we can know that if a person is neither student, unemployed, nor employed, he or she must be retired. In some routines (e.g., linear regression and logistic regression), you should not use all four variables—the redundant information will cause the algorithm to fail. Note, also, that typical methods of creating dummy variables will leave the original categorical variable intact; obviously you should not use both the original variable and

Statistics.com PA 3 041219 may not be redistributed

the dummies. The code to create binary dummies from all categorical variables in the West Roxbury data frame is given in Table 2.6 (here only REMODEL is categorical).

TABLE 2.6 CREATING DUMMY VARIABLES IN `pandas`

code for creating binary dummies (indicators)

```
# use drop_first=True to drop the first dummy variable
housing_df = pd.get_dummies(housing_df, prefix_sep='_', drop_first=True)
housing_df.columns
housing_df.loc[:, 'REMODEL_Old':'REMODEL_Recent'].head(5)
```

Partial Output

```
   REMODEL_Old  REMODEL_Recent
0            0               0
1            0               1
2            0               0
3            0               0
4            0               0
```

Variable Selection More is not necessarily better when it comes to selecting variables for a model. Other things being equal, parsimony, or compactness, is a desirable feature in a model. For one thing, the more variables we include and the more complex the model, the greater the number of records we will need to assess relationships among the variables. Fifteen records may suffice to give us a rough idea of the relationship between Y and a single predictor variable X. If we now want information about the relationship between Y and 15 predictor variables $X_1, \ldots, X_{15}$, 15 records will not be enough (each estimated relationship would have an average of only one record's worth of information, making the estimate very unreliable). In addition, models based on many variables are often less robust, as they require the collection of more variables in the future, are subject to more data quality and availability issues, and require more data cleaning and preprocessing.

How Many Variables and How Much Data? Statisticians give us procedures to learn with some precision how many records we would need to achieve a given degree of reliability with a given dataset and a given model. These are called "power calculations" and are intended to assure that an average population effect will be estimated with sufficient precision from a sample. Data miners' needs are usually different, because the focus is not on identifying an average effect but rather on predicting individual records. This purpose typically

Statistics.com PA 3 041219 may not be redistributed

requires larger samples than those used for statistical inference. A good rule of thumb is to have 10 records for every predictor variable. Another rule, used by Delmaster and Hancock (2001, p. 68) for classification procedures, is to have at least $6 \times m \times p$ records, where m is the number of outcome classes and p is the number of variables.

In general, compactness or parsimony is a desirable feature in a data mining model. Even when we start with a small number of variables, we often end up with many more after creating new variables (such as converting a categorical variable into a set of dummy variables). Data visualization and dimension reduction methods help reduce the number of variables so that redundancies and information overlap are reduced.

Even when we have an ample supply of data, there are good reasons to pay close attention to the variables that are included in a model. Someone with domain knowledge (i.e., knowledge of the business process and the data) should be consulted, as knowledge of what the variables represent is typically critical for building a good model and avoiding errors. For example, suppose we're trying to predict the total purchase amount spent by customers, and we have a few predictor columns that are coded $X_1, X_2, X_3, \ldots$, where we don't know what those codes mean. We might find that X_1 is an excellent predictor of the total amount spent. However, if we discover that X_1 is the amount spent on shipping, calculated as a percentage of the purchase amount, then obviously a model that uses shipping amount cannot be used to predict purchase amount, because the shipping amount is not known until the transaction is completed. Another example is if we are trying to predict loan default at the time a customer applies for a loan. If our dataset includes only information on approved loan applications, we will not have information about what distinguishes defaulters from non-defaulters among denied applicants. A model based on approved loans alone can therefore not be used to predict defaulting behavior at the time of loan application, but rather only once a loan is approved.

Outliers The more data we are dealing with, the greater the chance of encountering erroneous values resulting from measurement error, data-entry error, or the like. If the erroneous value is in the same range as the rest of the data, it may be harmless. If it is well outside the range of the rest of the data (a misplaced decimal, for example), it may have a substantial effect on some of the data mining procedures we plan to use.

Values that lie far away from the bulk of the data are called *outliers*. The term *far away* is deliberately left vague because what is or is not called an outlier is an arbitrary decision. Analysts use rules of thumb such as "anything over three standard deviations away from the mean is an outlier," but no statistical rule can tell us whether such an outlier is the result of an error. In this statistical sense, an outlier is not necessarily an invalid data point, it is just a distant one.

Statistics.com FAQ 04-12-19 may not be redistributed

The purpose of identifying outliers is usually to call attention to values that need further review. We might come up with an explanation looking at the data—in the case of a misplaced decimal, this is likely. We might have no explanation, but know that the value is wrong—a temperature of 178°F for a sick person. Or, we might conclude that the value is within the realm of possibility and leave it alone. All these are judgments best made by someone with *domain knowledge*, knowledge of the particular application being considered: direct mail, mortgage finance, and so on, as opposed to technical knowledge of statistical or data mining procedures. Statistical procedures can do little beyond identifying the record as something that needs review.

If manual review is feasible, some outliers may be identified and corrected. In any case, if the number of records with outliers is very small, they might be treated as missing data. How do we inspect for outliers? One technique is to sort the records by the first column (e.g., using the `pandas` method *sort_values()* such as `df.sort_values(by=['col1'])`), then review the data for very large or very small values in that column. Then repeat for each successive column. Another option is to examine the minimum and maximum values of each column using `pandas`'s *min()* and *max()* methods. For a more automated approach that considers each record as a unit, rather than each column in isolation, clustering techniques (see Chapter 14) could be used to identify clusters of one or a few records that are distant from others. Those records could then be examined.

Missing Values Typically, some records will contain missing values. If the number of records with missing values is small, those records might be omitted. However, if we have a large number of variables, even a small proportion of missing values can affect a lot of records. Even with only 30 variables, if only 5% of the values are missing (spread randomly and independently among cases and variables), almost 80% of the records would have to be omitted from the analysis. (The chance that a given record would escape having a missing value is $0.95^{30} = 0.215$.)

An alternative to omitting records with missing values is to replace the missing value with an imputed value, based on the other values for that variable across all records. For example, if among 30 variables, household income is missing for a particular record, we might substitute the mean household income across all records. Doing so does not, of course, add any information about how household income affects the outcome variable. It merely allows us to proceed with the analysis and not lose the information contained in this record for the other 29 variables. Note that using such a technique will understate the variability in a dataset. However, we can assess variability and the performance of our data mining technique using the validation data, and therefore this need not present a major problem. One option is to replace missing values using fairly simple substitutes (e.g., mean, median). More sophisticated procedures do exist—for

Statistics.com FAS 3041210 may not be redistributed

TABLE 2.7 MISSING DATA

code for imputing missing data with median

```
# To illustrate missing data procedures, we first convert a few entries for
# bedrooms to NA's. Then we impute these missing values using the median of the
# remaining values.
missingRows = housing_df.sample(10).index
housing_df.loc[missingRows, 'BEDROOMS'] = np.nan
print('Number of rows with valid BEDROOMS values after setting to NAN: ',
      housing_df['BEDROOMS'].count())

# remove rows with missing values
reduced_df = housing_df.dropna()
print('Number of rows after removing rows with missing values: ', len(reduced_df))

# replace the missing values using the median of the remaining values.
medianBedrooms = housing_df['BEDROOMS'].median()
housing_df.BEDROOMS = housing_df.BEDROOMS.fillna(value=medianBedrooms)
print('Number of rows with valid BEDROOMS values after filling NA values: ',
      housing_df['BEDROOMS'].count())
```

Output

```
Number of rows with valid BEDROOMS values after setting to NAN:  5782
Number of rows after removing rows with missing values:  5772
Number of rows with valid BEDROOMS values after filling NA values:  5802
```

example, using linear regression, based on other variables, to fill in the missing values. These methods have been elaborated mainly for analysis of medical and scientific studies, where each patient or subject record comes at great expense. In data mining, where data are typically plentiful, simpler methods usually suffice. Table 2.7 shows some Python code to illustrate the use of the median to replace missing values. Since the data are complete to begin with, the first step is an artificial one of creating some missing records for illustration purposes. The median is used for imputation, rather than the mean, to preserve the integer nature of the counts for bedrooms.

Some datasets contain variables that have a very large number of missing values. In other words, a measurement is missing for a large number of records. In that case, dropping records with missing values will lead to a large loss of data. Imputing the missing values might also be useless, as the imputations are based on a small number of existing records. An alternative is to examine the importance of the predictor. If it is not very crucial, it can be dropped. If it is important, perhaps a proxy variable with fewer missing values can be used instead. When such a predictor is deemed central, the best solution is to invest in obtaining the missing data.

Statistics.com LLC 3 04 19 may not be redistributed

TABLE 2.8 NORMALIZING AND RESCALING DATA

code for normalizing and rescaling a data frame

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
df = housing_df.copy()

# Normalizing a data frame

# pandas:
norm_df = (housing_df - housing_df.mean()) / housing_df.std()

# scikit-learn:
scaler = StandardScaler()
norm_df = pd.DataFrame(scaler.fit_transform(housing_df),
                       index=housing_df.index, columns=housing_df.columns)
# the result of the transformation is a numpy array, we convert it into a dataframe

# Rescaling a data frame

# pandas:
norm_df = (housing_df - housing_df.min()) / (housing_df.max() - housing_df.min())

# scikit-learn:
scaler = MinMaxScaler()
norm_df = pd.DataFrame(scaler.fit_transform(housing_df),
                       index=housing_df.index, columns=housing_df.columns)
```

Significant time may be required to deal with missing data, as not all situations are susceptible to automated solutions. In a messy dataset, for example, a "0" might mean two things: (1) the value is missing, or (2) the value is actually zero. In the credit industry, a "0" in the "past due" variable might mean a customer who is fully paid up, or a customer with no credit history at all—two very different situations. Human judgment may be required for individual cases or to determine a special rule to deal with the situation.

Normalizing (Standardizing) and Rescaling Data Some algorithms require that the data be normalized before the algorithm can be implemented effectively. To normalize a variable, we subtract the mean from each value and then divide by the standard deviation. This operation is also sometimes called *standardizing*. `pandas` has no custom method for this, however the operation can be easily performed using the methods *mean* and *std*; `(df - df.mean()) / df.std()`. In effect, we are expressing each value as the "number of standard deviations away from the mean," also called a *z-score*. An alternative is the class *StandardScaler()*, which is one of a number different transformers available in `scikit-learn`. You can use the methods *fit()* or *fit_transform()* to train the

Statistics.com PA3 041219 may not be redistributed

transformer on the training set and the method *transform()* to apply on the validation set. The result of the transformation is no longer a `pandas` dataframe, however you can convert it back into one easily. Table 2.8 demonstrates both approaches.

Normalizing is one way to bring all variables to the same scale. Another popular approach is rescaling each variable to a [0,1] scale. This is done by subtracting the minimum value and then dividing by the range. Subtracting the minimum shifts the variable origin to zero. Dividing by the range shrinks or expands the data to the range [0,1]. In `pandas`, use the expression `(df-df.min())/(df.max()-df.min())`. The corresponding `scikit-learn` transformer is *MinMaxScaler*.

To consider why normalizing or scaling to [0,1] might be necessary, consider the case of clustering. Clustering typically involves calculating a distance measure that reflects how far each record is from a cluster center or from other records. With multiple variables, different units will be used: days, dollars, counts, and so on. If the dollars are in the thousands and everything else is in the tens, the dollar variable will come to dominate the distance measure. Moreover, changing units from, say, days to hours or months, could alter the outcome completely.

2.5 Predictive Power and Overfitting

In supervised learning, a key question presents itself: How well will our prediction or classification model perform when we apply it to new data? We are particularly interested in comparing the performance of various models so that we can choose the one we think will do the best when it is implemented in practice. A key concept is to make sure that our chosen model generalizes beyond the dataset that we have at hand. To assure generalization, we use the concept of *data partitioning* and try to avoid *overfitting*. These two important concepts are described next.

Overfitting

The more variables we include in a model, the greater the risk of overfitting the particular data used for modeling. What is overfitting?

In Table 2.9, we show hypothetical data about advertising expenditures in one time period and sales in a subsequent time period. A scatter plot of the data is shown in Figure 2.2. We could connect up these points with a smooth but complicated function, one that interpolates all these data points perfectly and leaves no error (residuals). This can be seen in Figure 2.3. However, we can see that such a curve is unlikely to be accurate, or even useful, in predicting future

Statistics.com PA 3 04 2019 may not be redistributed

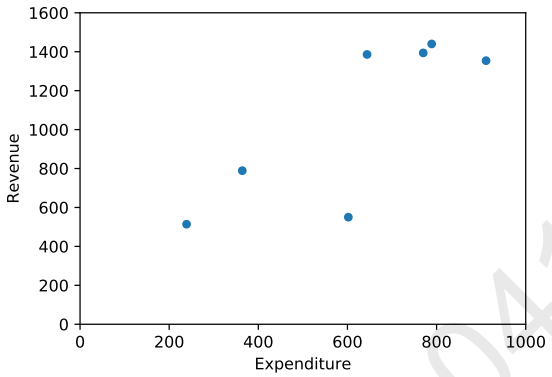

FIGURE 2.2 SCATTER PLOT FOR ADVERTISING AND SALES DATA

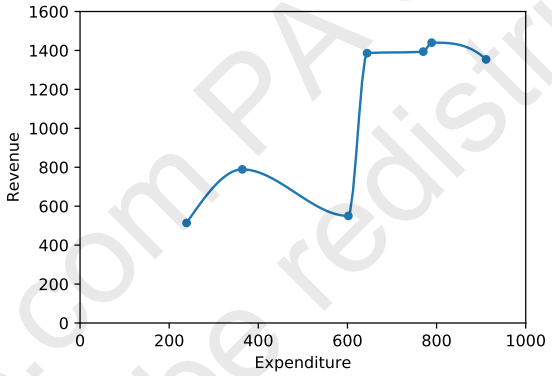

FIGURE 2.3 OVERFITTING: THIS FUNCTION FITS THE DATA WITH NO ERROR

sales on the basis of advertising expenditures. For instance, it is hard to believe that increasing expenditures from $400 to $500 will actually decrease revenue.

A basic purpose of building a model is to represent relationships among variables in such a way that this representation will do a good job of predicting future outcome values on the basis of future predictor values. Of course, we want the

TABLE 2.9

Advertising	Sales
239	514
364	789
602	550
644	1386
770	1394
789	1440
911	1354

Statistics.com PA 3 041219 may not be redistributed

model to do a good job of describing the data we have, but we are more interested in its performance with future data.

In the hypothetical advertising example, a simple straight line might do a better job than the complex function in terms of predicting future sales on the basis of advertising. Instead, we devised a complex function that fit the data perfectly, and in doing so, we overreached. We ended up modeling some variation in the data that is nothing more than chance variation. We mistreated the noise in the data as if it were a signal.

Similarly, we can add predictors to a model to sharpen its performance with the data at hand. Consider a database of 100 individuals, half of whom have contributed to a charitable cause. Information about income, family size, and zip code might do a fair job of predicting whether or not someone is a contributor. If we keep adding additional predictors, we can improve the performance of the model with the data at hand and reduce the misclassification error to a negligible level. However, this low error rate is misleading, because it probably includes spurious effects, which are specific to the 100 individuals, but not beyond that sample.

For example, one of the variables might be height. We have no basis in theory to suppose that tall people might contribute more or less to charity, but if there are several tall people in our sample and they just happened to contribute heavily to charity, our model might include a term for height—the taller you are, the more you will contribute. Of course, when the model is applied to additional data, it is likely that this will not turn out to be a good predictor.

If the dataset is not much larger than the number of predictor variables, it is very likely that a spurious relationship like this will creep into the model. Continuing with our charity example, with a small sample just a few of whom are tall, whatever the contribution level of tall people may be, the algorithm is tempted to attribute it to their being tall. If the dataset is very large relative to the number of predictors, this is less likely to occur. In such a case, each predictor must help predict the outcome for a large number of cases, so the job it does is much less dependent on just a few cases, which might be flukes.

Somewhat surprisingly, even if we know for a fact that a higher-degree curve is the appropriate model, if the model-fitting dataset is not large enough, a lower-degree function (that is not as likely to fit the noise) is likely to perform better in terms of predicting new values. Overfitting can also result from the application of many different models, from which the best performing model is selected.

Creation and Use of Data Partitions

At first glance, we might think it best to choose the model that did the best job of classifying or predicting the outcome variable of interest with the data at hand. However, when we use the same data both to develop the model and

Statistics.com PAS 04 12 09 may not be redistributed

to assess its performance, we introduce an "optimism" bias. This is because when we choose the model that works best with the data, this model's superior performance comes from two sources:

- A superior model
- Chance aspects of the data that happen to match the chosen model better than they match other models

The latter is a particularly serious problem with techniques (such as trees and neural nets) that do not impose linear or other structure on the data, and thus end up overfitting it.

To address the overfitting problem, we simply divide (partition) our data and develop our model using only one of the partitions. After we have a model, we try it out on another partition and see how it performs, which we can measure in several ways. In a classification model, we can count the proportion of held-back records that were misclassified. In a prediction model, we can measure the residuals (prediction errors) between the predicted values and the actual values. This evaluation approach in effect mimics the deployment scenario, where our model is applied to data that it hasn't "seen."

We typically deal with two or three partitions: a training set, a validation set, and sometimes an additional test set. Partitioning the data into training, validation, and test sets is done either randomly according to predetermined proportions or by specifying which records go into which partition according to some relevant variable (e.g., in time-series forecasting, the data are partitioned according to their chronological order). In most cases, the partitioning should be done randomly to minimize the chance of getting a biased partition. Note the varying nomenclature—the training partition is nearly always called "training" but the names for the other partitions can vary and overlap.

Training Partition The training partition, typically the largest partition, contains the data used to build the various models we are examining. The same training partition is generally used to develop multiple models.

Validation Partition The validation partition (sometimes called the *test partition*) is used to assess the predictive performance of each model so that you can compare models and choose the best one. In some algorithms (e.g., classification and regression trees, k-nearest neighbors), the validation partition may be used in an automated fashion to tune and improve the model.

Test Partition The test partition (sometimes called the *holdout* or *evaluation partition*) is used to assess the performance of the chosen model with new data.

Statistics.com PAS 041219 may not be redistributed

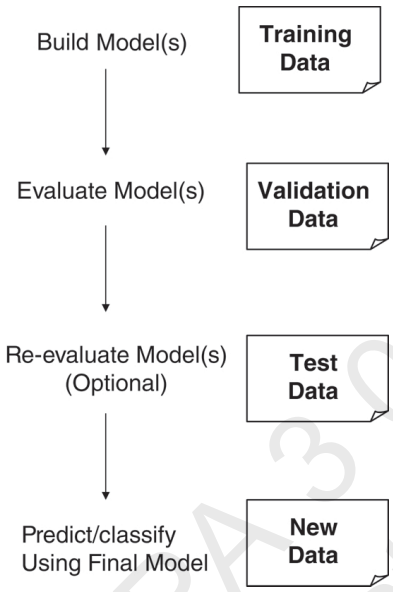

FIGURE 2.4 THREE DATA PARTITIONS AND THEIR ROLE IN THE DATA MINING PROCESS

Why have both a validation and a test partition? When we use the validation data to assess multiple models and then choose the model that performs best with the validation data, we again encounter another (lesser) facet of the overfitting problem—chance aspects of the validation data that happen to match the chosen model better than they match other models. In other words, by using the validation data to choose one of several models, the performance of the chosen model on the validation data will be overly optimistic.

The random features of the validation data that enhance the apparent performance of the chosen model will probably not be present in new data to which the model is applied. Therefore, we may have overestimated the accuracy of our model. The more models we test, the more likely it is that one of them will be particularly effective in modeling the noise in the validation data. Applying the model to the test data, which it has not seen before, will provide an unbiased estimate of how well the model will perform with new data. Figure 2.4 shows the three data partitions and their use in the data mining process. When we are concerned mainly with finding the best model and less with exactly how well it will do, we might use only training and validation partitions. Table 2.10 shows Python code to partition the West Roxbury data into two sets (training and validation) or into three sets (training, validation, and test). This is done by first drawing a random sample of records into the training set, then assigning the

Statistics.com PA 3 041219 may not be redistributed

TABLE 2.10 DATA PARTITIONING IN PYTHON

code for partitioning the West Roxbury data into training, validation (and test) sets

```
# random_state is set to a defined value to get the same partitions when re-running the code
# training (60%) and validation (40%)
trainData, validData = train_test_split(housing_df, test_size=0.40, random_state=1)

print('Training : ', trainData.shape)
print('Validation : ', validData.shape)
print()

# training (50%), validation (30%), and test (20%)
trainData, temp = train_test_split(housing_df, test_size=0.5, random_state=1)
validData, testData = train_test_split(temp, test_size=0.4, random_state=1)

print('Training   : ', trainData.shape)
print('Validation : ', validData.shape)
print('Test       : ', testData.shape)
```

Output

```
Training   :  (3481, 15)
Validation :  (2321, 15)

Training   :  (2901, 15)
Validation :  (1741, 15)
Test       :  (1160, 15)
```

remaining records as validation. In the case of three partitions, the validation records are chosen randomly from the data after excluding the records already sampled into the training set.

Note that with some algorithms, such as nearest-neighbor algorithms, records in the validation and test partitions, and in new data, are compared to records in the training data to find the nearest neighbor(s). As k-nearest neighbors is discussed in this book, the use of two partitions is an essential part of the classification or prediction process, not merely a way to improve or assess it. Nonetheless, we can still interpret the error in the validation data in the same way that we would interpret error from any other model.

Cross-Validation When the number of records in our sample is small, data partitioning might not be advisable as each partition will contain too few records for model building and performance evaluation. Furthermore, some data mining methods are sensitive to small changes in the training data, so that a different partitioning can lead to different results. An alternative to data partitioning is cross-validation, which is especially useful with small samples. Cross-validation, or *k-fold cross-validation*, is a procedure that starts with partitioning the data into

Statistics.com PA 3 041219 may not be redistributed

"folds," or non-overlapping subsamples. Often we choose $k = 5$ folds, meaning that the data are randomly partitioned into 5 equal parts, where each fold has 20% of the observations. A model is then fit k times. Each time, one of the folds is used as the validation set and the remaining $k - 1$ folds serve as the training set. The result is that each fold is used once as the validation set, thereby producing predictions for every observation in the dataset. We can then combine the model's predictions on each of the k validation sets in order to evaluate the overall performance of the model. In Python, cross-validation is achieved using the *cross_val_score()* or the more general *cross_validate* function, where argument `cv` determines the number of folds. Sometimes cross-validation is built into a data mining algorithm, with the results of the cross-validation used for choosing the algorithm's parameters (see, e.g., Chapter 9).

2.6 Building a Predictive Model

Let us go through the steps typical to many data mining tasks using a familiar procedure: multiple linear regression. This will help you understand the overall process before we begin tackling new algorithms.

Modeling Process

We now describe in detail the various model stages using the West Roxbury home values example.

1. *Determine the purpose.* Let's assume that the purpose of our data mining project is to predict the value of homes in West Roxbury for new records.
2. *Obtain the data.* We will use the 2014 West Roxbury housing data. The dataset in question is small enough that we do not need to sample from it—we can use it in its entirety.
3. *Explore, clean, and preprocess the data.* Let's look first at the description of the variables, also known as the "data dictionary," to be sure that we understand them all. These descriptions are available in Table 2.1. The variable names and descriptions in this dataset all seem fairly straightforward, but this is not always the case. Often, variable names are cryptic and their descriptions may be unclear or missing.

   It is useful to pause and think about what the variables mean and whether they should be included in the model. Consider the variable TAX. At first glance, we consider that the tax on a home is usually a function of its assessed value, so there is some circularity in the model—we want to predict a home's value using TAX as a predictor, yet TAX itself is determined by a home's value. TAX might be a very good predictor of home value in a numerical sense, but would it be useful if we

Statistics.com PA 3 04.12.20 may not be redistributed

TABLE 2.11 OUTLIER IN WEST ROXBURY DATA

FLOORS	ROOMS
15	8
2	10
1.5	6
1	6

wanted to apply our model to homes whose assessed value might not be known? For this reason, we will exclude TAX from the analysis.

It is also useful to check for outliers that might be errors. For example, suppose that the column FLOORS (number of floors) looked like the one in Table 2.11, after sorting the data in descending order based on floors. We can tell right away that the 15 is in error—it is unlikely that a home has 15 floors. Since all other values are between 1 and 2 the decimal was probably misplaced and the value should be 1.5.

Lastly, we create dummy variables for categorical variables. Here we have one categorical variable: REMODEL, which has three categories.

4. *Reduce the data dimension.* The West Roxbury dataset has been prepared for presentation with fairly low dimension—it has only 13 variables, and the single categorical variable considered has only three categories (and hence adds two dummy variables when used in a linear regression model). If we had many more variables, at this stage we might want to apply a variable reduction technique, such as condensing multiple categories into a smaller number, or applying principal components analysis to consolidate multiple similar numerical variables (e.g., LIVING AREA, ROOMS, BEDROOMS, BATH, HALF BATH) into a smaller number of variables.

5. *Determine the data mining task.* The specific task is to predict the value of TOTAL VALUE using the predictor variables. This is a supervised prediction task. For simplicity, we excluded several additional variables present in the original dataset, which have many categories (BLDG TYPE, ROOF TYPE, and EXT FIN). We therefore use all the numerical variables (except TAX) and the dummies created for the remaining categorical variables.

6. *Partition the data (for supervised tasks).* In this case we divide the data into two partitions: training and validation (see Table 2.10). The training partition is used to build the model, and the validation partition is used to see how well the model does when applied to new data. We need to specify the percent of the data used in each partition. *Note*: Although not used in our example, a test partition might also be used.

Statistics.com PA 3 041219 may not be redistributed

7. *Choose the technique.* In this case, it is multiple linear regression. Having divided the data into training and validation partitions, we can build a multiple linear regression model with the training data. We want to predict the value of a house in West Roxbury on the basis of all the other predictors (except TAX).

8. *Use the algorithm to perform the task.* In Python, we use the `scikit-learn` *LinearRegression* method to predict house value with the training data, then use the same model to predict values for the validation data. Chapter 6 on linear regression goes into more detail. Table 2.12 shows the predicted values for the first few records in the training data along with

TABLE 2.12 PREDICTIONS (FITTED VALUES) FOR A SAMPLE OF TRAINING DATA

code for fitting a regression model to training data (West Roxbury)

```
from sklearn.linear_model import LinearRegression

# data loading and preprocessing
housing_df = pd.read_csv('WestRoxbury.csv')
housing_df.columns = [s.strip().replace(' ', '_') for s in housing_df.columns]
housing_df = pd.get_dummies(housing_df, prefix_sep='_', drop_first=True)

# create list of predictors and outcome
excludeColumns = ('TOTAL_VALUE', 'TAX')
predictors = [s for s in housing_df.columns if s not in excludeColumns]
outcome = 'TOTAL_VALUE'

# partition data
X = housing_df[predictors]
y = housing_df[outcome]
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

model = LinearRegression()
model.fit(train_X, train_y)

train_pred = model.predict(train_X)
train_results = pd.DataFrame({
    'TOTAL_VALUE': train_y,
    'predicted': train_pred,
    'residual': train_y - train_pred
})
train_results.head()
```

Output

```
      TOTAL_VALUE   predicted   residual
2024        392.0  387.726258   4.273742
5140        476.3  430.785540  45.514460
5259        367.4  384.042952 -16.642952
421         350.3  369.005551 -18.705551
1401        348.1  314.725722  33.374278
```

Statistics.com PA 3 04/19 may not be redistributed

TABLE 2.13 PREDICTIONS FOR A SAMPLE OF VALIDATION DATA

code for applying the regression model to predict validation set (West Roxbury)

```
valid_pred = model.predict(valid_X)
valid_results = pd.DataFrame({
    'TOTAL_VALUE': valid_y,
    'predicted': valid_pred,
    'residual': valid_y - valid_pred
})
valid_results.head()
```

Output

```
      TOTAL_VALUE   predicted   residual
1822        462.0  406.946377  55.053623
1998        370.4  362.888928   7.511072
5126        407.4  390.287208  17.112792
808         316.1  382.470203 -66.370203
4034        393.2  434.334998 -41.134998
```

the actual values and the residuals (prediction errors). Note that the predicted values are often called the *fitted values*, since they are for the records to which the model was fit. The results for the validation data are shown in Table 2.13. The prediction errors for the training and validation data are compared in Table 2.14.

Prediction error can be aggregated in several ways. Five common measures are shown in Table 2.14. The first is *mean error (ME)*, simply the average of the residuals (errors). In both cases, it is quite small relative to the units of TOTAL VALUE, indicating that, on balance, predictions average about right—our predictions are "unbiased." Of course, this simply means that the positive and negative errors balance out. It tells us nothing about how large these errors are.

The *RMS error (RMSE)* (root-mean-squared error) is more informative of the error magnitude: it takes the square root of the average squared error, so it gives an idea of the typical error (whether positive or negative) in the same scale as that used for the original outcome variable. The RMS error for the validation data (42.7 thousand dollars), which the model is seeing for the first time in making these predictions, is in the same range as for the training data (43.0 thousand dollars), which were used in training the model. Normally, we expect the validation set error to be higher than for the training set. Here they are practically the same, within the expected variation, which is a good indication the the model is not overfitting the data. The other measures are discussed in Chapter 5.

Statistics.com PA 3 041219 may not be redistributed

TABLE 2.14 PREDICTION ERROR METRICS FOR TRAINING AND VALIDATION DATA (ERROR FIGURES ARE IN THOUSANDS OF $)

code for computing model evaluation metrics

```
# import the utility function regressionSummary
from dmba import regressionSummary

# training set
regressionSummary(train_results.TOTAL_VALUE, train_results.predicted)

# validation set
regressionSummary(valid_results.TOTAL_VALUE, valid_results.predicted)
```

Output

```
# training set
Regression statistics

                    Mean Error (ME) : -0.0000
       Root Mean Squared Error (RMSE) : 43.0306
            Mean Absolute Error (MAE) : 32.6042
          Mean Percentage Error (MPE) : -1.1116
Mean Absolute Percentage Error (MAPE) : 8.4886

# validation set
Regression statistics

                    Mean Error (ME) : -0.1463
       Root Mean Squared Error (RMSE) : 42.7292
            Mean Absolute Error (MAE) : 31.9663
          Mean Percentage Error (MPE) : -1.0884
Mean Absolute Percentage Error (MAPE) : 8.3283
```

Statistics.com PA 3 041219 may not be redistributed

9. *Interpret the results*. At this stage, we would typically try other prediction algorithms (e.g., regression trees) and see how they do error-wise. We might also try different "settings" on the various models (e.g., we could use the *best subsets* variable selection approach in multiple linear regression to choose a reduced set of variables that might perform better with the validation data). After choosing the best model—typically, the model with the lowest error on the validation data while also recognizing that "simpler is better"—we use that model to predict the output variable in fresh data. These steps are covered in more detail in the analysis of cases.

10. *Deploy the model*. After the best model is chosen, it is applied to new data to predict TOTAL VALUE for homes where this value is unknown. This was, of course, the original purpose. Predicting the output value for new records is called *scoring*. For predictive tasks, scoring produces predicted numerical values. For classification tasks, scoring produces classes and/or propensities. Table 2.15 shows an example of a data frame with three homes to be scored using our regression model. Note that all the required predictor columns are present, and the output column is absent.

2.7 Using Python for Data Mining on a Local Machine

An important aspect of the data mining process is that the heavy-duty analysis does not necessarily require a huge number of records. The dataset to be analyzed may have millions of records, of course, but in applying multiple linear regression or applying a classification tree, the use of a sample of 20,000 is likely to yield as accurate an answer as that obtained when using the entire dataset. The principle involved is the same as the principle behind polling: If sampled judiciously, 2000 voters can give an estimate of the entire population's opinion within one or two percentage points. (See "How Many Variables and How Much Data" in Section 2.4 for further discussion.)

Therefore, in most cases, the number of records required in each partition (training, validation, and test) can be accommodated within the memory limit allowed of your local machine.

When we apply Big Data analytics in Python, it might be useful to remove unused objects (function *del*) and call the garbage collection (function *gc.collect()*) afterwards. In general, this is not necessary.

Statistics.com FAST 3 04.12.19 may not be redistributed

TABLE 2.15 DATA FRAME WITH THREE RECORDS TO BE SCORED

```
new_data = pd.DataFrame({
    'LOT_SQFT': [4200, 6444, 5035],
    'YR_BUILT': [1960, 1940, 1925],
    'GROSS_AREA': [2670, 2886, 3264],
    'LIVING_AREA': [1710, 1474, 1523],
    'FLOORS': [2.0, 1.5, 1.9],
    'ROOMS': [10, 6, 6],
    'BEDROOMS': [4, 3, 2],
    'FULL_BATH': [1, 1, 1],
    'HALF_BATH': [1, 1, 0],
    'KITCHEN': [1, 1, 1],
    'FIREPLACE': [1, 1, 0],
    'REMODEL_Old': [0, 0, 0],
    'REMODEL_Recent': [0, 0, 1],
})
print(new_data)

print('Predictions: ', model.predict(new_data))
```

Output

```
   LOT_SQFT  YR_BUILT  GROSS_AREA  LIVING_AREA  FLOORS  ROOMS  BEDROOMS  \
0      4200      1960        2670         1710     2.0     10         4
1      6444      1940        2886         1474     1.5      6         3
2      5035      1925        3264         1523     1.9      6         2

   FULL_BATH  HALF_BATH  KITCHEN  FIREPLACE  REMODEL_Old  REMODEL_Recent
0          1          1        1          1            0               0
1          1          1        1          1            0               0
2          1          0        1          0            0               1
Predictions:  [384.47210285 378.06696706 386.01773842]
```

2.8 AUTOMATING DATA MINING SOLUTIONS

In most supervised data mining applications, the goal is not a static, one-time analysis of a particular dataset. Rather, we want to develop a model that can be used on an ongoing basis to predict or classify new records. Our initial analysis will be in prototype mode, while we explore and define the problem and test different models. We will follow all the steps outlined earlier in this chapter.

At the end of that process, we will typically want our chosen model to be deployed in automated fashion. For example, the US Internal Revenue Service (IRS) receives several hundred million tax returns per year—it does not want to have to pull each tax return out into an Excel sheet or other environment separate from its main database to determine the predicted probability that the return is fraudulent. Rather, it would prefer that determination to be made as part of the normal tax filing environment and process. Music streaming services, such as Pandora or Spotify, need to determine "recommendations" for next songs

Statistics.com PA 3 041219 may not be redistributed

quickly for each of millions of users; there is no time to extract the data for manual analysis.

In practice, this is done by building the chosen algorithm into the computational setting in which the rest of the process lies. A tax return is entered directly into the IRS system by a tax preparer, a predictive algorithm is immediately applied to the new data in the IRS system, and a predicted classification is decided by the algorithm. Business rules would then determine what happens with that classification. In the IRS case, the rule might be "if no predicted fraud, continue routine processing; if fraud is predicted, alert an examiner for possible audit."

This flow of the tax return from data entry, into the IRS system, through a predictive algorithm, then back out to a human user is an example of a "data pipeline." The different components of the system communicate with one another via Application Programming Interfaces (APIs) that establish locally valid rules for transmitting data and associated communications. An API for a data mining algorithm would establish the required elements for a predictive algorithm to work—the exact predictor variables, their order, data formats, etc. It would also establish the requirements for communicating the results of the algorithm. Algorithms to be used in an automated data pipeline will need to be compliant with the rules of the APIs where they operate.

Finally, once the computational environment is set and functioning, the data miner's work is not done. The environment in which a model operates is typically dynamic, and predictive models often have a short shelf life—one leading consultant finds they rarely continue to function effectively for more than a year. So, even in a fully deployed state, models must be periodically checked and re-evaluated. Once performance flags, it is time to return to prototype mode and see if a new model can be developed.

In this book, our focus will be on the prototyping phase—all the steps that go into properly defining the model and developing and selecting a model. You should be aware, though, that most of the actual work of implementing a data mining model lies in the automated deployment phase. Much of this work is not in the analytic domain; rather, it lies in the domains of databases and computer science, to assure that detailed nuts and bolts of an automated dataflow all work properly.

2.9 Ethical Practice in Data Mining[5]

Prior to the advent of internet-connected devices, the biggest source of big data was public interaction on the internet. Social media users, as well as shoppers and searchers on the internet, have made an implicit deal with the big companies that provide these services: users can take advantage of powerful search, shopping

Statistics.com LLC 9041219 may not be reproduced or distributed

and social interaction tools for free, and, in return, the companies get access to user data. Since the first edition of this book was published in 2007, ethical issues in data mining and data science have received increasing attention, and new rules and laws are emerging. As a data scientist, you must be aware of the rules and follow them, but you must also think about the implications and use of your work once it goes beyond the prototyping phase.

More and more news stories appear concerning illegal, fraudulent, unsavory or just controversial uses of data science. Many people are unaware just how much detailed data on their personal lives is collected, shared, and sold. A writer for *The Guardian* newspaper downloaded his own personal Facebook data and it came to over 600 megabytes - a vivid and detailed portrait of his life[7]. Any one of dozens of apps on your smartphone can collect a flow of your location information and keep tabs on you.

Financial and medical data have long been covered by both industry standards and government regulation to protect privacy and security. While academic research using human subjects' data in most developed countries has been strictly regulated, the collection, storage, and use of personal data in industry has historically faced much less regulatory scrutiny. But concern has reached beyond the basic issues of protecting against theft and unauthorized disclosure of personal information, to the data mining and analytics methods that underlie the harvest and use of such data. In credit scoring, for example, industry regulatory requirements (such as those under Basel II) require that any deployed credit scoring models be highly repeatable, transparent, and auditable (Saddiqi, 2017).

In 2018, the European Union came out, for the first time, with an EU-wide regulation called the General Data Protection Regulation (GDPR). The GDPR limits and restricts the use and storage of personal data by companies and organizations operating in the EU and abroad, insofar as these organizations "monitor the behavior" of or "offer goods or services" to EU-residing data subjects. The GDPR thereby has the potential to affect any organization processing the personal data of EU-based data subjects, regardless of where the processing occurs. "Processing" includes any operation on the data, including pre-processing and the use of data mining algorithms.

The story of Cambridge University Professor Alexander Kogan is a cautionary tale. Kogan helped develop a Facebook app, "This is Your Digital Life," which collected the answers to quiz questions, as well as user data from Facebook. The purported purpose was a research project on online personality. Although fewer than 270,000 people downloaded the app, it was able to access (via friend

[5]This section copyright ©2019 Datastats, LLC and Galit Shmueli. Used by permission.

[7]Dylan Curran, March 30, 2018, *The Guardian* online US edition, accessed Feb 1, 2019, "Are you ready? Here is all the data Facebook and Google have on you". www.theguardian.com/commentisfree/2018/mar/28/all-the-data-facebook-google-has-on-you-privacy

Statistics.com FA 3 04.12.19 may not be redistributed

connections) data on 87 million users. This feature was of great value to the political consulting company Cambridge Analytica, which used data from the app in its political targeting efforts, and ended up doing work for the Trump 2016 campaign, as well as the 2016 Brexit campaign in the UK.

Great controversy ensued: Facebook's CEO, Mark Zuckerberg was called to account before the U.S. Congress, and Cambridge Analytica was eventually forced into bankruptcy. In an interview with Lesley Stahl on the 60 Minutes television program, Kogan contended that he was an innocent researcher who had fallen into deep water, ethically:

> "You know, I was kinda acting, honestly, quite naively. I thought we were doing everything okay…. I ran this lab that studied happiness and kindness."

How did analytics play a role? Facebook's sophisticated algorithms on network relationships (Chapter 19) allowed the Kogan app to reach beyond its user base and get data from millions of users who had never used the app[8].

In the legal sphere, organizations must be aware of government, contractual, and industry "best-practices" regulations and obligations. But the analytics professional and data scientist must think beyond the codified rules, and think how the predictive technologies they are working on might ultimately be used. Here is just a sampling of some of the concerns that have arisen:

1. **Big Brother Watching:** Law enforcement and state security analysts can use personal demographic and location information for legitimate purposes, e.g. collecting information on associates of a known terrorist. But what is considered "legitimate" may be very different in Germany, the U.S., China, and North Korea. At the time of this writing, an active issue in the U.S. and Europe was fear that the Chinese company Huawei's control of new 5G network standards would compromise Western security. Predictive modeling (Chapters 6-13) and network analysis (Chapter 19) are widely used in surveillance, whether for good or ill. At the same time, companies are also using similar privacy-invading technologies for commercial gains. Google was recently fined 50 million Euro by French regulators for "[depriving] the users of essential guarantees regarding processing operations that can reveal important parts of their private life".[9]
2. **Automated Weapon Targeting and Use:** Armed drones play an increasing role in war, and image recognition is used to facilitate navigation of flight paths, as well as help to identify and suggest targets. Should this capability also be allowed to "pull the trigger," say, if a target appears to be a match to an intended target or class of targets? The leading actors

[8] www.cbsnews.com/news/aleksandr-kogan-the-link-between-cambridge-analytica-and-facebook-60-minutes/
[9] www.washingtonpost.com/world/europe/france-fines-google-nearly-57-million-for-first-major-violation-of-new-european-privacy-regime/2019/01/21/

Statistics.com PA 3 04 12 19
may not be reproduced

in automated targeting may say now that they draw a red line at turning decision-making over to the machine, but the technology has a dynamic of its own. Adversaries and other actors may not draw such a line. Moreover, in an arms race situation, one or more parties are likely to conclude that others will not draw such a line, and seek to proactively develop this capability. Automatic weapon targeting is just an extreme case of automated decision-making made possible by data mining analytics.

3. **Bias and Discrimination:** Automated decision making based on predictive models is now becoming pervasive in many aspects of human life, from credit card transaction alerts and airport security detentions, through college admissions, CV screening for employment, to predictive policing and recidivism scores used by judges. Cathy O'Neil describes multiple cases where such automated decision making becomes "weapons of math destruction," defined by opacity, scale, and damage (O'Neil, 2016). Predictive models in general facilitate these automated decisions; deep learning style neural networks are the standard tool for image and voice recognition. When algorithms are trained on data that already contain human bias, the algorithms learn the bias thereby perpetuating, expanding and magnifying it. The ProPublica group has been publishing articles about various such discrimination cases[11].
   Governments have already started trying to address this issue: New York City passed the U.S.'s first algorithmic accountability law in 2017, where a task force will monitor the fairness and validity of algorithms used by municipal agencies. In February 2019, legislators in the U.S. Congress held a hearing on an algorithmic accountability bill that would establish guidelines for procurement and use of automated decision systems in government "in order to protect consumers, improve transparency, and create more market predictability."[12]. As for company use of automated decision making, in the EU, the GDPR allows EU subjects to opt-out of automated decision making.

4. **Internet conspiracy theories, rumors and "lynch" mobs:** Before the internet, the growth and dissemination of false rumors and fake conspiracy theories was limited naturally: "broadcast" media had gatekeepers, and one-to-one communications, though unrestricted, were too slow to support much more than local gossip. Social media has not only removed the gatekeepers and expanded the reach of an individual, but also provided interfaces and algorithms that add fuel to the fire. The messaging application WhatsApp, via its message forwarding facility, was

[11] www.propublica.org/series/machine-bias
[12] www.fastcompany.com/90302465/washington-introduces-landmark-algorithmic-accountability-laws

Statistics.com PAS 04 12 19 may not be redistributed

instrumental in sparking instant lynch mobs in India in 2018, as false rumors about child abductors spread rapidly. Fake Facebook and Twitter accounts, most notably under Russian control, have helped create and spread divisive and destabilizing messaging in Western democracies with a goal of affecting election outcomes.

A key element in fostering the rapid viral spread of false and damaging messaging is the recommendation algorithms used in social media - these are trained to show you "news" that you are most likely to be interested in (see Chapter 14). And what interests people, and makes them click, "like," and forward, is the car wreck, not the free flow of traffic.

5. **Psychological manipulation and distraction:** A notable aspect of the Cambridge Analytica controversy was the company's claim that it could use "psychographic profiling" based on Facebook data to manipulate behavior. There is some evidence that such psychological manipulation is possible: Matz et al. (in their 2017 PNAS paper "Psychological targeting as an effective approach to digital mass persuasion") found "In three field experiments that reached over 3.5 million individuals with psychologically tailored advertising, we find that matching the content of persuasive appeals to individuals' psychological characteristics significantly altered their behavior as measured by clicks and purchases." But prior psychological profiles are not needed to manipulate behavior. Facebook advisers embedded with the U.S. presidential campaign of Donald Trump guided an extremely complex and continuous system of microtargeted experimentation to determine what display and engagement variables were most effective with specific voters.

   Predictive algorithms, and the statistical principles of experimentation, are central to these automated algorithmic efforts to affect individual behavior. At a more basic level, there is concern that these continuous engagement algorithms can contribute to "digital addiction" and a reduction in ability to concentrate. Paul Lewis, writing in *The Guardian*, says algorithms like these contribute to a state of "'continuous partial attention', severely limiting people's ability to focus."[13] Some of the most powerful critiques have come from those with inside knowledge. An early Facebook investor, Roger McNamee, writes in his book *Zucked*, that Facebook is "terrible for America." Chamath Palihapatiya[14], formerly Facebook's VP for user growth, now says Facebook is "destroying how society works."

[13] *The Guardian* online, posted October 6, 2017 www.theguardian.com/technology/2017/oct/05/smartphone-addiction-silicon-valley-dystopia

[14] Palihapatiya was quoted by Jared Gilmour in the Sacramento Bee online, posted Dec. 11, 2017 www.sacbee.com/news/nation-world/national/article189213899.html

Statistics.com PAS 041219 may not be redistributed

6. **Data brokers and unintended uses of personal data:** Reacting to a growing U.S. crisis of opioid addiction, some actuarial firms and data brokers are providing doctors with "opioid addiction risk scores" for their patients, to guide them in offering the patients appropriate medications at appropriate doses and durations, with appropriate instructions. It sounds constructive, but what happens when the data also gets to insurers (which it will), employers, and government agencies? For example, could an individual be denied a security clearance simply due to a high opiod addiction risk score?[15] Individuals themselves contribute to the supply of health data when they use health apps that track personal behavior and medical information. The ability to sell such data is often essential for the app developer - a review of apps that track menstrual cycles "found that most rely on the production and analysis of data for financial sustainability."[16]

Where does this leave the analytics professional and data scientist? Data scientists and analytics professionals must consider not just the powerful benefits that their methods can yield in a specific context, but also the harm they can do in other contexts. A good question to ask is "how might an individual, or a country, with malicious designs make use of this technology?" It is not sufficient just to observe the rules that are currently in place; as a data professional you must also think and judge for yourself. Your judgement must cover not just your current intentions and plans (which may be all to the good), but also future implications and possibilities.

DATA MINING SOFTWARE: THE STATE OF THE MARKET

by Herb Edelstein*

The data mining market has changed in some important ways since the last edition of this book. The most significant trends have been the increasing volume of information available and the growing use of the cloud for data storage and analytics. Data mining and analysis have evolved to meet these new demands.

The term "Big Data" reflects the surge in the amount and types of data collected. There is still an enormous amount of transactional data, data warehousing data, scientific data, and clickstream data. However, adding to the massive storage requirements of traditional data is the influx of information from unstructured sources (e.g., customer service calls and images), social media, and more

[15] www.politico.com/story/2019/02/03/health-risk-scores-opioid-abuse-1139978

[16] broadly.vice.com/en_us/article/8xe4yz/menstrual-app-period-tracker-data-cyber-security

Statistics.com PA 5/04/2019 may not be redistributed

recently the Internet of Things, which produces a flood of sensor data. The number of organizations collecting such data has greatly increased as virtually every business is expanding in these areas.

Rapid technological change has been an important factor. The price of data storage has dropped precipitously. At this writing, hard disk storage has fallen to under $50 per terabyte and solid state drives are under $200 per terabyte. Concomitant with the decrease in storage costs has been a dramatic increase in bandwidth at ever lower costs. This has enabled the spread of cloud-based computing. Cloud-based computing refers to using remote platforms for storage, data management, and now analysis. Because scaling up the hardware and software infrastructure for Big Data is so complex, many organizations are entrusting their data to outside vendors. The three largest cloud players (Amazon, IBM, and Microsoft) are each reporting annual revenues in excess of US$20 billion, according to *Forbes* magazine.

Managing this much data is a challenging task. While traditional relational DBMSs—such as Oracle, Microsoft's SQL Server, IBMs DB2, and SAPs Adaptive Server Enterprise (formerly Sybase)—are still among the leading data management tools, open source DBMSs, such as Oracles MySQL are becoming increasingly popular. In addition, nonrelational data storage is making headway in storing extremely large amounts of data. For example, Hadoop, an open source tool for managing large distributed database architectures, has become an important player in the cloud database space. However, Hadoop is a tool for the application developer community rather than end users. Consequently, many of the data mining analytics vendors such as SAS have built interfaces to Hadoop.

All the major database management system vendors offer data mining capabilities, usually integrated into their DBMS. Leading products include Microsoft SQL Server Analysis Services, Oracle Data Mining, and Teradata Warehouse Miner. The target user for embedded data mining is a database professional. Not surprisingly, these products take advantage of database functionality, including using the DBMS to transform variables, storing models in the database, and extending the data access language to include model-building and scoring the database. A few products also supply a separate graphical interface for building data mining models. Where the DBMS has parallel processing capabilities, embedded data mining tools will generally take advantage of it, resulting in greater performance. As with the data mining suites described below, these tools offer an assortment of algorithms. Not only does IBM have embedded analytics in DB2, but following its acquisition of SPSS, IBM has incorporated Clementine and SPSS into IBM Modeler.

There are still a large number of stand-alone data mining tools based on a single algorithm or on a collection of algorithms called a suite. Target users include both statisticians and business intelligence analysts. The leading suites include SAS Enterprise Miner, SAS JMP, IBM Modeler, Salford Systems SPM, Statistica, XLMiner, and RapidMiner. Suites are characterized by providing a wide range of functionality, frequently accessed via a graphical user interface designed to enhance model-building productivity. A popular approach for many of these GUIs is to provide a workflow interface in which the data mining steps and analysis are linked together.

Many suites have outstanding visualization tools and links to statistical packages that extend the range of tasks they can perform. They provide interactive data transformation tools as well as a procedural scripting language for more complex data transformations. The suite vendors are working to link their tools more closely to underlying DBMSs; for example, data transformations might be handled by the DBMS. Data mining models can be exported to be incorporated into the DBMS through generating SQL, procedural language code (e.g., C++ or Java), or a standardized data mining model language called Predictive Model Markup Language (PMML).

In contrast to the general-purpose suites, application-specific tools are intended for particular analytic applications such as credit scoring, customer retention, and product marketing. Their focus may be further sharpened to address the needs of specialized markets such as mortgage lending or financial services. The target user is an analyst with expertise in the application domain. Therefore the interfaces, the algorithms, and even the terminology are customized for that particular industry, application, or customer. While less flexible than general-purpose tools, they offer the advantage of already incorporating domain knowledge into the product design, and can provide very good solutions with less effort. Data mining companies including SAS, IBM, and RapidMiner offer vertical market tools, as do industry specialists such as Fair Isaac. Other companies, such as Domo, are focusing on creating dashboards with analytics and visualizations for business intelligence.

Another technological shift has occurred with the spread of open source model building tools and open core tools. A somewhat simplified view of open source software is that the source code for the tool is available at no charge to the community of users and can be modified or enhanced by them. These enhancements are submitted to the originator or copyright holder, who can add them to the base package. Open core is a more recent approach in which a core set of functionality remains open and free, but there are proprietary extensions that are not free.

The most important open source statistical analysis software is R. R is descended from a Bell Labs program called S, which was commercialized as S+. Many data mining algorithms have been added to R, along with a plethora of statistics, data management tools, and visualization tools. Because it is essentially a programming language, R has enormous flexibility but a steeper learning curve than many of the GUI-based tools. Although there are some GUIs for R, the overwhelming majority of use is through programming.

Python is rapidly growing in popularity as a data analysis tool and for developing analytical applications. This is due in part because it is faster than R and easier to use than C++ or Java. Scikit-Learn is an open source Python library that provides a comprehensive suite of tools for data analysis that is widely used in the Python community. Orange is a visual interface for Python and Scikit-Learn using wrappers designed to simplify doing analysis. Developed at the University of Ljubljana in Slovenia, Orange is open-source software that uses a workflow interface to provide a greatly improved way to use Python for analysis.

As mentioned above, the cloud-computing vendors have moved into the data mining/predictive analytics business by offering AaaS (Analytics as a Service) and pricing their products on a transaction basis. These products are oriented more

Statistics.com 2019 may not be reprinted

toward application developers than business intelligence analysts. A big part of the attraction of mining data in the cloud is the ability to store and manage enormous amounts of data without requiring the expense and complexity of building an in-house capability. This can also enable a more rapid implementation of large distributed multi-user applications. Cloud based data can be used with non-cloud-based analytics if the vendors analytics do not meet the users needs.

Amazon has added Amazon Machine Learning to its Amazon Web Services (AWS), taking advantage of predictive modeling tools developed for Amazon's internal use. AWS supports both relational databases and Hadoop data management. Models cannot be exported, because they are intended to be applied to data stored on the Amazon cloud.

Google is very active in cloud analytics with its BigQuery and Prediction API. BigQuery allows the use of Google infrastructure to access large amounts of data using a SQL-like interface. The Prediction API can be accessed from a variety of languages including R and Python. It uses a variety of machine learning algorithms and automatically selects the best results. Unfortunately, this is not a transparent process. Furthermore, as with Amazon, models cannot be exported.

Microsoft is an active player in cloud analytics with its Azure Machine Learning Studio and Stream Analytics. Azure works with Hadoop clusters as well as with traditional relational databases. Azure ML offers a broad range of algorithms such as boosted trees and support vector machines as well as supporting R scripts and Python. Azure ML also supports a workflow interface making it more suitable for the nonprogrammer data scientist. The real-time analytics component is designed to allow streaming data from a variety of sources to be analyzed on the fly. Microsoft also acquired Revolution Analytics, a major player in the R analytics business, with a view to integrating Revolution's "R Enterprise" with SQL Server and Azure ML. R Enterprise includes extensions to R that eliminate memory limitations and take advantage of parallel processing.

One drawback of the cloud-based analytics tools is a relative lack of transparency and user control over the algorithms and their parameters. In some cases, the service will simply select a single model that is a black box to the user. Another drawback is that for the most part cloud-based tools are aimed at more sophisticated data scientists who are systems savvy.

Data science is playing a central role in enabling many organizations to optimize everything from production to marketing. New storage options and analytical tools promise even greater capabilities. The key is to select technology that's appropriate for an organization's unique goals and constraints. As always, human judgment is the most important component of a data mining solution.

This book's focus is on a comprehensive understanding of the different techniques and algorithms used in data mining, and less on the data management requirements of real-time deployment of data mining models.

***

*Herb Edelstein is president of Two Crows Consulting (www.twocrows.com), a leading data mining consulting firm near Washington, DC. He is an internationally recognized expert in data mining and data warehousing, a widely published author on these topics, and a popular speaker.
Copyright © 2019 Herb Edelstein.

Statistics.com FYI 3 04/2019 may not be copied

PROBLEMS

2.1 Assuming that data mining techniques are to be used in the following cases, identify whether the task required is supervised or unsupervised learning.

- **a.** Deciding whether to issue a loan to an applicant based on demographic and financial data (with reference to a database of similar data on prior customers).
- **b.** In an online bookstore, making recommendations to customers concerning additional items to buy based on the buying patterns in prior transactions.
- **c.** Identifying a network data packet as dangerous (virus, hacker attack) based on comparison to other packets whose threat status is known.
- **d.** Identifying segments of similar customers.
- **e.** Predicting whether a company will go bankrupt based on comparing its financial data to those of similar bankrupt and nonbankrupt firms.
- **f.** Estimating the repair time required for an aircraft based on a trouble ticket.
- **g.** Automated sorting of mail by zip code scanning.
- **h.** Printing of custom discount coupons at the conclusion of a grocery store checkout based on what you just bought and what others have bought previously.

2.2 Describe the difference in roles assumed by the validation partition and the test partition.

2.3 Consider the sample from a database of credit applicants in Table 2.16. Comment on the likelihood that it was sampled randomly, and whether it is likely to be a useful sample.

TABLE 2.16 SAMPLE FROM A DATABASE OF CREDIT APPLICATIONS

OBS	CHECK ACCT	DURATION	HISTORY	NEW CAR	USED CAR	FURNITURE	RADIO TV	EDUC	RETRAIN	AMOUNT	SAVE ACCT	RESPONSE
1	0	6	4	0	0	0	1	0	0	1169	4	1
8	1	36	2	0	1	0	0	0	0	6948	0	1
16	0	24	2	0	0	0	1	0	0	1282	1	0
24	1	12	4	0	1	0	0	0	0	1804	1	1
32	0	24	2	0	0	1	0	0	0	4020	0	1
40	1	9	2	0	0	0	1	0	0	458	0	1
48	0	6	2	0	1	0	0	0	0	1352	2	1
56	3	6	1	1	0	0	0	0	0	783	4	1
64	1	48	0	0	0	0	0	0	1	14421	0	0
72	3	7	4	0	0	0	1	0	0	730	4	1
80	1	30	2	0	0	1	0	0	0	3832	0	1
88	1	36	2	0	0	0	0	1	0	12612	1	0
96	1	54	0	0	0	0	0	0	1	15945	0	0
104	1	9	4	0	0	1	0	0	0	1919	0	1
112	2	15	2	0	0	0	0	1	0	392	0	1

2.4 Consider the sample from a bank database shown in Table 2.17; it was selected randomly from a larger database to be the training set. *Personal Loan* indicates whether a solicitation for a personal loan was accepted and is the response variable. A campaign is planned for a similar solicitation in the future and the bank is looking for a model that will identify likely responders. Examine the data carefully and indicate what your next step would be.

Statistics.com PA's 04/2019 may not be redistributed

TABLE 2.17 SAMPLE FROM A BANK DATABASE

OBS	AGE	EXPERIENCE	INCOME	ZIP CODE	FAMILY	CC AVG	EDUC	MORTGAGE	PERSONAL LOAN	SECURITIES ACCT
1	25	1	49	91107	4	1.6	1	0	0	1
4	35	9	100	94112	1	2.7	2	0	0	0
5	35	8	45	91330	4	1	2	0	0	0
9	35	10	81	90089	3	0.6	2	104	0	0
10	34	9	180	93023	1	8.9	3	0	1	0
12	29	5	45	90277	3	0.1	2	0	0	0
17	38	14	130	95010	4	4.7	3	134	1	0
18	42	18	81	94305	4	2.4	1	0	0	0
21	56	31	25	94015	4	0.9	2	111	0	0
26	43	19	29	94305	3	0.5	1	97	0	0
29	56	30	48	94539	1	2.2	3	0	0	0
30	38	13	119	94104	1	3.3	2	0	1	0
35	31	5	50	94035	4	1.8	3	0	0	0
36	48	24	81	92647	3	0.7	1	0	0	0
37	59	35	121	94720	1	2.9	1	0	0	0
38	51	25	71	95814	1	1.4	3	198	0	0
39	42	18	141	94114	3	5	3	0	1	1
41	57	32	84	92672	3	1.6	3	0	0	1

2.5 Using the concept of overfitting, explain why when a model is fit to training data, zero error with those data is not necessarily good.

2.6 In fitting a model to classify prospects as purchasers or nonpurchasers, a certain company drew the training data from internal data that include demographic and purchase information. Future data to be classified will be lists purchased from other sources, with demographic (but not purchase) data included. It was found that "refund issued" was a useful predictor in the training data. Why is this not an appropriate variable to include in the model?

2.7 A dataset has 1000 records and 50 variables with 5% of the values missing, spread randomly throughout the records and variables. An analyst decides to remove records with missing values. About how many records would you expect to be removed?

2.8 Normalize the data in Table 2.18, showing calculations.

TABLE 2.18

Age	Income ($)
25	49,000
56	156,000
65	99,000
32	192,000
41	39,000
49	57,000

2.9 Statistical distance between records can be measured in several ways. Consider Euclidean distance, measured as the square root of the sum of the squared differences. For the first two records in Table 2.18, it is

$$\sqrt{(25-56)^2+(49,000-156,000)^2}.$$

Statistics.com PA 3 04219 may not be redistributed

Can normalizing the data change which two records are farthest from each other in terms of Euclidean distance?

2.10 Two models are applied to a dataset that has been partitioned. Model A is considerably more accurate than model B on the training data, but slightly less accurate than model B on the validation data. Which model are you more likely to consider for final deployment?

2.11 The dataset *ToyotaCorolla.csv* contains data on used cars on sale during the late summer of 2004 in the Netherlands. It has 1436 records containing details on 38 attributes, including *Price*, *Age*, *Kilometers*, *HP*, and other specifications.

We plan to analyze the data using various data mining techniques described in future chapters. Prepare the data for use as follows:

a. The dataset has two categorical attributes, *Fuel Type* and *Metallic.* Describe how you would convert these to binary variables. Confirm this using `pandas` methods to transform categorical data into dummies.

b. Prepare the dataset (as factored into dummies) for data mining techniques of supervised learning by creating partitions in Python. Select all the variables and use default values for the random seed and partitioning percentages for training (50%), validation (30%), and test (20%) sets. Describe the roles that these partitions will play in modeling.

Statistics.com PA 3041210 may not be redistributed

CHAPTER 3

Data Visualization

In this chapter, we describe a set of plots that can be used to explore the multidimensional nature of a data set. We present basic plots (bar charts, line graphs, and scatter plots), distribution plots (boxplots and histograms), and different enhancements that expand the capabilities of these plots to visualize more information. We focus on how the different visualizations and operations can support data mining tasks, from supervised tasks (prediction, classification, and time series forecasting) to unsupervised tasks, and provide a few guidelines on specific visualizations to use with each data mining task. We also describe the advantages of interactive visualization over static plots. The chapter concludes with a presentation of specialized plots suitable for data with special structure (hierarchical, network, and geographical).

3.1 USES OF DATA VISUALIZATION[1]

The popular saying "a picture is worth a thousand words" refers to the ability to condense diffused verbal information into a compact and quickly understood graphical image. In the case of numbers, data visualization and numerical summarization provide us with both a powerful tool to explore data and an effective way to present results.

Where do visualization techniques fit into the data mining process, as described so far? They are primarily used in the preprocessing portion of the data mining process. Visualization supports data cleaning by finding incorrect values (e.g., patients whose age is 999 or −1), missing values, duplicate rows,

[1]This and subsequent sections in this chapter copyright ©2019 Statistics.com and Galit Shmueli. Used by permission.

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Statistics.com LLC 041219 may not be redistributed

columns with all the same value, and the like. Visualization techniques are also useful for variable derivation and selection: they can help determine which variables to include in the analysis and which might be redundant. They can also help with determining appropriate bin sizes, should binning of numerical variables be needed (e.g., a numerical outcome variable might need to be converted to a binary variable if a yes/no decision is required). They can also play a role in combining categories as part of the data reduction process. Finally, if the data have yet to be collected and collection is expensive (as with the Pandora project at its outset, see Chapter 7), visualization methods can help determine, using a sample, which variables and metrics are useful.

In this chapter, we focus on the use of graphical presentations for the purpose of *data exploration*, particularly with relation to predictive analytics. Although our focus is not on visualization for the purpose of data reporting, this chapter offers ideas as to the effectiveness of various graphical displays for the purpose of data presentation. These offer a wealth of useful presentations beyond tabular summaries and basic bar charts, which are currently the most popular form of data presentation in the business environment. For an excellent discussion of using charts to report business data, see Few (2004). In terms of reporting data mining results graphically, we describe common graphical displays elsewhere in the book, some of which are technique-specific [e.g., dendrograms for hierarchical clustering (Chapter 15), network charts for social network analysis (Chapter 19), and tree charts for classification and regression trees (Chapter 9)] while others are more general [e.g., receiver operating characteristic (ROC) curves and lift charts for classification (Chapter 5) and profile plots and heatmaps for clustering (Chapter 15)].

Note: The term "graph" can have two meanings in statistics. It can refer, particularly in popular usage, to any of a number of figures to represent data (e.g., line chart, bar plot, histogram, etc.). In a more technical use, it refers to the data structure and visualization in networks (see Chapter 19). Using the term "plot" for the visualizations we explore in this chapter avoids this confusion.

Data exploration is a mandatory initial step whether or not more formal analysis follows. Graphical exploration can support free-form exploration for the purpose of understanding the data structure, cleaning the data (e.g., identifying unexpected gaps or "illegal" values), identifying outliers, discovering initial patterns (e.g., correlations among variables and surprising clusters), and generating interesting questions. Graphical exploration can also be more focused, geared toward specific questions of interest. In the data mining context, a combination is needed: free-form exploration performed with the purpose of supporting a specific goal.

Graphical exploration can range from generating very basic plots to using operations such as filtering and zooming interactively to explore a set of interconnected visualizations that include advanced features such as color and

Statistics.com PA 3-04-2013 may not be distributed

multiple-panels. This chapter is not meant to be an exhaustive guidebook on visualization techniques, but instead discusses main principles and features that support data exploration in a data mining context. We start by describing varying levels of sophistication in terms of visualization, and show the advantages of different features and operations. Our discussion is from the perspective of how visualization supports the subsequent data mining goal. In particular, we distinguish between supervised and unsupervised learning; within supervised learning, we also further distinguish between classification (categorical outcome variable) and prediction (numerical outcome variable).

Python

There are a variety of Python libraries for data visualizations. The oldest, but also most flexible library is `matplotlib`. It is widely used and there are many resources available to get started and to find help. A number of other libraries are built around it and make the generation of plots often quicker.

`Seaborn` and `pandas` are two of the libraries that are wrappers around `matplotlib`. Both are very useful to create plots quickly. However, even if these are used, a good knowledge of `matplotlib` helps to have more control over the final plot.

The `ggplot` library is modeled after the equally named R package by Hadley Wickham. It is based on the "Grammar of Graphics," a term coined by Leland Wilkinson to define a system of plotting theory and nomenclature. Learning `ggplot` effectively means becoming familiar with this philosophy and technical language of plotting.

`Bokeh` is an interesting new development to create interactive plots, dashboards and data applications for web browers.

In this chapter, we mainly use `pandas` for data visualizations and cover possible customization of the plots using `matplotlib` commands. A few other packages are used for specialized plots: `seaborn` for heatmaps, `gmaps` and `cartopy` for map visualisations.

Statistics.com DA 3 04.12.19
may not be redistributed

import required functionality for this chapter

```
import os
import calendar
import numpy as np
import networkx as nx
import pandas as pd
from pandas.plotting import scatter_matrix, parallel_coordinates
import seaborn as sns
from sklearn import preprocessing
import matplotlib.pylab as plt
```

3.2 Data Examples

To illustrate data visualization, we use two datasets used in additional chapters in the book.

Example 1: Boston Housing Data

The Boston Housing data contain information on census tracts in Boston[2] for which several measurements are taken (e.g., crime rate, pupil/teacher ratio). It has 14 variables. A description of each variable is given in Table 3.1 and a sample of the first nine records is shown in Table 3.2. In addition to the original 13 variables, the dataset also contains the additional variable CAT.MEDV, which was created by categorizing median value (MEDV) into two categories: high and low. With *pandas* it is more convenient to have columns names that contain only characters, numbers, and the '_'. We therefore rename the column in the code examples to `CAT_MEDV`.

We consider three possible tasks:

1. A supervised predictive task, where the outcome variable of interest is the median value of a home in the tract (MEDV).
2. A supervised classification task, where the outcome variable of interest is the binary variable CAT.MEDV that indicates whether the home value is above or below $30,000.
3. An unsupervised task, where the goal is to cluster census tracts.

(MEDV and CAT.MEDV are not used together in any of the three cases).

[2]The Boston Housing dataset was originally published by Harrison and Rubinfeld in "Hedonic prices and the demand for clean air", *Journal of Environmental Economics & Management*, vol. 5, p. 81–102, 1978.

Statistics.com PA 3 041219 may not be redistributed

TABLE 3.1 DESCRIPTION OF VARIABLES IN BOSTON HOUSING DATASET

CRIM	Crime rate
ZN	Percentage of residential land zoned for lots over 25,000 ft^2
INDUS	Percentage of land occupied by nonretail business
CHAS	Does tract bound Charles River (= 1 if tract bounds river, = 0 otherwise)
NOX	Nitric oxide concentration (parts per 10 million)
RM	Average number of rooms per dwelling
AGE	Percentage of owner-occupied units built prior to 1940
DIS	Weighted distances to five Boston employment centers
RAD	Index of accessibility to radial highways
TAX	Full-value property tax rate per $10,000
PTRATIO	Pupil-to-teacher ratio by town
LSTAT	Percentage of lower status of the population
MEDV	Median value of owner-occupied homes in $1000s
CAT.MEDV	Is median value of owner-occupied homes in tract above $30,000 (CAT.MEDV = 1) or not (CAT.MEDV = 0)

TABLE 3.2 FIRST NINE RECORDS IN THE BOSTON HOUSING DATA

code for opening the Boston Housing file and viewing the first 9 records

```
housing_df = pd.read_csv('BostonHousing.csv')
# rename CAT. MEDV column for easier data handling
housing_df = housing_df.rename(columns={'CAT. MEDV': 'CAT_MEDV'})
housing_df.head(9)
```

Output

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	MEDV	CAT_MEDV
0	0.0063	18.0	2.31	0	0.538	6.575	65.2	4.090	1	296	15.3	4.98	24.0	0
1	0.0273	0.0	7.07	0	0.469	6.421	78.9	4.967	2	242	17.8	9.14	21.6	0
2	0.0273	0.0	7.07	0	0.469	7.185	61.1	4.967	2	242	17.8	4.03	34.7	1
3	0.0324	0.0	2.18	0	0.458	6.998	45.8	6.062	3	222	18.7	2.94	33.4	1
4	0.0691	0.0	2.18	0	0.458	7.147	54.2	6.062	3	222	18.7	5.33	36.2	1
5	0.0299	0.0	2.18	0	0.458	6.430	58.7	6.062	3	222	18.7	5.21	28.7	0
6	0.0883	12.5	7.87	0	0.524	6.012	66.6	5.561	5	311	15.2	12.43	22.9	0
7	0.1446	12.5	7.87	0	0.524	6.172	96.1	5.951	5	311	15.2	19.15	27.1	0
8	0.2112	12.5	7.87	0	0.524	5.631	100.0	6.082	5	311	15.2	29.93	16.5	0

Statistics.com PA 3 041219 May not be redistributed

Example 2: Ridership on Amtrak Trains

Amtrak, a US railway company, routinely collects data on ridership. Here we focus on forecasting future ridership using the series of monthly ridership between January 1991 and March 2004. The data and their source are described in Chapter 16. Hence our task here is (numerical) time series forecasting.

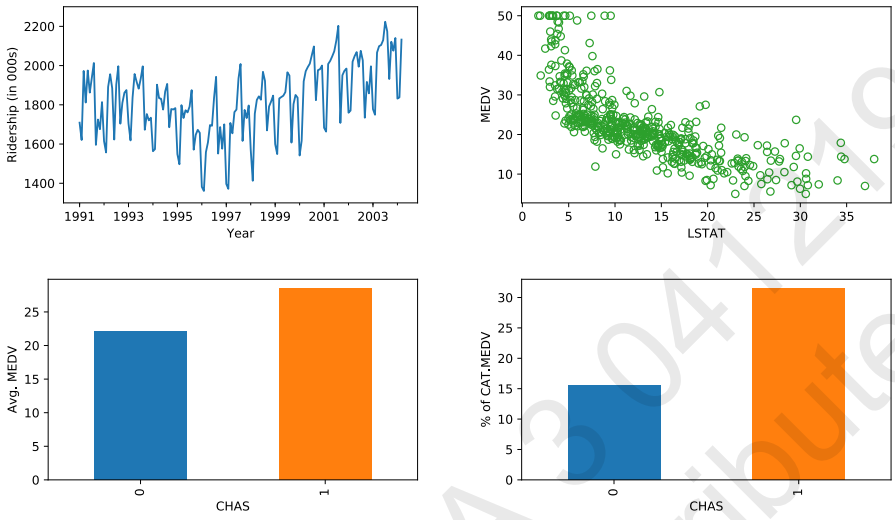

FIGURE 3.1 BASIC PLOTS: LINE GRAPH (TOP LEFT), SCATTER PLOT (TOP RIGHT), BAR CHART FOR NUMERICAL VARIABLE (BOTTOM LEFT), AND BAR CHART FOR CATEGORICAL VARIABLE (BOTTOM RIGHT)

3.3 BASIC CHARTS: BAR CHARTS, LINE GRAPHS, AND SCATTER PLOTS

The three most effective basic plots are bar charts, line graphs, and scatter plots. These plots are easy to create in Python using `pandas` and are the plots most commonly used in the current business world, in both data exploration and presentation (unfortunately, pie charts are also popular, although they are usually ineffective visualizations). Basic charts support data exploration by displaying one or two columns of data (variables) at a time. This is useful in the early stages of getting familiar with the data structure, the amount and types of variables, the volume and type of missing values, etc.

The nature of the data mining task and domain knowledge about the data will affect the use of basic charts in terms of the amount of time and effort allocated to different variables. In supervised learning, there will be more focus on the outcome variable. In scatter plots, the outcome variable is typically associated with the y-axis. In unsupervised learning (for the purpose of data reduction or clustering), basic plots that convey relationships (such as scatter plots) are preferred.

The top-left panel in Figure 3.1 displays a line chart for the time series of monthly railway passengers on Amtrak. Line graphs are used primarily for showing time series. The choice of time frame to plot, as well as the temporal scale, should depend on the horizon of the forecasting task and on the nature of the data.

Statistics.com DA 30412019 may not be redistributed

code for creating Figure 3.1

```
## Load the Amtrak data and convert them to be suitable for time series analysis
Amtrak_df = pd.read_csv('Amtrak.csv', squeeze=True)
Amtrak_df['Date'] = pd.to_datetime(Amtrak_df.Month, format='%d/%m/%Y')
ridership_ts = pd.Series(Amtrak_df.Ridership.values, index=Amtrak_df.Date)

## Boston housing data
housing_df = pd.read_csv('BostonHousing.csv')
housing_df = housing_df.rename(columns={'CAT. MEDV': 'CAT_MEDV'})
```

Pandas version

```
## line graph
ridership_ts.plot(ylim=[1300, 2300], legend=False)
plt.xlabel('Year')  # set x-axis label
plt.ylabel('Ridership (in 000s)')  # set y-axis label

## scatter plot with axes names
housing_df.plot.scatter(x='LSTAT', y='MEDV', legend=False)

## barchart of CHAS vs. mean MEDV
# compute mean MEDV per CHAS = (0, 1)
ax = housing_df.groupby('CHAS').mean().MEDV.plot(kind='bar')
ax.set_ylabel('Avg. MEDV')

## barchart of CHAS vs. CAT_MEDV
dataForPlot = housing_df.groupby('CHAS').mean()['CAT_MEDV'] * 100
ax = dataForPlot.plot(kind='bar', figsize=[5, 3])
ax.set_ylabel('% of CAT.MEDV')
```

matplotlib version

```
## line graph
plt.plot(ridership_ts.index, ridership_ts)
plt.xlabel('Year')  # set x-axis label
plt.ylabel('Ridership (in 000s)')  # set y-axis label

## Set the color of the points in the scatterplot and draw as open circles.
plt.scatter(housing_df.LSTAT, housing_df.MEDV, color='C2', facecolor='none')
plt.xlabel('LSTAT'); plt.ylabel('MEDV')

## barchart of CHAS vs. mean MEDV
# compute mean MEDV per CHAS = (0, 1)
dataForPlot = housing_df.groupby('CHAS').mean().MEDV
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C5', 'C1'])
ax.set_xticks([0, 1], False)
ax.set_xlabel('CHAS')
ax.set_ylabel('Avg. MEDV')

## barchart of CHAS vs. CAT.MEDV
dataForPlot = housing_df.groupby('CHAS').mean()['CAT_MEDV'] * 100
fig, ax = plt.subplots()
ax.bar(dataForPlot.index, dataForPlot, color=['C5', 'C1'])
ax.set_xticks([0, 1], False)
ax.set_xlabel('CHAS'); ax.set_ylabel('% of CAT.MEDV')
```

Statistics.com PA 3 041219
may not be redistributed

Bar charts are useful for comparing a single statistic (e.g., average, count, percentage) across groups. The height of the bar (or length in a horizontal display) represents the value of the statistic, and different bars correspond to different groups. Two examples are shown in the bottom panels in Figure 3.1. The left panel shows a bar chart for a numerical variable (MEDV) and the right panel shows a bar chart for a categorical variable (CAT.MEDV). In each, separate bars are used to denote homes in Boston that are near the Charles River vs. those that are not (thereby comparing the two categories of CHAS). The chart with the numerical output MEDV (bottom left) uses the average MEDV on the y-axis. This supports the predictive task: the numerical outcome is on the y-axis and the x-axis is used for a potential categorical predictor.[3] (Note that the x-axis on a bar chart must be used only for categorical variables, because the order of bars in a bar chart should be interchangeable.) For the classification task (bottom right), the y-axis indicates the percent of tracts with median value above \$30K and the x-axis is a binary variable indicating proximity to the Charles. This plot shows us that the tracts bordering the Charles are much more likely to have median values above \$30K.

The top-right panel in Figure 3.1 displays a scatter plot of MEDV vs. LSTAT. This is an important plot in the prediction task. Note that the output MEDV is again on the y-axis (and LSTAT on the x-axis is a potential predictor). Because both variables in a basic scatter plot must be numerical, it cannot be used to display the relation between CAT.MEDV and potential predictors for the classification task (but we can enhance it to do so—see Section 3.4). For unsupervised learning, this particular scatter plot helps study the association between two numerical variables in terms of information overlap as well as identifying clusters of observations.

All three basic plots highlight global information such as the overall level of ridership or MEDV, as well as changes over time (line chart), differences between subgroups (bar chart), and relationships between numerical variables (scatter plot).

Distribution Plots: Boxplots and Histograms

Before moving on to more sophisticated visualizations that enable multidimensional investigation, we note two important plots that are usually not considered "basic charts" but are very useful in statistical and data mining contexts. The *boxplot* and the *histogram* are two plots that display the entire distribution of a numerical variable. Although averages are very popular and useful summary statistics,

[3]We refer here to a bar chart with vertical bars. The same principles apply if using a bar chart with horizontal lines, except that the x-axis is now associated with the numerical variable and the y-axis with the categorical variable.

Statistics.com PA 3 04.2.15 may not be redistributed

there is usually much to be gained by looking at additional statistics such as the median and standard deviation of a variable, and even more so by examining the entire distribution. Whereas bar charts can only use a single aggregation, boxplots and histograms display the entire distribution of a numerical variable. Boxplots are also effective for comparing subgroups by generating side-by-side boxplots, or for looking at distributions over time by creating a series of boxplots.

Distribution plots are useful in supervised learning for determining potential data mining methods and variable transformations. For example, skewed numerical variables might warrant transformation (e.g., moving to a logarithmic scale) if used in methods that assume normality (e.g., linear regression, discriminant analysis).

A histogram represents the frequencies of all x values with a series of vertical connected bars. For example, in the left panel of Figure 3.2, there are over 150 tracts where the median value (MEDV) is between \$20K–\$25K.

A boxplot represents the variable being plotted on the y-axis (although the plot can potentially be turned in a 90 degrees angle, so that the boxes are parallel to the x-axis). In the right panel of Figure 3.2 there are two boxplots (called a side-by-side boxplot). The box encloses 50% of the data—for example, in the right-hand box half of the tracts have median values (MEDV) between \$20,000–\$33,000. The horizontal line inside the box represents the median (50th percentile). The top and bottom of the box represent the 75th and 25th percentiles, respectively. Lines extending above and below the box cover the rest of the data range; outliers may be depicted as points or circles. Sometimes the average is marked by a + (or similar) sign. Comparing the average and the median helps in assessing how skewed the data are. Boxplots are often arranged in a series with a different plot for each of the various values of a second variable, shown on the x-axis.

Because histograms and boxplots are geared toward numerical variables, their basic form is useful for *prediction* tasks. Boxplots can also support *unsupervised learning* by displaying relationships between a numerical variable (y-axis) and a categorical variable (x-axis). To illustrate these points, look again at Figure 3.2. The left panel shows a histogram of MEDV, revealing a skewed distribution. Transforming the output variable to log(MEDV) might improve results of a linear regression predictor.

The right panel in Figure 3.2 shows side-by-side boxplots comparing the distribution of MEDV for homes that border the Charles River (1) or not (0), similar to Figure 3.1. We see that not only is the average MEDV for river-bounding homes higher than the non-river-bounding homes, the entire distribution is higher (median, quartiles, min, and max). We can also see that all river-bounding homes have MEDV above \$10 thousand, unlike non-river-bounding homes. This information is useful for identifying the potential importance of

Statistics.com FAS 04-2019 May not be redistributed

this predictor (CHAS), and for choosing data mining methods that can capture the non-overlapping area between the two distributions (e.g., trees).

Boxplots and histograms applied to numerical variables can also provide directions for deriving new variables, for example, they can indicate how to bin a numerical variable (for example, binning a numerical outcome in order to use a naive Bayes classifier, or in the Boston Housing example, choosing the cutoff to convert MEDV to CAT.MEDV).

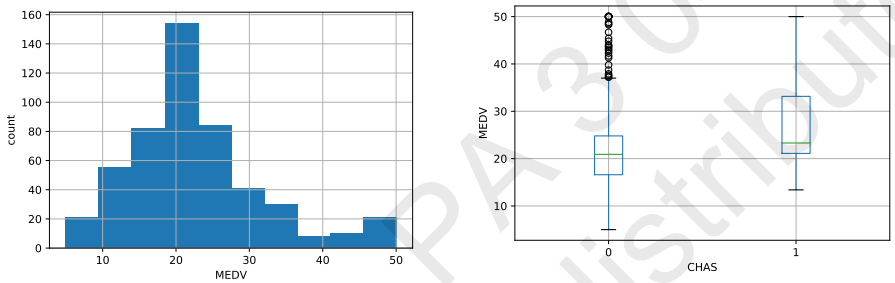

FIGURE 3.2 **DISTRIBUTION CHARTS FOR NUMERICAL VARIABLE MEDV. LEFT: HISTOGRAM, RIGHT: BOXPLOT**

code for creating Figure 3.2

```
## histogram of MEDV
ax = housing_df.MEDV.hist()
ax.set_xlabel('MEDV'); ax.set_ylabel('count')

# alternative plot with matplotlib
fig, ax = plt.subplots()
ax.hist(housing_df.MEDV)
ax.set_axisbelow(True)  # Show the grid lines behind the histogram
ax.grid(which='major', color='grey', linestyle='--')
ax.set_xlabel('MEDV'); ax.set_ylabel('count')
plt.show()

## boxplot of MEDV for different values of CHAS
ax = housing_df.boxplot(column='MEDV', by='CHAS')
ax.set_ylabel('MEDV')
plt.suptitle('')  # Suppress the titles
plt.title('')

# alternative plot with matplotlib
dataForPlot = [list(housing_df[housing_df.CHAS==0].MEDV),
               list(housing_df[housing_df.CHAS==1].MEDV)]
fig, ax = plt.subplots()
ax.boxplot(dataForPlot)
ax.set_xticks([1, 2], False)
ax.set_xticklabels([0, 1])
ax.set_xlabel('CHAS'); ax.set_ylabel('MEDV')
plt.show()
```

Statistics.com PA 3041219 may not be redistributed

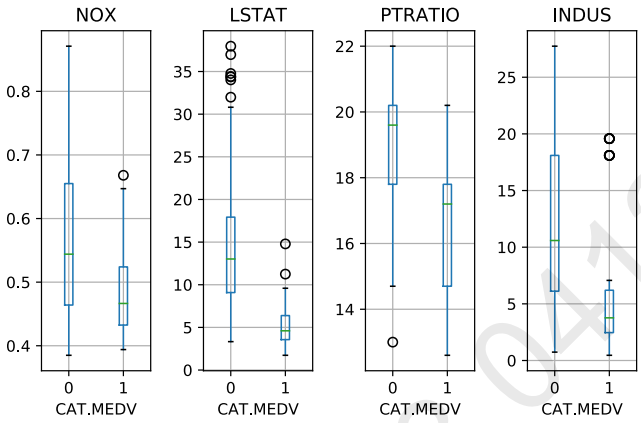

FIGURE 3.3 SIDE-BY-SIDE BOXPLOTS FOR EXPLORING THE CAT.MEDV OUTPUT VARIABLE BY DIFFERENT NUMERICAL PREDICTORS. IN A SIDE-BY-SIDE BOXPLOT, ONE AXIS IS USED FOR A CATEGORICAL VARIABLE, AND THE OTHER FOR A NUMERICAL VARIABLE. PLOTTING A CATEGORICAL OUTCOME VARIABLE AND A NUMERICAL PREDICTOR COMPARES THE PREDICTOR'S DISTRIBUTION ACROSS THE OUTCOME CATEGORIES. PLOTTING A NUMERICAL OUTCOME VARIABLE AND A CATEGORICAL PREDICTOR DISPLAYS THE DISTRIBUTION OF THE OUTCOME VARIABLE ACROSS DIFFERENT LEVELS OF THE PREDICTOR

code for creating Figure 3.3

```
## side-by-side boxplots
fig, axes = plt.subplots(nrows=1, ncols=4)
housing_df.boxplot(column='NOX', by='CAT_MEDV', ax=axes[0])
housing_df.boxplot(column='LSTAT', by='CAT_MEDV', ax=axes[1])
housing_df.boxplot(column='PTRATIO', by='CAT_MEDV', ax=axes[2])
housing_df.boxplot(column='INDUS', by='CAT_MEDV', ax=axes[3])
for ax in axes:
    ax.set_xlabel('CAT.MEDV')
plt.suptitle('')  # Suppress the overall title
plt.tight_layout()  # Increase the separation between the plots
```

Finally, side-by-side boxplots are useful in classification tasks for evaluating the potential of numerical predictors. This is done by using the x-axis for the categorical outcome and the y-axis for a numerical predictor. An example is shown in Figure 3.3, where we can see the effects of four numerical predictors on CAT.MEDV. The pairs that are most separated (e.g., PTRATIO and INDUS) indicate potentially useful predictors.

The main weakness of basic charts and distribution plots, in their basic form (that is, using position in relation to the axes to encode values), is that they can only display two variables and therefore cannot reveal high-dimensional information. Each of the basic charts has two dimensions, where each dimension is

Statistics.com 3041219 may not be reproduced

dedicated to a single variable. In data mining, the data are usually multivariate by nature, and the analytics are designed to capture and measure multivariate information. Visual exploration should therefore also incorporate this important aspect. In the next section, we describe how to extend basic charts (and distribution plots) to multidimensional data visualization by adding features, employing manipulations, and incorporating interactivity. We then present several specialized charts that are geared toward displaying special data structures (Section 3.5).

Heatmaps: Visualizing Correlations and Missing Values

A *heatmap* is a graphical display of numerical data where color is used to denote values. In a data mining context, heatmaps are especially useful for two purposes: for visualizing correlation tables and for visualizing missing values in the data. In both cases the information is conveyed in a two-dimensional table. A correlation table for p variables has p rows and p columns. A data table contains p columns (variables) and n rows (observations). If the number of rows is huge, then a subset can be used. In both cases, it is much easier and faster to scan the color-coding rather than the values. Note that heatmaps are useful when examining a large number of values, but they are not a replacement for more precise graphical display, such as bar charts, because color differences cannot be perceived accurately.

An example of a correlation table heatmap is shown in Figure 3.4, showing all the pairwise correlations between 13 variables (MEDV and 12 predictors). Darker shades correspond to stronger (positive or negative) correlation. It is easy to quickly spot the high and low correlations. The use of blue/red is used in this case to highlight positive vs. negative correlations.

In a missing value heatmap, rows correspond to records and columns to variables. We use a binary coding of the original dataset where 1 denotes a missing value and 0 otherwise. This new binary table is then colored such that only missing value cells (with value 1) are colored. Figure 3.5 shows an example of a missing value heatmap for a dataset on motor vehicle collisions[4]. The missing data heatmap helps visualize the level and amount of "missingness" in the dataset. Some patterns of "missingness" easily emerge: variables that are missing for nearly all observations, as well as clusters of rows that are missing many values. Variables with little missingness are also visible. This information can then be used for determining how to handle the missingness (e.g., dropping some variables, dropping some records, imputing, or via other techniques).

[4]The data were obtained from https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95

Statistics.com PA 3 04 1219 may not be reproduced

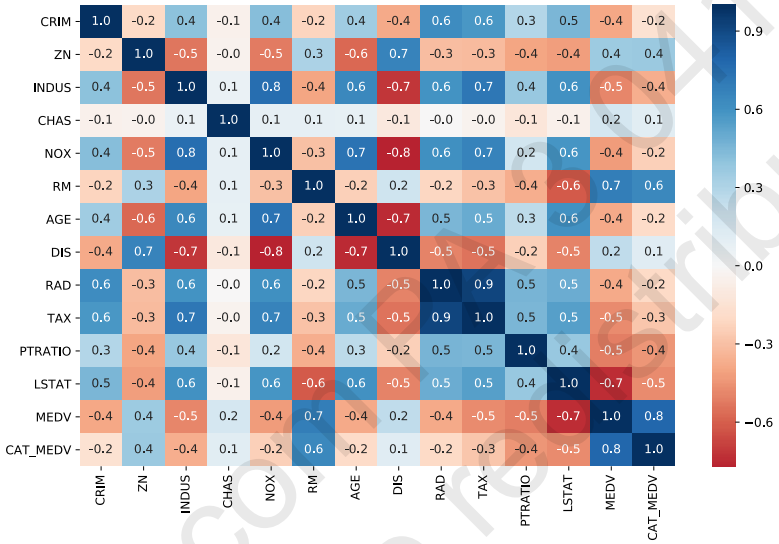

FIGURE 3.4 **HEATMAP OF A CORRELATION TABLE. DARKER VALUES DENOTE STRONGER CORRELATION. BLUE/RED REPRESENT POSITIVE/NEGATIVE VALUES, RESPECTIVELY.**

code for creating Figure 3.4

```
## simple heatmap of correlations (without values)
corr = housing_df.corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns)

# Change the colormap to a divergent scale and fix the range of the colormap
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, vmin=-1, vmax=1, cmap="RdBu")

# Include information about values (example demonstrate how to control the size of the plot
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax)
```

Statistics com Python 4219 may not be redistributed

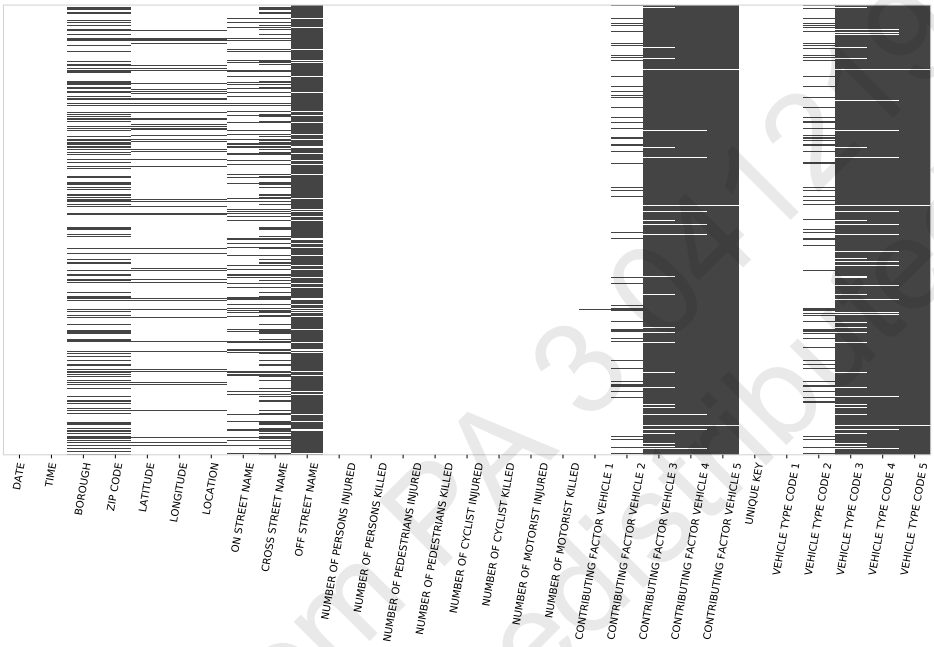

FIGURE 3.5 **HEATMAP OF MISSING VALUES IN A DATASET ON MOTOR VEHICLE COLLISIONS. GREY DENOTES MISSING VALUE**

code for generating a heatmap of missing values similar to Figure 3.5

```
df = pd.read_csv('NYPD_Motor_Vehicle_Collisions_1000.csv').sort_values(['DATE'])

# given a dataframe df create a copy of the array that is 0 if a field contains a value
# and 1 for NaN
naInfo = np.zeros(df.shape)
naInfo[df.isna().values] = 1
naInfo = pd.DataFrame(naInfo, columns=df.columns)

fig, ax = plt.subplots()
fig.set_size_inches(13, 9)
ax = sns.heatmap(naInfo, vmin=0, vmax=1, cmap=["white", "#666666"], cbar=False, ax=ax)
ax.set_yticks([])

# draw frame around figure
rect = plt.Rectangle((0, 0), naInfo.shape[1], naInfo.shape[0], linewidth=1,
                     edgecolor='lightgrey', facecolor='none')
rect = ax.add_patch(rect)
rect.set_clip_on(False)

plt.xticks(rotation=80)
```

Statistics.com MA 3941219 may not be reposted

3.4 Multidimensional Visualization

Basic plots can convey richer information with features such as color, size, and multiple panels, and by enabling operations such as rescaling, aggregation, and interactivity. These additions allow looking at more than one or two variables at a time. The beauty of these additions is their effectiveness in displaying complex information in an easily understandable way. Effective features are based on understanding how visual perception works (see Few (2009) for a discussion). The purpose is to make the information more understandable, not just to represent the data in higher dimensions (such as three-dimensional plots that are usually ineffective visualizations).

Adding Variables: Color, Size, Shape, Multiple Panels, and Animation

In order to include more variables in a plot, we must consider the type of variable to include. To represent additional categorical information, the best way is to use hue, shape, or multiple panels. For additional numerical information, we can use color intensity or size. Temporal information can be added via animation.

Incorporating additional categorical and/or numerical variables into the basic (and distribution) plots means that we can now use all of them for both prediction and classification tasks. For example, we mentioned earlier that a basic scatter plot cannot be used for studying the relationship between a categorical outcome and predictors (in the context of classification). However, a very effective plot for classification is a scatter plot of two numerical predictors color-coded by the categorical outcome variable. An example is shown in the top panel of Figure 3.6, with color denoting CAT.MEDV.

In the context of prediction, color-coding supports the exploration of the conditional relationship between the numerical outcome (on the y-axis) and a numerical predictor. Color-coded scatter plots then help assess the need for creating interaction terms (for example, is the relationship between MEDV and LSTAT different for homes near vs. away from the river?).

Color can also be used to include further categorical variables into a bar chart, as long as the number of categories is small. When the number of categories is large, a better alternative is to use multiple panels. Creating multiple panels (also called "trellising") is done by splitting the observations according to a categorical variable, and creating a separate plot (of the same type) for each category. An example is shown in the right-hand panel of Figure 3.6, where a bar chart of average MEDV by RAD is broken down into two panels by CHAS. We see that the average MEDV for different highway accessibility levels (RAD) behaves differently for homes near the river (lower panel) compared to homes away from the river (upper panel). This is especially salient for RAD = 1. We

Statistics.com RA 3 04-12-19 may not be redistributed

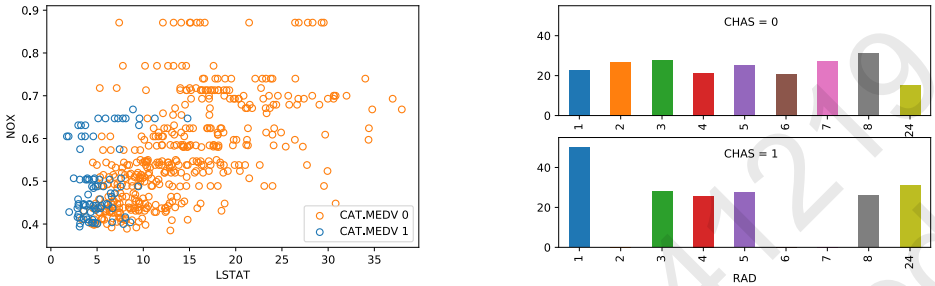

FIGURE 3.6 ADDING CATEGORICAL VARIABLES BY COLOR-CODING AND MULTIPLE PANELS. LEFT: SCATTER PLOT OF TWO NUMERICAL PREDICTORS, COLOR-CODED BY THE CATEGORICAL OUTCOME CAT.MEDV. RIGHT: BAR CHART OF MEDV BY TWO CATEGORICAL PREDICTORS (CHAS AND RAD), USING MULTIPLE PANELS FOR CHAS

code for creating Figure 3.6

```
# Color the points by the value of CAT.MEDV
housing_df.plot.scatter(x='LSTAT', y='NOX',
                        c=['C0' if c == 1 else 'C1' for c in housing_df.CAT_MEDV])

# Plot first the data points for CAT.MEDV of 0 and then of 1
# Setting color to 'none' gives open circles
_, ax = plt.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
    subset_df = housing_df[housing_df.CAT_MEDV == catValue]
    ax.scatter(subset_df.LSTAT, subset_df.NOX, color='none', edgecolor=color)
ax.set_xlabel('LSTAT')
ax.set_ylabel('NOX')
ax.legend(["CAT.MEDV 0", "CAT.MEDV 1"])
plt.show()

## panel plots
# compute mean MEDV per RAD and CHAS
dataForPlot_df = housing_df.groupby(['CHAS','RAD']).mean()['MEDV']
# We determine all possible RAD values to use as ticks
ticks = set(housing_df.RAD)
for i in range(2):
    for t in ticks.difference(dataForPlot_df[i].index):
        dataForPlot_df.loc[(i, t)] = 0
# reorder to rows, so that the index is sorted
dataForPlot_df = dataForPlot_df[sorted(dataForPlot_df.index)]

# Determine a common range for the y axis
yRange = [0, max(dataForPlot_df) * 1.1]

fig, axes = plt.subplots(nrows=2, ncols=1)
dataForPlot_df[0].plot.bar(x='RAD', ax=axes[0], ylim=yRange)
dataForPlot_df[1].plot.bar(x='RAD', ax=axes[1], ylim=yRange)
axes[0].annotate('CHAS = 0', xy=(3.5, 45))
axes[1].annotate('CHAS = 1', xy=(3.5, 45))
plt.show()
```

Statistics.com PA 3041219 may not be redistributed

also see that there are no near-river homes in RAD levels 2, 6, and 7. Such information might lead us to create an interaction term between RAD and CHAS, and to consider condensing some of the bins in RAD. All these explorations are useful for prediction and classification.

A special plot that uses scatter plots with multiple panels is the *scatter plot matrix*. In it, all pairwise scatter plots are shown in a single display. The panels in a matrix scatter plot are organized in a special way, such that each column and each row correspond to a variable, thereby the intersections create all the possible pairwise scatter plots. The scatter plot matrix is useful in unsupervised learning for studying the associations between numerical variables, detecting outliers and identifying clusters. For supervised learning, it can be used for examining pairwise relationships (and their nature) between predictors to support variable transformations and variable selection (see Correlation Analysis in Chapter 4). For prediction, it can also be used to depict the relationship of the outcome with the numerical predictors.

An example of a scatter plot matrix is shown in Figure 3.7, with MEDV and three predictors. Below the diagonal are the scatter plots. Variable name indicates the y-axis variable. For example, the plots in the bottom row all have MEDV on the y-axis (which allows studying the individual outcome–predictor relations). We can see different types of relationships from the different shapes (e.g., an exponential relationship between MEDV and LSTAT and a highly skewed relationship between CRIM and INDUS), which can indicate needed transformations. Along the diagonal, where just a single variable is involved, the frequency distribution for that variable is displayed. The scatterplots above the diagonal contain the correlation coefficients corresponding to the two variables.

Once hue is used, further categorical variables can be added via shape and multiple panels. However, one must proceed cautiously in adding multiple variables, as the display can become over-cluttered and then visual perception is lost.

Adding a numerical variable via size is useful especially in scatter plots (thereby creating "bubble plots"), because in a scatter plot, points represent individual observations. In plots that aggregate across observations (e.g., boxplots, histograms, bar charts), size and hue are not normally incorporated.

Finally, adding a temporal dimension to a plot to show how the information changes over time can be achieved via animation. A famous example is Rosling's animated scatter plots showing how world demographics changed over the years (www.gapminder.org). However, while animations of this type work for "statistical storytelling," they are not very effective for data exploration.

Manipulations: Rescaling, Aggregation and Hierarchies, Zooming, Filtering

Most of the time spent in data mining projects is spent in preprocessing. Typically, considerable effort is expended getting all the data in a format that can actually be used in the data mining software. Additional time is spent processing the

Statistics.com PA 04/2019
may not be distributed

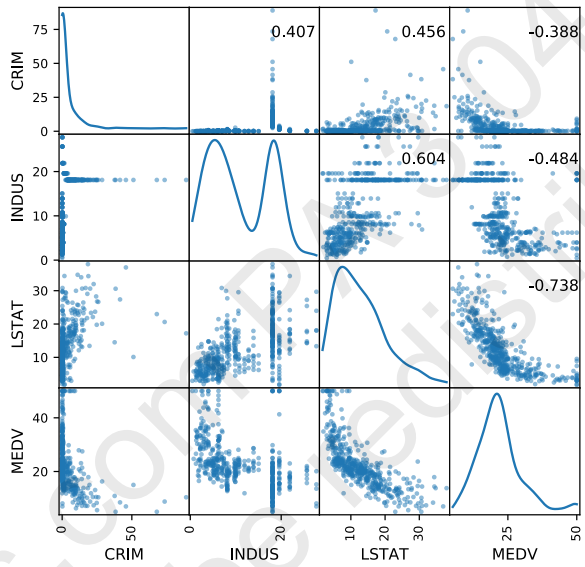

FIGURE 3.7 SCATTER PLOT MATRIX FOR MEDV AND THREE NUMERICAL PREDICTORS

code for creating Figure 3.7

```
# Display scatterplots between the different variables
# The diagonal shows the distribution for each variable
df = housing_df[['CRIM', 'INDUS', 'LSTAT', 'MEDV']]
axes = scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='kde')
corr = df.corr().as_matrix()
for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate('%.3f' %corr[i,j], (0.8, 0.8),
                        xycoords='axes fraction', ha='center', va='center')
plt.show()
```

Statistics.com Data 04219 may not be redistributed

data in ways that improve the performance of the data mining procedures. This preprocessing step in data mining includes variable transformation and derivation of new variables to help models perform more effectively. Transformations include changing the numeric scale of a variable, binning numerical variables, and condensing categories in categorical variables. The following manipulations support the preprocessing step as well the choice of adequate data mining methods. They do so by revealing patterns and their nature.

Rescaling Changing the scale in a display can enhance the plot and illuminate relationships. For example, in Figure 3.8, we see the effect of changing both axes of the scatter plot (top) and the y-axis of a boxplot (bottom) to logarithmic (log) scale. Whereas the original plots (left) are hard to understand, the patterns become visible in log scale (right). In the histograms, the nature of the relationship between MEDV and CRIM is hard to determine in the original scale, because too many of the points are "crowded" near the y-axis. The rescaling removes this crowding and allows a better view of the linear relationship between the two log-scaled variables (indicating a log–log relationship). In the boxplot displaying the crowding toward the x-axis in the original units does not allow us to compare the two box sizes, their locations, lower outliers, and most of the distribution information. Rescaling removes the "crowding to the x-axis" effect, thereby allowing a comparison of the two boxplots.

Aggregation and Hierarchies Another useful manipulation of scaling is changing the level of aggregation. For a temporal scale, we can aggregate by different granularity (e.g., monthly, daily, hourly) or even by a "seasonal" factor of interest such as month-of-year or day-of-week. A popular aggregation for time series is a moving average, where the average of neighboring values within a given window size is plotted. Moving average plots enhance visualizing a global trend (see Chapter 16).

Non-temporal variables can be aggregated if some meaningful hierarchy exists: geographical (tracts within a zip code in the Boston Housing example), organizational (people within departments within units), etc. Figure 3.9 illustrates two types of aggregation for the railway ridership time series. The original monthly series is shown in the top-left panel. Seasonal aggregation (by month-of-year) is shown in the top-right panel, where it is easy to see the peak in ridership in July–August and the dip in January–February. The bottom-right panel shows temporal aggregation, where the series is now displayed in yearly aggregates. This plot reveals the global long-term trend in ridership and the generally increasing trend from 1996 on.

Examining different scales, aggregations, or hierarchies supports both supervised and unsupervised tasks in that it can reveal patterns and relationships at various levels, and can suggest new sets of variables with which to work.

Statistics.com PA 3-04 12 2019 may not be reproduced

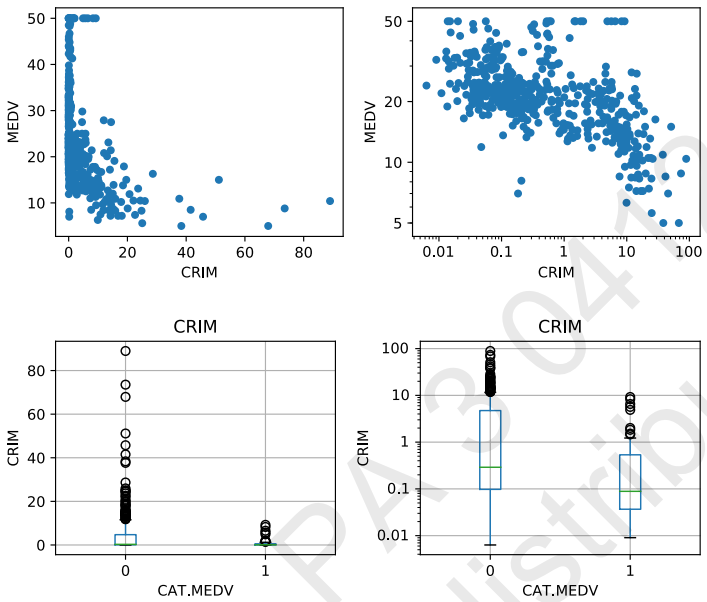

FIGURE 3.8 **RESCALING CAN ENHANCE PLOTS AND REVEAL PATTERNS. LEFT: ORIGINAL SCALE, RIGHT: LOGARITHMIC SCALE**

code for creating Figure 3.8

```
# Avoid the use of scientific notation for the log axis
plt.rcParams['axes.formatter.min_exponent'] = 4

## scatter plot: regular and log scale
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 4))

# regular scale
housing_df.plot.scatter(x='CRIM', y='MEDV', ax=axes[0])
# log scale
ax = housing_df.plot.scatter(x='CRIM', y='MEDV', logx=True, logy=True, ax=axes[1])
ax.set_yticks([5, 10, 20, 50])
ax.set_yticklabels([5, 10, 20, 50])
plt.tight_layout(); plt.show()

## boxplot: regular and log scale
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))

# regular scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[0])
ax.set_xlabel('CAT.MEDV'); ax.set_ylabel('CRIM')
# log scale
ax = housing_df.boxplot(column='CRIM', by='CAT_MEDV', ax=axes[1])
ax.set_xlabel('CAT.MEDV'); ax.set_ylabel('CRIM'); ax.set_yscale('log')
# suppress the title
axes[0].get_figure().suptitle(''); plt.tight_layout(); plt.show()
```

Statistics.com PA 3 04 2019
may not be redistributed

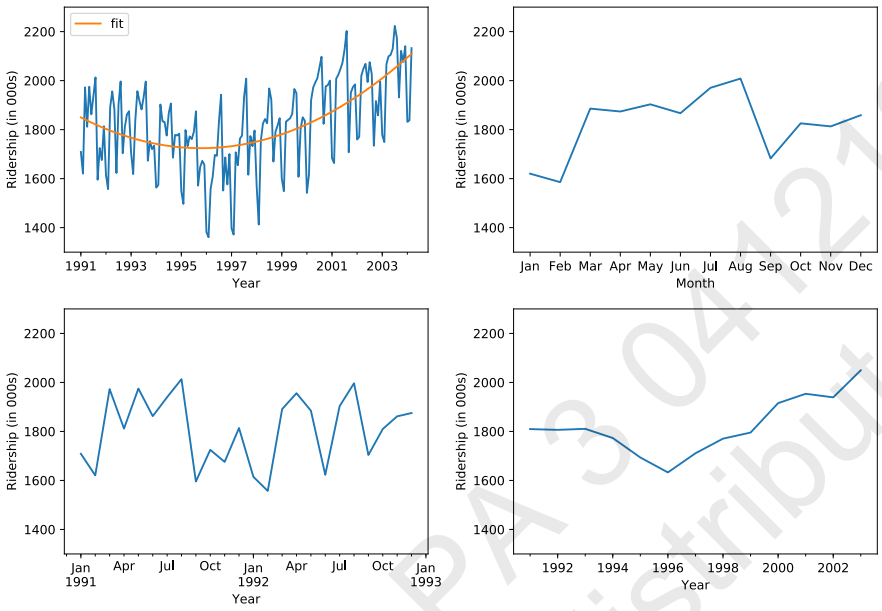

FIGURE 3.9 TIME SERIES LINE GRAPHS USING DIFFERENT AGGREGATIONS (RIGHT PANELS), ADDING CURVES (TOP-LEFT PANEL), AND ZOOMING IN (BOTTOM-LEFT PANEL)

Zooming and Panning The ability to zoom in and out of certain areas of the data on a plot is important for revealing patterns and outliers. We are often interested in more detail on areas of dense information or of special interest. Panning refers to the operation of moving the zoom window to other areas (popular in mapping applications such as Google Maps). An example of zooming is shown in the bottom-left panel of Figure 3.9, where the ridership series is zoomed in to the first two years of the series.

Zooming and panning support supervised and unsupervised methods by detecting areas of different behavior, which may lead to creating new interaction terms, new variables, or even separate models for data subsets. In addition, zooming and panning can help choose between methods that assume global behavior (e.g., regression models) and data-driven methods (e.g., exponential smoothing forecasters and k-nearest-neighbors classifiers), and indicate the level of global/local behavior (as manifested by parameters such as k in k-nearest neighbors, the size of a tree, or the smoothing parameters in exponential smoothing).

Filtering Filtering means removing some of the observations from the plot. The purpose of filtering is to focus the attention on certain data while eliminating "noise" created by other data. Filtering supports supervised and

Statistics.com PA 3 041219 may not be redistributed

code for creating Figure 3.9

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 7))

Amtrak_df = pd.read_csv('Amtrak.csv')
Amtrak_df['Month'] = pd.to_datetime(Amtrak_df.Month, format='%d/%m/%Y')
Amtrak_df.set_index('Month', inplace=True)

# fit quadratic curve and display
quadraticFit = np.poly1d(np.polyfit(range(len(Amtrak_df)), Amtrak_df.Ridership, 2))
Amtrak_fit = pd.DataFrame({'fit': [quadraticFit(t) for t in range(len(Amtrak_df))]})
Amtrak_fit.index = Amtrak_df.index

ax = Amtrak_df.plot(ylim=[1300, 2300], legend=False, ax=axes[0][0])
Amtrak_fit.plot(ax=ax)
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)')  # set x and y-axis label

# Zoom in 2-year period 1/1/1991 to 12/1/1992
ridership_2yrs = Amtrak_df.loc['1991-01-01':'1992-12-01']
ax = ridership_2yrs.plot(ylim=[1300, 2300], legend=False, ax=axes[1][0])
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)')  # set x and y-axis label

# Average by month
byMonth = Amtrak_df.groupby(by=[Amtrak_df.index.month]).mean()
ax = byMonth.plot(ylim=[1300, 2300], legend=False, ax=axes[0][1])
ax.set_xlabel('Month'); ax.set_ylabel('Ridership (in 000s)')  # set x and y-axis label
yticks = [-2.0,-1.75,-1.5,-1.25,-1.0,-0.75,-0.5,-0.25,0.0]
ax.set_xticks(range(1, 13))
ax.set_xticklabels([calendar.month_abbr[i] for i in range(1, 13)]);

# Average by year (exclude data from 2004)
byYear = Amtrak_df.loc['1991-01-01':'2003-12-01'].groupby(pd.Grouper(freq='A')).mean()
ax = byYear.plot(ylim=[1300, 2300], legend=False, ax=axes[1][1])
ax.set_xlabel('Year'); ax.set_ylabel('Ridership (in 000s)')  # set x and y-axis label

plt.tight_layout()
plt.show()
```

Statistics.com PA 3 041219
may not be redistributed

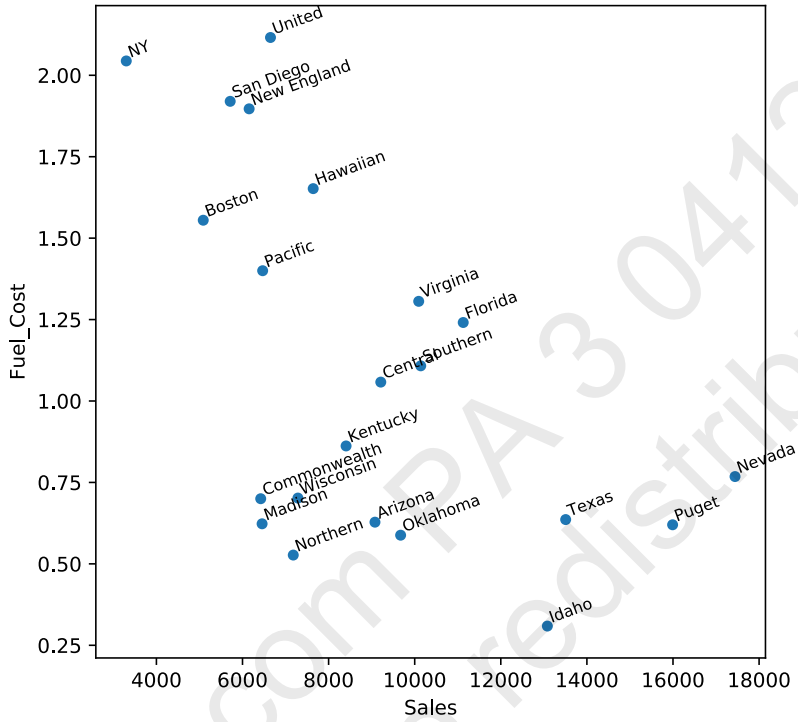

FIGURE 3.10 SCATTER PLOT WITH LABELED POINTS

code for creating Figure 3.10

```
utilities_df = pd.read_csv('Utilities.csv')

ax = utilities_df.plot.scatter(x='Sales', y='Fuel_Cost', figsize=(6, 6))
points = utilities_df[['Sales','Fuel_Cost','Company']]
_ = points.apply(lambda x:
        ax.text(*x, rotation=20, horizontalalignment='left',
                verticalalignment='bottom', fontsize=8), axis=1)
```

unsupervised learning in a similar way to zooming and panning: it assists in identifying different or unusual local behavior.

Reference: Trend Lines and Labels

Trend lines and using in-plot labels also help to detect patterns and outliers. Trend lines serve as a reference, and allow us to more easily assess the shape of

Statistics.com PA 3 041219 may not be redistributed

a pattern. Although linearity is easy to visually perceive, more elaborate relationships such as exponential and polynomial trends are harder to assess by eye. Trend lines are useful in line graphs as well as in scatter plots. An example is shown in the top-left panel of Figure 3.9, where a polynomial curve is overlaid on the original line graph (see also Chapter 16).

In displays that are not overcrowded, the use of in-plot labels can be useful for better exploration of outliers and clusters. An example is shown in Figure 3.10 (a reproduction of Figure 15.1 with the addition of labels). The figure shows different utilities on a scatter plot that compares fuel cost with total sales. We might be interested in clustering the data, and using clustering algorithms to identify clusters that differ markedly with respect to fuel cost and sales. Figure 3.10, with the labels, helps visualize these clusters and their members (e.g., Nevada and Puget are part of a clear cluster with low fuel costs and high sales). For more on clustering and on this example, see Chapter 15.

Scaling up to Large Datasets

When the number of observations (rows) is large, plots that display each individual observation (e.g., scatter plots) can become ineffective. Aside from using aggregated charts such as boxplots, some alternatives are:

1. Sampling—drawing a random sample and using it for plotting
2. Reducing marker size
3. Using more transparent marker colors and removing fill
4. Breaking down the data into subsets (e.g., by creating multiple panels)
5. Using aggregation (e.g., bubble plots where size corresponds to number of observations in a certain range)
6. Using jittering (slightly moving each marker by adding a small amount of noise)

An example of the advantage of plotting a sample over the large dataset is shown in Figure 12.2 in Chapter 12, where a scatter plot of 5000 records is plotted alongside a scatter plot of a sample. Figure 3.11 illustrates an improved plot of the full dataset by using smaller markers, using jittering to uncover overlaid points, and more transparent colors. We can see that larger areas of the plot are dominated by the grey class, the black class is mainly on the right, while there is a lot of overlap in the top-right area.

Multivariate Plot: Parallel Coordinates Plot

Another approach toward presenting multidimensional information in a two-dimensional plot is via specialized plots such as the *parallel coordinates plot*. In this

Statistics.com PA 6/4/2019 may not be redistributed

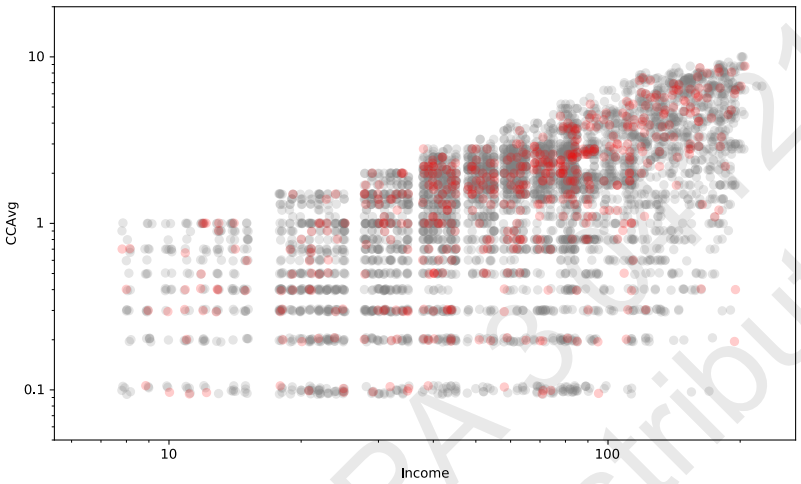

FIGURE 3.11 SCATTER PLOT OF LARGE DATASET WITH REDUCED MARKER SIZE, JITTERING, AND MORE TRANSPARENT COLORING

code for creating Figure 3.11

```
def jitter(x, factor=1):
    """ Add random jitter to x values """
    sx = np.array(sorted(x))
    delta = sx[1:] - sx[:-1]
    minDelta = min(d for d in delta if d > 0)
    a = factor * minDelta / 5
    return x + np.random.uniform(-a, a, len(x))

universal_df = pd.read_csv('UniversalBank.csv')

saIdx = universal_df[universal_df['Securities Account'] == 1].index

plt.figure(figsize=(10,6))
plt.scatter(jitter(universal_df.drop(saIdx).Income),
            jitter(universal_df.drop(saIdx).CCAvg),
            marker='o', color='grey', alpha=0.2)
plt.scatter(jitter(universal_df.loc[saIdx].Income),
            jitter(universal_df.loc[saIdx].CCAvg),
            marker='o', color='red', alpha=0.2)
plt.xlabel('Income')
plt.ylabel('CCAvg')
plt.ylim((0.05, 20))
axes = plt.gca()
axes.set_xscale("log")
axes.set_yscale("log")
plt.show()
```

Statistics.com PASS 2019
may not be redistributed

plot a vertical axis is drawn for each variable. Then each observation is represented by drawing a line that connects its values on the different axes, thereby creating a "multivariate profile." An example is shown in Figure 3.12 for the Boston Housing data. In this display, separate panels are used for the two values of CAT.MEDV, in order to compare the profiles of homes in the two classes (for a classification task). We see that the more expensive homes (bottom panel) consistently have low CRIM, low LSAT, and high RM compared to cheaper homes (top panel), which are more mixed on CRIM, and LSAT, and have a medium level of RM. This observation gives indication of useful predictors and suggests possible binning for some numerical predictors.

Parallel coordinates plots are also useful in unsupervised tasks. They can reveal clusters, outliers, and information overlap across variables. A useful manipulation is to reorder the columns to better reveal observation clusterings.

Interactive Visualization

Similar to the interactive nature of the data mining process, interactivity is key to enhancing our ability to gain information from graphical visualization. In the words of Stephen Few (Few, 2009), an expert in data visualization:

> We can only learn so much when staring at a static visualization such as a printed graph …If we can't interact with the data …we hit the wall.

By interactive visualization, we mean an interface that supports the following principles:

1. Making changes to a chart is *easy, rapid, and reversible*.
2. Multiple concurrent charts and tables can be easily combined and displayed on a single screen.
3. A set of visualizations can be linked, so that operations in one display are reflected in the other displays.

Let us consider a few examples where we contrast a static plot generator (e.g., Excel) with an interactive visualization interface.

Histogram rebinning Consider the need to bin a numerical variables and using a histogram for that purpose. A static histogram would require replotting for each new binning choice. If the user generates multiple plots, then the screen becomes cluttered. If the same plot is recreated, then it is hard to compare different binning choices. In contrast, an interactive visualization would provide an easy way to change bin width interactively (see, e.g., the slider below the histogram in Figure 3.13), and then the histogram would automatically and rapidly replot as the user changes the bin width.

Statistics.com PA 2 04 2019 may not be redistributed

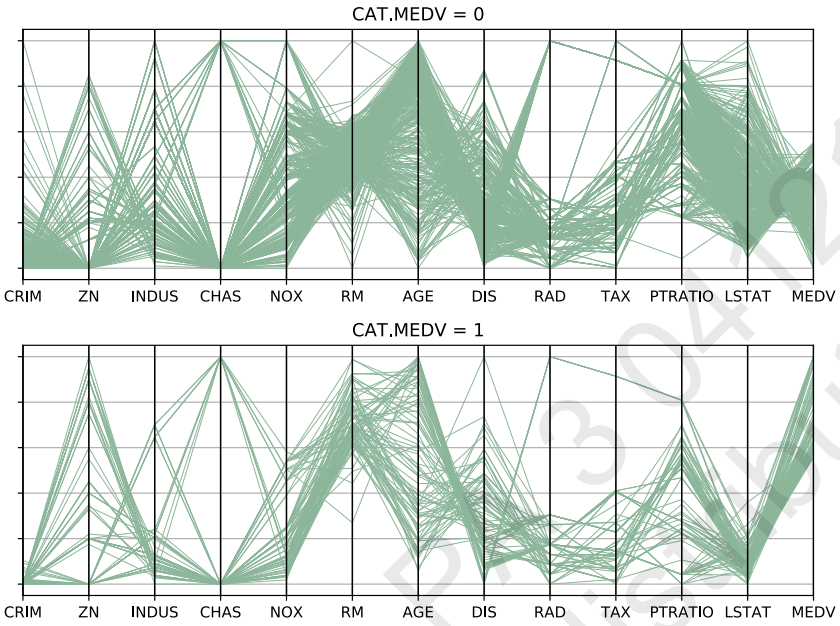

FIGURE 3.12 **PARALLEL COORDINATES PLOT FOR BOSTON HOUSING DATA. EACH OF THE VARIABLES (SHOWN ON THE HORIZONTAL AXIS) IS SCALED TO 0–100%. PANELS ARE USED TO DISTINGUISH CAT.MEDV (TOP PANEL = HOMES BELOW $30,000)**

code for creating Figure 3.12

```
# Transform the axes, so that they all have the same range
min_max_scaler = preprocessing.MinMaxScaler()
dataToPlot = pd.DataFrame(min_max_scaler.fit_transform(housing_df),
                          columns=housing_df.columns)

fig, axes = plt.subplots(nrows=2, ncols=1)
for i in (0, 1):
    parallel_coordinates(dataToPlot.loc[dataToPlot.CAT_MEDV == i],
                         'CAT_MEDV', ax=axes[i], linewidth=0.5)
    axes[i].set_title('CAT.MEDV = {}'.format(i))
    axes[i].set_yticklabels([])
    axes[i].legend().set_visible(False)

plt.tight_layout()  # Increase the separation between the plots
```

Aggregation and Zooming Consider a time series forecasting task, given a long series of data. Temporal aggregation at multiple levels is needed for determining short and long term patterns. Zooming and panning are used to identify unusual periods. A static plotting software requires the user to compute new variables for each temporal aggregation (e.g., aggregate daily data to obtain weekly aggregates). Zooming and panning requires manually changing the min

Statistics.com pp. 3 04/12/19 may not be redistributed

and max values on the axis scale of interest (thereby losing the ability to quickly move between different areas without creating multiple charts). An interactive visualization would provide immediate temporal hierarchies which the user can easily switch between. Zooming would be enabled as a slider near the axis (see, e.g., the sliders on the top-left panel in Figure 3.13), thereby allowing direct manipulation and rapid reaction.

Combining Multiple Linked Plots That Fit in a Single Screen To support a classification task, multiple plots are created of the outcome variable vs. potential categorical and numerical predictors. These can include side-by-side boxplots, color-coded scatter plots, and multipanel bar charts. The user wants to detect possible multidimensional relationships (and identify possible outliers) by selecting a certain subset of the data (e.g., a single category of some variable) and locating the observations on the other plots. In a static interface, the user would have to manually organize the plots of interest and resize them in order to fit within a single screen. A static interface would usually not support inter-plot linkage, and even if it did, the entire set of plots would have to be regenerated each time a selection is made. In contrast, an interactive visualization would provide an easy way to automatically organize and resize the set of plots to fit within a screen. Linking the set of plots would be easy, and in response to the users selection on one plot, the appropriate selection would be automatically highlighted in the other plots (see example in Figure 3.13).

In earlier sections, we used plots to illustrate the advantages of visualizations, because "a picture is worth a thousand words." The advantages of an interactive visualization are even harder to convey in words. As Ben Shneiderman, a well-known researcher in information visualization and interfaces, puts it:

> A picture is worth a thousand words. An interface is worth a thousand pictures.

The ability to interact with plots, and link them together turns plotting into an analytical tool that supports continuous exploration of the data. Several commercial visualization tools provide powerful capabilities along these lines; two very popular ones are Spotfire (http://spotfire.tibco.com) and Tableau (www.tableausoftware.com); Figure 3.13 was generated using Spotfire.

Tableau and Spotfire have spent hundreds of millions of dollars on software R&D and review of interactions with customers to hone interfaces that allow analysts to interact with data via plots smoothly and efficiently. It is difficult to replicate the sophisticated and highly engineered user interface required for rapid progression through different exploratory views of the data in a programming language like Python. The need is there, however, and the Python community is moving to provide interactivity in plots. The widespread use of JavaScript in web development has led some programmers to combine Python with JavaScript libraries such as Plotly or Bokeh (see *Plotly Python Library* at

Statistics.com PA 3 Due 12/19 may not be redistributed

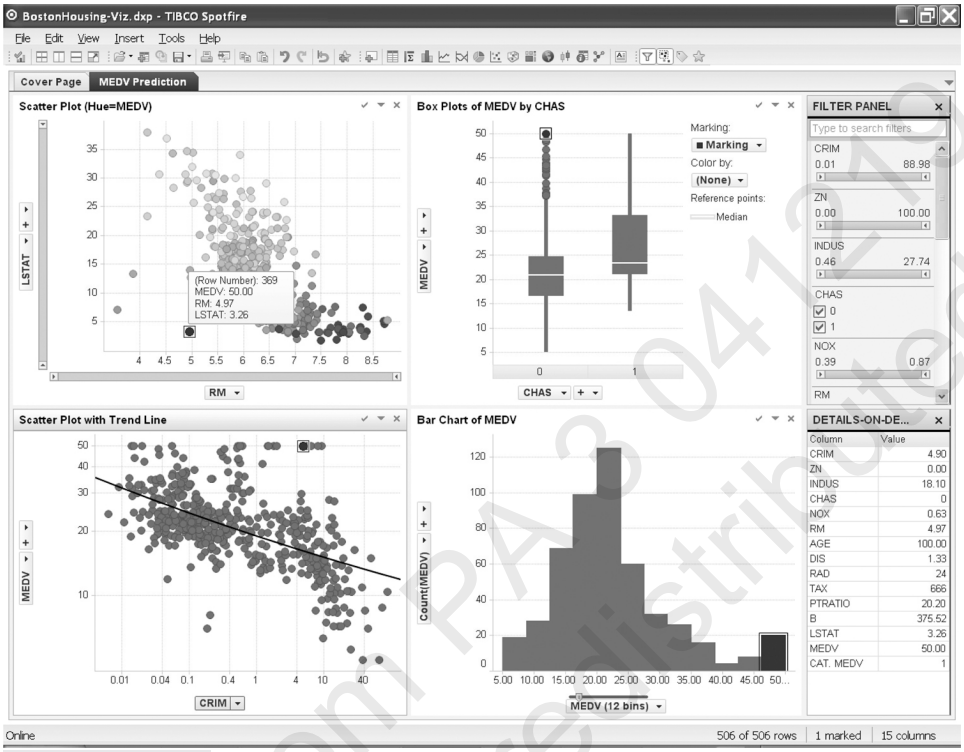

FIGURE 3.13 MULTIPLE INTER-LINKED PLOTS IN A SINGLE VIEW (USING SPOTFIRE). NOTE THE MARKED OBSERVATION IN THE TOP-LEFT PANEL, WHICH IS ALSO HIGHLIGHTED IN ALL OTHER PLOTS

https://plot.ly/python/ or *Developing with JavaScript* at http://bokeh.pydata.org/en/latest/docs/user_guide/bokehjs.html). As of the time of this writing, interactivity tools in Python were evolving, and Python programmers are likely to see more and higher level tools emerging. This should allow them to develop custom interactive plots quickly and deploy these visualizations for use by other non-programming analysts in the organization.

3.5 Specialized Visualizations

In this section, we mention a few specialized visualizations that are able to capture data structures beyond the standard time series and cross-sectional structures—special types of relationships that are usually hard to capture with ordinary plots. In particular, we address hierarchical data, network data, and geographical data—three types of data that are becoming increasingly available.

Statistics.com PA 3 04/12/19 may not be redistributed

Visualizing Networked Data

Network analysis techniques were spawned by the explosion of social and product network data. Examples of social networks are networks of sellers and buyers on eBay and networks of users on Facebook. An example of a product network is the network of products on Amazon (linked through the recommendation system). Network data visualization is available in various network-specialized software, and also in general-purpose software.

A network diagram consists of actors and relations between them. "Nodes" are the actors (e.g., users in a social network or products in a product network), and represented by circles. "Edges" are the relations between nodes, and are represented by lines connecting nodes. For example, in a social network such as Facebook, we can construct a list of users (nodes) and all the pairwise relations (edges) between users who are "Friends." Alternatively, we can define edges as a posting that one user posts on another user's Facebook page. In this setup, we might have more than a single edge between two nodes. Networks can also have nodes of multiple types. A common structure is networks with two types of nodes. An example of a two-type node network is shown in Figure 3.14, where we see a set of transactions between a network of sellers and buyers on the online auction site www.eBay.com [the data are for auctions selling Swarovski beads, and took place during a period of several months; from Jank and Yahav (2010)]. The black circles represent sellers and the grey circles represent buyers. We can see that this marketplace is dominated by a few high-volume sellers. We can also see that many buyers interact with a single seller. The market structures for many individual products could be reviewed quickly in this way. Network providers could use the information, for example, to identify possible partnerships to explore with sellers.

Figure 3.14 was produced using Python's `networkx` package. Another useful package is `python-igraph`.Using these packages, networks can be imported from a variety of sources. The plot's appearance can be customized and various features are available such as filtering nodes and edges, altering the plot's layout, finding clusters of related nodes, calculating network metrics, and performing network analysis (see Chapter 19 for details and examples).

Network plots can be potentially useful in the context of association rules (see Chapter 14). For example, consider a case of mining a dataset of consumers' grocery purchases to learn which items are purchased together ("what goes with what"). A network can be constructed with items as nodes and edges connecting items that were purchased together. After a set of rules is generated by the data mining algorithm (which often contains an excessive number of rules, many of which are unimportant), the network plot can help visualize different rules for the purpose of choosing the interesting ones. For example, a popular "beer and diapers" combination would appear in the network plot as a pair of nodes with

Statistics.com DA 3 04 12 19
may not be reproduced

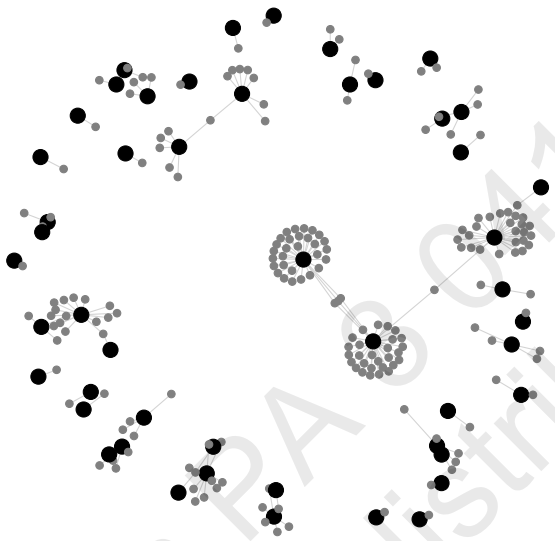

FIGURE 3.14 NETWORK PLOT OF EBAY SELLERS (BLACK CIRCLES) AND BUYERS (GREY CIRCLES) OF SWAROVSKI BEADS

code for creating Figure 3.14

```
ebay_df = pd.read_csv('eBayNetwork.csv')

G = nx.from_pandas_edgelist(ebay_df, source='Seller', target='Bidder')

isBidder = [n in set(ebay_df.Bidder) for n in G.nodes()]
pos = nx.spring_layout(G, k=0.13, iterations=60, scale=0.5)
plt.figure(figsize=(10,10))
nx.draw_networkx(G, pos=pos, with_labels=False,
                 edge_color='lightgray',
                 node_color=['gray' if bidder else 'black' for bidder in isBidder],
                 node_size=[50 if bidder else 200 for bidder in isBidder])
plt.axis('off')
plt.show()
```

very high connectivity. An item which is almost always purchased regardless of other items (e.g., milk) would appear as a very large node with high connectivity to all other nodes.

Statistics.com PA 6 041219 may not be redistributed

Visualizing Hierarchical Data: Treemaps

We discussed hierarchical data and the exploration of data at different hierarchy levels in the context of plot manipulations. *Treemaps* are useful visualizations specialized for exploring large data sets that are hierarchically structured (tree-structured). They allow exploration of various dimensions of the data while maintaining the hierarchical nature of the data. An example is shown in Figure 3.15, which displays a large set of auctions from eBay.com,[5] hierarchically ordered by item category, sub-category, and brand. The levels in the hierarchy of the treemap are visualized as rectangles containing sub-rectangles. Categorical variables can be included in the display by using hue. Numerical variables can be included via rectangle size and color intensity (ordering of the rectangles is sometimes used to reinforce size). In Figure 3.15, size is used to represent the average closing price (which reflects item value) and color intensity represents the percent of sellers with negative feedback (a negative seller feedback indicates buyer dissatisfaction in past transactions and is often indicative of fraudulent seller behavior). Consider the task of classifying ongoing auctions in terms of a fraudulent outcome. From the treemap, we see that the highest proportion of sellers with negative ratings (black) is concentrated in expensive item auctions (Rolex and Cartier wristwatches). Currently, Python offers only simple treemaps with a single hierarchy level and coloring or sizing boxes by additional variables. Figure 3.16 shows a treemap corresponding to Figure 3.15 that was created in R.

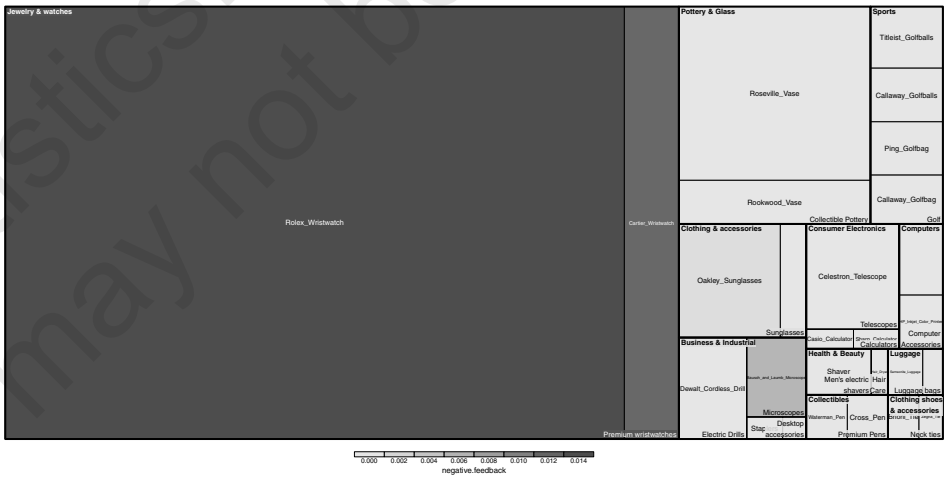

FIGURE 3.15 TREEMAP SHOWING NEARLY 11,000 EBAY AUCTIONS, ORGANIZED BY ITEM CATEGORY, SUBCATEGORY, AND BRAND. RECTANGLE SIZE REPRESENTS AVERAGE CLOSING PRICE (REFLECTING ITEM VALUE). SHADE REPRESENTS PERCENTAGE OF SELLERS WITH NEGATIVE FEEDBACK (DARKER = HIGHER). THIS GRAPH WAS GENERATED USING R; FOR A PYTHON VERSION SEE FIGURE 3.16.

[5] We thank Sharad Borle for sharing this dataset.

Statistics.com PASS 04/2019 may not be posted/shared

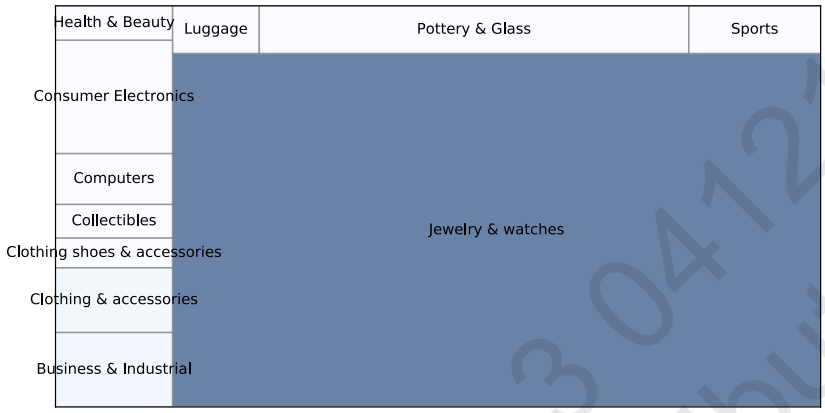

FIGURE 3.16 TREEMAP SHOWING NEARLY 11,000 EBAY AUCTIONS, ORGANIZED BY ITEM CATEGORY. RECTANGLE SIZE REPRESENTS AVERAGE CLOSING PRICE. SHADE REPRESENTS % OF SELLERS WITH NEGATIVE FEEDBACK (DARKER = HIGHER)

code for creating Figure 3.16

```
import squarify

ebayTreemap = pd.read_csv('EbayTreemap.csv')

grouped = []
for category, df in ebayTreemap.groupby(['Category']):
    negativeFeedback = sum(df['Seller Feedback'] < 0) / len(df)
    grouped.append({
        'category': category,
        'negativeFeedback': negativeFeedback,
        'averageBid': df['High Bid'].mean()
    })
byCategory = pd.DataFrame(grouped)

norm = matplotlib.colors.Normalize(vmin=byCategory.negativeFeedback.min(),
                                   vmax=byCategory.negativeFeedback.max())
colors = [matplotlib.cm.Blues(norm(value)) for value in byCategory.negativeFeedback]

fig, ax = plt.subplots()
fig.set_size_inches(9, 5)

squarify.plot(label=byCategory.category, sizes=byCategory.averageBid, color=colors,
              ax=ax, alpha=0.6, edgecolor='grey')

ax.get_xaxis().set_ticks([])
ax.get_yaxis().set_ticks([])

plt.subplots_adjust(left=0.1)
plt.show()
```

Statistics.com PA 3 041219 may not be redistributed

Ideally, treemaps should be explored interactively, zooming to different levels of the hierarchy. One example of an interactive online application of treemaps is currently available at www.drasticdata.nl. One of their treemap examples displays player-level data from the 2014 World Cup, aggregated to team level. The user can choose to explore players and team data.

Visualizing Geographical Data: Map Charts

Many datasets used for data mining now include geographical information. Zip codes are one example of a categorical variable with many categories, where it is not straightforward to create meaningful variables for analysis. Plotting the data on a geographic map can often reveal patterns that are harder to identify otherwise. A map chart uses a geographical map as its background; then color, hue, and other features are used to include categorical or numerical variables. Besides specialized mapping software, maps are now becoming part of general-purpose software, and Google Maps provides APIs (application programming interfaces) that allow organizations to overlay their data on a Google map. While Google Maps is readily available, resulting map charts (such as Figure 3.17) are somewhat inferior in terms of effectiveness compared to map charts in dedicated interactive visualization software.

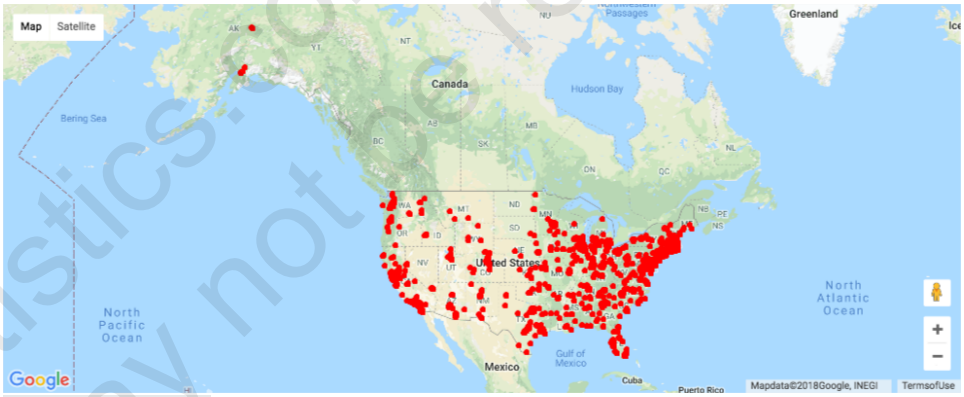

FIGURE 3.17 MAP CHART OF STUDENTS' AND INSTRUCTORS' LOCATIONS ON A GOOGLE MAP (DATA FROM STATISTICS.COM)

code for creating Figure 3.17

```
import gmaps
SCstudents = pd.read_csv('SC-US-students-GPS-data-2016.csv')

gmaps.configure(api_key=os.environ['GMAPS_API_KEY'])
fig = gmaps.figure(center=(39.7, -105), zoom_level=3)
fig.add_layer(gmaps.symbol_layer(SCstudents, scale=2,fill_color='red', stroke_color='red'))
fig
```

Statistics.com PA 3041219 may not be redistributed

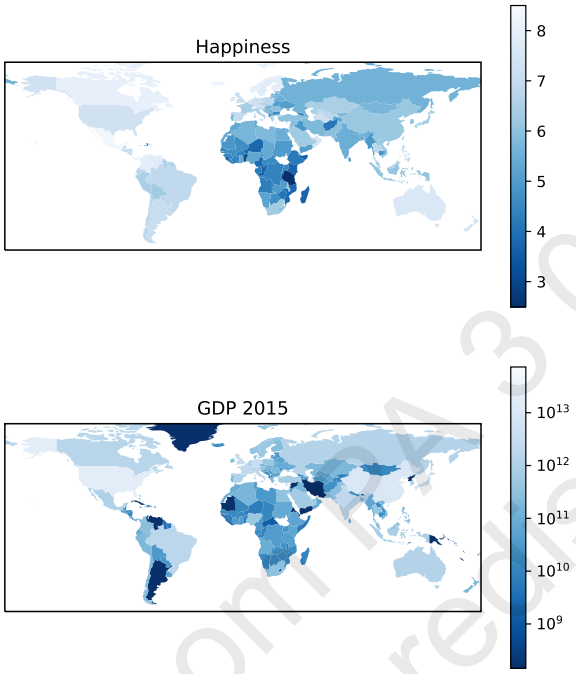

FIGURE 3.18 WORLD MAPS COMPARING "WELL-BEING" (TOP) TO GDP (BOTTOM). SHADING BY AVERAGE "GLOBAL WELL-BEING" SCORE (TOP) OR GDP (BOTTOM) OF COUNTRY. LIGHTER CORRESPONDS TO HIGHER SCORE OR LEVEL. DATA FROM VEENHOVEN'S WORLD DATABASE OF HAPPINESS

Figure 3.18 shows two world map charts comparing countries' "well-being" (according to a 2006 Gallup survey) in the top map, to gross domestic product (GDP) in the bottom map. Lighter shade means higher value.

Statistics.com DA 3041219 may not be redistributed

code for creating Figure 3.18

```
import matplotlib
import matplotlib.pyplot as plt
import cartopy
import cartopy.io.shapereader as shpreader
import cartopy.crs as ccrs

gdp_df = pd.read_csv('gdp.csv', skiprows=4)
gdp_df.rename(columns={'2015': 'GDP2015'}, inplace=True)
gdp_df.set_index('Country Code', inplace=True)  # use three letter country code to access rows

# The file contains a column with two letter combinations, use na_filter to avoid converting
# the combination NA into not-a-number
happiness_df = pd.read_csv('Veerhoven.csv', na_filter = False)
happiness_df.set_index('Code', inplace=True)  # use the country name to access rows

fig = plt.figure(figsize=(7, 8))
ax1 = plt.subplot(2, 1, 1, projection=ccrs.PlateCarree())
ax1.set_extent([-150, 60, -25, 60])
ax2 = plt.subplot(2, 1, 2, projection=ccrs.PlateCarree())
ax2.set_extent([-150, 60, -25, 60])

# Create a color mapper
cmap = plt.cm.Blues_r
norm1 = matplotlib.colors.Normalize(vmin=happiness_df.Score.dropna().min(),
                                    vmax=happiness_df.Score.dropna().max())
norm2 = matplotlib.colors.LogNorm(vmin=gdp_df.GDP2015.dropna().min(),
                                  vmax=gdp_df.GDP2015.dropna().max())

shpfilename = shpreader.natural_earth(resolution='110m', category='cultural', name='admin_0_countries')
reader = shpreader.Reader(shpfilename)
countries = reader.records()
for country in countries:
    countryCode = country.attributes['ADM0_A3']
    if countryCode in gdp_df.index:
        ax2.add_geometries(country.geometry, ccrs.PlateCarree(),
                           facecolor=cmap(norm2(gdp_df.loc[countryCode].GDP2015)))
    # check various attributes to find the matching two-letter combinations
    nation = country.attributes['POSTAL']
    if nation not in happiness_df.index: nation = country.attributes['ISO_A2']
    if nation not in happiness_df.index: nation = country.attributes['WB_A2']
    if nation not in happiness_df.index and country.attributes['NAME'] == 'Norway': nation = 'NO'
    if nation in happiness_df.index:
        ax1.add_geometries(country.geometry, ccrs.PlateCarree(),
                           facecolor=cmap(norm1(happiness_df.loc[nation].Score)))

ax2.set_title("GDP 2015")
sm = plt.cm.ScalarMappable(norm=norm2, cmap=cmap)
sm._A = []
cb = plt.colorbar(sm, ax=ax2)
cb.set_ticks([1e8, 1e9, 1e10, 1e11, 1e12, 1e13])

ax1.set_title("Happiness")
sm = plt.cm.ScalarMappable(norm=norm1, cmap=cmap)
sm._A = []
cb = plt.colorbar(sm, ax=ax1)
cb.set_ticks([3, 4, 5, 6, 7, 8])
plt.show()
```

Statistics.com PA 3 04219
may not be redistributed

3.6 Summary: Major Visualizations and Operations, by Data Mining Goal

Prediction

- Plot outcome on the y-axis of boxplots, bar charts, and scatter plots.
- Study relation of outcome to categorical predictors via side-by-side boxplots, bar charts, and multiple panels.
- Study relation of outcome to numerical predictors via scatter plots.
- Use distribution plots (boxplot, histogram) for determining needed transformations of the outcome variable (and/or numerical predictors).
- Examine scatter plots with added color/panels/size to determine the need for interaction terms.
- Use various aggregation levels and zooming to determine areas of the data with different behavior, and to evaluate the level of global vs. local patterns.

Classification

- Study relation of outcome to categorical predictors using bar charts with the outcome on the y-axis.
- Study relation of outcome to pairs of numerical predictors via color-coded scatter plots (color denotes the outcome).
- Study relation of outcome to numerical predictors via side-by-side boxplots: Plot boxplots of a numerical variable by outcome. Create similar displays for each numerical predictor. The most separable boxes indicate potentially useful predictors.
- Use color to represent the outcome variable on a parallel coordinate plot.
- Use distribution plots (boxplot, histogram) for determining needed transformations of numerical predictor variables.
- Examine scatter plots with added color/panels/size to determine the need for interaction terms.
- Use various aggregation levels and zooming to determine areas of the data with different behavior, and to evaluate the level of global vs. local patterns.

Time Series Forecasting

- Create line graphs at different temporal aggregations to determine types of patterns.
- Use zooming and panning to examine various shorter periods of the series to determine areas of the data with different behavior.

Statistics.com PA 3 041219 may not be redistributed

- Use various aggregation levels to identify global and local patterns.
- Identify missing values in the series (that will require handling).
- Overlay trend lines of different types to determine adequate modeling choices.

Unsupervised Learning

- Create scatter plot matrices to identify pairwise relationships and clustering of observations.
- Use heatmaps to examine the correlation table.
- Use various aggregation levels and zooming to determine areas of the data with different behavior.
- Generate a parallel coordinates plot to identify clusters of observations.

Statistics.com PA 3 041219
may not be redistributed

PROBLEMS

3.1 Shipments of Household Appliances: Line Graphs. The file *ApplianceShipments.csv* contains the series of quarterly shipments (in millions of dollars) of US household appliances between 1985 and 1989.

a. Create a well-formatted time plot of the data using Python.

b. Does there appear to be a quarterly pattern? For a closer view of the patterns, zoom in to the range of 3500–5000 on the y-axis.

c. Using Python, create one chart with four separate lines, one line for each of Q1, Q2, Q3, and Q4. In Python, this can be achieved by add column for quarter and year. Then group the data frame by quarter and then plot shipment versus year for each quarter as a separate series on a line graph. Zoom in to the range of 3500–5000 on the y-axis. Does there appear to be a difference between quarters?

d. Using Python, create a line graph of the series at a yearly aggregated level (i.e., the total shipments in each year).

3.2 Sales of Riding Mowers: Scatter Plots. A company that manufactures riding mowers wants to identify the best sales prospects for an intensive sales campaign. In particular, the manufacturer is interested in classifying households as prospective owners or nonowners on the basis of Income (in \$1000s) and Lot Size (in 1000 ft^2). The marketing expert looked at a random sample of 24 households, given in the file *RidingMowers.csv*.

a. Using Python, create a scatter plot of Lot Size vs. Income, color-coded by the outcome variable owner/nonowner. Make sure to obtain a well-formatted plot (create legible labels and a legend, etc.).

3.3 Laptop Sales at a London Computer Chain: Bar Charts and Boxplots. The file *LaptopSalesJanuary2008.csv* contains data for all sales of laptops at a computer chain in London in January 2008. This is a subset of the full dataset that includes data for the entire year.

a. Create a bar chart, showing the average retail price by store. Which store has the highest average? Which has the lowest?

b. To better compare retail prices across stores, create side-by-side boxplots of retail price by store. Now compare the prices in the two stores from (a). Does there seem to be a difference between their price distributions?

3.4 Laptop Sales at a London Computer Chain: Interactive Visualization. *The next exercises are designed for using an interactive visualization tool. The file* LaptopSales.csv *is a comma-separated file with nearly 300,000 rows. ENBIS (the European Network for Business and Industrial Statistics) provided these data as part of a contest organized in the fall of 2009.*
Scenario: Imagine that you are a new analyst for a company called Acell (a company selling laptops). You have been provided with data about products and sales. You need to help the company with their business goal of planning a product strategy and pricing policies that will maximize Acell's projected revenues in 2009. Using an interactive visualization tool, answer the following questions.

a. Price Questions:

i. At what price are the laptops actually selling?

ii. Does price change with time? (*Hint*: Make sure that the date column is recognized as such. The software should then enable different temporal aggregation

Statistics.com LLC 9041219 may not be redistributed

choices, e.g., plotting the data by weekly or monthly aggregates, or even by day of week.)

iii. Are prices consistent across retail outlets?

iv. How does price change with configuration?

b. Location Questions:

i. Where are the stores and customers located?

ii. Which stores are selling the most?

iii. How far would customers travel to buy a laptop?

- *Hint 1*: You should be able to aggregate the data, for example, plot the sum or average of the prices.
- *Hint 2*: Use the coordinated highlighting between multiple visualizations in the same page, for example, select a store in one view to see the matching customers in another visualization.
- *Hint 3*: Explore the use of filters to see differences. Make sure to filter in the zoomed out view. For example, try to use a "store location" slider as an alternative way to dynamically compare store locations. This might be more useful to spot outlier patterns if there were 50 store locations to compare.

iv. Try an alternative way of looking at how far customers traveled. Do this by creating a new data column that computes the distance between customer and store.

c. Revenue Questions:

i. How do the sales volume in each store relate to Acell's revenues?

ii. How does this relationship depend on the configuration?

d. Configuration Questions:

i. What are the details of each configuration? How does this relate to price?

ii. Do all stores sell all configurations?

Statistics.com PA 3-041219
may not be redistributed

CHAPTER 8

The Naive Bayes Classifier

In this chapter, we introduce the naive Bayes classifier, which can be applied to data with categorical predictors. We review the concept of conditional probabilities, then present the complete, or exact, Bayesian classifier. We next see how it is impractical in most cases, and learn how to modify it and use instead the *naive Bayes* classifier, which is more generally applicable.

Python

In this chapter, we will use `pandas` for data handling, `scikit-learn` for naive Bayes models, and `matplotlib` for visualization. We will also make use of the utility functions from the Python Utilities Functions Appendix.

import required functionality for this chapter

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
import matplotlib.pylab as plt
from dmba import classificationSummary, gainsChart
```

8.1 INTRODUCTION

The naive Bayes method (and, indeed, an entire branch of statistics) is named after the Reverend Thomas Bayes (1702–1761). To understand the naive Bayes classifier, we first look at the complete, or exact, Bayesian classifier. The basic principle is simple. For each record to be classified:

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

Statistics.com PASS 041219 may not be redistributed

1. Find all the other records with the same predictor profile (i.e., where the predictor values are the same).
2. Determine what classes the records belong to and which class is most prevalent.
3. Assign that class to the new record.

Alternatively (or in addition), it may be desirable to tweak the method so that it answers the question: "What is the propensity of belonging to the class of interest?" instead of "Which class is the most probable?" Obtaining class probabilities allows using a sliding cutoff to classify a record as belonging to class C_i, even if C_i is not the most probable class for that record. This approach is useful when there is a specific class of interest that we are interested in identifying, and we are willing to "overidentify" records as belonging to this class. (See Chapter 5 for more details on the use of cutoffs for classification and on asymmetric misclassification costs)

Cutoff Probability Method

1. Establish a cutoff probability for the class of interest above which we consider that a record belongs to that class.
2. Find all the training records with the same predictor profile as the new record (i.e., where the predictor values are the same).
3. Determine the probability that those records belong to the class of interest.
4. If that probability is above the cutoff probability, assign the new record to the class of interest.

Conditional Probability

Both procedures incorporate the concept of *conditional probability*, or the probability of event A given that event B has occurred [denoted $P(A|B)$]. In this case, we will be looking at the probability of the record belonging to class C_i given that its predictor values are $x_1, x_2, \ldots, x_p$. In general, for a response with m classes $C_1, C_2, \ldots, C_m$, and the predictor values $x_1, x_2, \ldots, x_p$, we want to compute

$$P(C_i|x_1, \ldots, x_p). \tag{8.1}$$

To classify a record, we compute its probability of belonging to each of the classes in this way, then classify the record to the class that has the highest probability or use the cutoff probability to decide whether it should be assigned to the class of interest.

From this definition, we see that the Bayesian classifier works only with categorical predictors. If we use a set of numerical predictors, then it is highly

Statistics.com PA 3/04/19 may not be redistributed

unlikely that multiple records will have identical values on these numerical predictors. Therefore, numerical predictors must be binned and converted to categorical predictors. *The Bayesian classifier is the only classification or prediction method presented in this book that is especially suited for (and limited to) categorical predictor variables.*

Example 1: Predicting Fraudulent Financial Reporting

An accounting firm has many large companies as customers. Each customer submits an annual financial report to the firm, which is then audited by the accounting firm. For simplicity, we will designate the outcome of the audit as "fraudulent" or "truthful," referring to the accounting firm's assessment of the customer's financial report. The accounting firm has a strong incentive to be accurate in identifying fraudulent reports—if it passes a fraudulent report as truthful, it would be in legal trouble.

The accounting firm notes that, in addition to all the financial records, it also has information on whether or not the customer has had prior legal trouble (criminal or civil charges of any nature filed against it). This information has not been used in previous audits, but the accounting firm is wondering whether it could be used in the future to identify reports that merit more intensive review. Specifically, it wants to know whether having had prior legal trouble is predictive of fraudulent reporting.

In this case, each customer is a record, and the outcome variable of interest, $Y = \{\text{fraudulent, truthful}\}$, has two classes into which a company can be classified: $C_1 = \text{fraudulent}$ and $C_2 = \text{truthful}$. The predictor variable—"prior legal trouble"—has two values: 0 (no prior legal trouble) and 1 (prior legal trouble).

The accounting firm has data on 1500 companies that it has investigated in the past. For each company, it has information on whether the financial report was judged fraudulent or truthful and whether the company had prior legal trouble. The data were partitioned into a training set (1000 firms) and a validation set (500 firms). Counts in the training set are shown in Table 8.1.

TABLE 8.1 PIVOT TABLE FOR FINANCIAL REPORTING EXAMPLE

	Prior Legal ($X = 1$)	No Prior Legal ($X = 0$)	Total
Fraudulent (C_1)	50	50	100
Truthful (C_2)	180	720	900
Total	230	770	1000

Statistics.com PA 3 041219 may not be redistributed

8.2 Applying the Full (Exact) Bayesian Classifier

Now consider the financial report from a new company, which we wish to classify as either fraudulent or truthful by using these data. To do this, we compute the probabilities, as above, of belonging to each of the two classes.

If the new company had had prior legal trouble, the probability of belonging to the fraudulent class would be P(fraudulent | prior legal) = 50/230 (of the 230 companies with prior legal trouble in the training set, 50 had fraudulent financial reports). The probability of belonging to the other class, "truthful," is, of course, the remainder = 180/230.

Using the "Assign to the Most Probable Class" Method

If a company had prior legal trouble, we assign it to the "truthful" class. Similar calculations for the case of no prior legal trouble are left as an exercise to the reader. In this example, using the rule "assign to the most probable class," all records are assigned to the "truthful" class. This is the same result as the naive rule of "assign all records to the majority class."

Using the Cutoff Probability Method

In this example, we are more interested in identifying the fraudulent reports—those are the ones that can land the auditor in jail. We recognize that, in order to identify the fraudulent reports, some truthful reports will be misidentified as fraudulent, and the overall classification accuracy may decline. Our approach is, therefore, to establish a cutoff value for the probability of being fraudulent, and classify all records above that value as fraudulent. The Bayesian formula for the calculation of this probability that a record belongs to class C_i is as follows:

$$P(C_i|x_1,\ldots,x_p) = \frac{P(x_1,\ldots,x_p|C_i)P(C_i)}{P(x_1,\ldots,x_p|C_1)P(C_1)+\cdots+P(x_1,\ldots,x_p|C_m)P(C_m)}. \quad (8.2)$$

In this example (where frauds are rarer), if the cutoff were established at 0.20, we would classify a prior legal trouble record as fraudulent because P(fraudulent | prior legal) = 50/230 = 0.22. The user can treat this cutoff as a "slider" to be adjusted to optimize performance, like other parameters in any classification model.

Practical Difficulty with the Complete (Exact) Bayes Procedure

The approach outlined above amounts to finding all the records in the sample that are exactly like the new record to be classified in the sense that all the

Statistics.com LLC 3 04.12.19 may not be redistributed

predictor values are all identical. This was easy in the small example presented above, where there was just one predictor.

When the number of predictors gets larger (even to a modest number like 20), many of the records to be classified will be without exact matches. This can be understood in the context of a model to predict voting on the basis of demographic variables. Even a sizable sample may not contain even a single match for a new record who is a male Hispanic with high income from the US Midwest who voted in the last election, did not vote in the prior election, has three daughters and one son, and is divorced. And this is just eight variables, a small number for most data mining exercises. The addition of just a single new variable with five equally frequent categories reduces the probability of a match by a factor of 5.

Solution: Naive Bayes

In the naive Bayes solution, we no longer restrict the probability calculation to those records that match the record to be classified. Instead we use the entire dataset.

Returning to our original basic classification procedure outlined at the beginning of the chapter, recall that the procedure for classifying a new record was:

1. Find all the other records with the same predictor profile (i.e., where the predictor values are the same).
2. Determine what classes the records belong to and which class is most prevalent.
3. Assign that class to the new record.

The naive Bayes modification (for the basic classification procedure) is as follows:

1. For class C_1, estimate the individual conditional probabilities for each predictor $P(x_j|C_1)$—these are the probabilities that the predictor value in the record to be classified occurs in class C_1. For example, for X_1 this probability is estimated by the proportion of x_1 values among the C_1 records in the training set.
2. Multiply these probabilities by each other, then by the proportion of records belonging to class C_1.
3. Repeat Steps 1 and 2 for all the classes.
4. Estimate a probability for class C_i by taking the value calculated in Step 2 for class C_i and dividing it by the sum of such values for all classes.
5. Assign the record to the class with the highest probability for this set of predictor values.

Statistics.com PA 3 04/12/19 may not be redistributed

The above steps lead to the naive Bayes formula for calculating the probability that a record with a given set of predictor values $x_1, \ldots, x_p$ belongs to class C_1 among m classes. The formula can be written as follows:

$$P_{nb}(C_1 \mid x_1, \ldots x_p) = \frac{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)]}{P(C_1)[P(x_1 \mid C_1)P(x_2 \mid C_1) \cdots P(x_p \mid C_1)] + \cdots + P(C_m)[P(x_1 \mid C_m)P(x_2 \mid C_m) \cdots P(x_p \mid C_m)]}. \quad (8.3)$$

This is a somewhat formidable formula; see Example 2 for a simpler numerical version. Note that all the needed quantities can be obtained from pivot tables of Y vs. each of the categorical predictors.

The Naive Bayes Assumption of Conditional Independence

In probability terms, we have made a simplifying assumption that the exact *conditional probability* of seeing a record with predictor profile $x_1, x_2, \ldots, x_p$ within a certain class, $P(x_1, x_2, \ldots, x_p|C_i)$, is well approximated by the product of the individual conditional probabilities $P(x_1|C_i) \times P(x_2|C_i) \cdots \times P(x_p|C_i)$. These two quantities are identical when the predictors are independent within each class.

For example, suppose that "lost money last year" is an additional variable in the accounting fraud example. The simplifying assumption we make with naive Bayes is that, within a given class, we no longer need to look for the records characterized both by "prior legal trouble" and "lost money last year." Rather, assuming that the two are independent, we can simply multiply the probability of "prior legal trouble" by the probability of "lost money last year." Of course, complete independence is unlikely in practice, where some correlation between predictors is expected.

In practice, despite the assumption violation, the procedure works quite well—primarily because what is usually needed is not a propensity for each record that is accurate in absolute terms but just a reasonably accurate *rank ordering* of propensities. Even when the assumption is violated, the rank ordering of the records' propensities is typically preserved.

Note that if all we are interested in is a rank ordering, and the denominator remains the same for all classes, it is sufficient to concentrate only on the numerator. The disadvantage of this approach is that the probability values it yields (the propensities), while ordered correctly, are not on the same scale as the exact values that the user would anticipate.

Statistics.com PA 3 041219 may not be redistributed

Using the Cutoff Probability Method

The above procedure is for the basic case where we seek maximum classification accuracy for all classes. In the case of the *relatively rare class of special interest*, the procedure is:

1. Establish a cutoff probability for the class of interest above which we consider that a record belongs to that class.
2. For the class of interest, compute the probability that each individual predictor value in the record to be classified occurs in the training data.
3. Multiply these probabilities times each other, then times the proportion of records belonging to the class of interest.
4. Estimate the probability for the class of interest by taking the value calculated in Step 3 for the class of interest and dividing it by the sum of the similar values for all classes.
5. If this value falls above the cutoff, assign the new record to the class of interest, otherwise not.
6. Adjust the cutoff value as needed, as a parameter of the model.

Example 2: Predicting Fraudulent Financial Reports, Two Predictors

Let us expand the financial reports example to two predictors, and, using a small subset of data, compare the complete (exact) Bayes calculations to the naive Bayes calculations.

Consider the 10 customers of the accounting firm listed in Table 8.2. For each customer, we have information on whether it had prior legal trouble, whether it is a small or large company, and whether the financial report was found to be fraudulent or truthful. Using this information, we will calculate the conditional probability of fraud, given each of the four possible combinations $\{\text{y, small}\}, \{\text{y, large}\}, \{\text{n, small}\}, \{\text{n, large}\}$.

Complete (Exact) Bayes Calculations: The probabilities are computed as

$$P(\text{fraudulent}|\text{PriorLegal} = \text{y}, \text{Size} = \text{small}) = 1/2 = 0.5$$

$$P(\text{fraudulent}|\text{PriorLegal} = \text{y}, \text{Size} = \text{large}) = 2/2 = 1$$

$$P(\text{fraudulent}|\text{PriorLegal} = \text{n}, \text{Size} = \text{small}) = 0/3 = 0$$

$$P(\text{fraudulent}|\text{PriorLegal} = \text{n}, \text{Size} = \text{large}) = 1/3 = 0.33$$

Naive Bayes Calculations: Now we compute the naive Bayes probabilities. For the conditional probability of fraudulent behaviors given {PriorLegal = y, Size = small}, the numerator is a multiplication of the proportion of {PriorLegal = y} instances among the fraudulent companies, times the proportion of {Size = small} instances among the fraudulent companies, times the proportion

Statistics.com PA 3 041219 may not be redistributed

TABLE 8.2 INFORMATION ON 10 COMPANIES

Company	Prior Legal Trouble	Company Size	Status
1	Yes	Small	Truthful
2	No	Small	Truthful
3	No	Large	Truthful
4	No	Large	Truthful
5	No	Small	Truthful
6	No	Small	Truthful
7	Yes	Small	Fraudulent
8	Yes	Large	Fraudulent
9	No	Large	Fraudulent
10	Yes	Large	Fraudulent

of fraudulent companies: $(3/4)(1/4)(4/10) = 0.075$. To get the actual probabilities, we must also compute the numerator for the conditional probability of truthful behaviors given {PriorLegal = y, Size = small}: $(1/6)(4/6)(6/10) = 0.067$. The denominator is then the sum of these two conditional probabilities $(0.075 + 0.067 = 0.14)$. The conditional probability of fraudulent behaviors given {PriorLegal = y, Size = small} is therefore $0.075/0.14 = 0.53$. In a similar fashion, we compute all four conditional probabilities:

$$P_{\text{nb}}(\text{fraudulent}|\text{PriorLegal = y, Size = small}) = \frac{(3/4)(1/4)(4/10)}{(3/4)(1/4)(4/10) + (1/6)(4/6)(6/10)} = 0.53$$

$$P_{\text{nb}}(\text{fraudulent}|\text{PriorLegal = y, Size = large}) = 0.87$$

$$P_{\text{nb}}(\text{fraudulent}|\text{PriorLegal = n, Size = small}) = 0.07$$

$$P_{\text{nb}}(\text{fraudulent}|\text{PriorLegal = n, Size = large}) = 0.31$$

Note how close these naive Bayes probabilities are to the exact Bayes probabilities. Although they are not equal, both would lead to exactly the same classification for a cutoff of 0.5 (and many other values). It is often the case that the rank ordering of probabilities is even closer to the exact Bayes method than the probabilities themselves, and for classification purposes it is the rank orderings that matter.

We now consider a larger numerical example, where information on flights is used to predict flight delays.

Example 3: Predicting Delayed Flights

Predicting flight delays can be useful to a variety of organizations: airport authorities, airlines, and aviation authorities. At times, joint task forces have been formed to address the problem. If such an organization were to provide ongoing real-time assistance with flight delays, it would benefit from some advance notice about flights that are likely to be delayed.

Statistics.com #A 3041229 may not be redistributed

TABLE 8.3 DESCRIPTION OF VARIABLES FOR FLIGHT DELAYS EXAMPLE

Day of Week	Coded as 1 = Monday, 2 = Tuesday, ..., 7 = Sunday
Sch. Dep. Time	Broken down into 18 intervals between 6:00 AM and 10:00 PM
Origin	Three airport codes: DCA (Reagan National), IAD (Dulles), BWI (Baltimore–Washington Int'l)
Destination	Three airport codes: JFK (Kennedy), LGA (LaGuardia), EWR (Newark)
Carrier	Eight airline codes: CO (Continental), DH (Atlantic Coast), DL (Delta), MQ (American Eagle), OH (Comair), RU (Continental Express), UA (United), and US (USAirways)

Statistics.com PA 3 04 1219 may not be redistributed

In this simplified illustration, we look at five predictors (see Table 8.3). The outcome of interest is whether or not the flight is delayed (*delayed* here means arrived more than 15 minutes late). Our data consist of all flights from the Washington, DC area into the New York City area during January 2004. A record is a particular flight. The percentage of delayed flights among these 2201 flights is 19.5%. The data were obtained from the Bureau of Transportation Statistics (available on the web at www.transtats.bts.gov). The goal is to accurately predict whether or not a new flight (not in this dataset), will be delayed. The outcome variable is whether the flight was delayed or not (1 = delayed and 0 = on time). In addition, information is collected on the predictors listed in Table 8.3.

After converting all predictors to categorical and creating dummies, the data were partitioned into training (60%) and validation (40%) sets, and then a naive Bayes classifier was applied to the training set (see Table 8.4).

Before using the output, let's see how the algorithm works. We start by generating pivot tables for the outcome vs. each of the 5 predictors using the training set, in order to obtain conditional probabilities (Table 8.5). Note that in this example, there are no predictor values that were not represented in the training data.

To classify a new flight, we compute the probability that it will be delayed and the probability that it will be on time. Recall that since both probabilities will have the same denominator, we can just compare the numerators. Each numerator is computed by multiplying all the conditional probabilities of the relevant predictor values and, finally, multiplying by the proportion of that class (in this case $\hat{P}(\text{delayed}) = 0.2$). Let us use an example: to classify a Delta flight from DCA to LGA departing between 10:00 AM and 11:00 AM on a Sunday, we first compute the numerators using the values from the pivot tables:

TABLE 8.4 NAIVE BAYES CLASSIFIER APPLIED TO FLIGHT DELAYS (TRAINING) DATA

code for running naive Bayes

```
delays_df = pd.read_csv('FlightDelays.csv')

# convert to categorical
delays_df.DAY_WEEK = delays_df.DAY_WEEK.astype('category')
delays_df['Flight Status'] = delays_df['Flight Status'].astype('category')

# create hourly bins departure time
delays_df.CRS_DEP_TIME = [round(t / 100) for t in delays_df.CRS_DEP_TIME]
delays_df.CRS_DEP_TIME = delays_df.CRS_DEP_TIME.astype('category')

predictors = ['DAY_WEEK', 'CRS_DEP_TIME', 'ORIGIN', 'DEST', 'CARRIER']
outcome = 'Flight Status'

X = pd.get_dummies(delays_df[predictors])
y = delays_df['Flight Status'].astype('category')
classes = list(y.cat.categories)

# split into training and validation
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.40, random_state=1)

# run naive Bayes
delays_nb = MultinomialNB(alpha=0.01)
delays_nb.fit(X_train, y_train)

# predict probabilities
predProb_train = delays_nb.predict_proba(X_train)
predProb_valid = delays_nb.predict_proba(X_valid)

# predict class membership
y_valid_pred = delays_nb.predict(X_valid)
```

Statistics.com PA 3 041219
may not be redistributed

TABLE 8.5 PIVOT TABLE OF FLIGHT STATUS BY DESTINATION AIRPORT (TRAINING DATA)

```
# split the original data frame into a train and test using the same random_state
train_df, valid_df = train_test_split(delays_df, test_size=0.4, random_state=1)

pd.set_option('precision', 4)
# probability of flight status
print(train_df['Flight Status'].value_counts() / len(train_df))
print()

for predictor in predictors:
    # construct the frequency table
    df = train_df[['Flight Status', predictor]]
    freqTable = df.pivot_table(index='Flight Status', columns=predictor, aggfunc=len)

    # divide each value by the sum of the row to get conditional probabilities
    propTable = freqTable.apply(lambda x: x / sum(x), axis=1)
    print(propTable)
    print()
pd.reset_option('precision')
```

Output

```
ontime     0.8023
delayed    0.1977

DAY_WEEK            1       2       3       4       5      6       7
Flight Status
delayed        0.1916  0.1494  0.1149  0.1264  0.1877  0.069  0.1609
ontime         0.1246  0.1416  0.1445  0.1794  0.1690  0.136  0.1048

CRS_DEP_TIME        6       7       8       9       10      11      12      13  \
Flight Status
delayed        0.0345  0.0536  0.0651  0.0192  0.0307  0.0115  0.0498  0.0460
ontime         0.0623  0.0633  0.0850  0.0567  0.0519  0.0340  0.0661  0.0746

CRS_DEP_TIME       14      15      16      17      18      19      20      21
Flight Status
delayed        0.0383  0.2031  0.0728  0.1533  0.0192  0.0996  0.0153  0.0881
ontime         0.0576  0.1171  0.0774  0.1001  0.0349  0.0397  0.0264  0.0529

ORIGIN            BWI     DCA     IAD
Flight Status
delayed        0.0805  0.5211  0.3985
ontime         0.0604  0.6478  0.2918

DEST              EWR     JFK     LGA
Flight Status
delayed        0.3793  0.1992  0.4215
ontime         0.2663  0.1558  0.5779

CARRIER            CO      DH      DL      MQ      OH      RU      UA      US
Flight Status
delayed        0.0575  0.3142  0.0958  0.2222  0.0077  0.2184  0.0153  0.0690
ontime         0.0349  0.2295  0.2040  0.1171  0.0104  0.1690  0.0170  0.2181
```

Statistics.com PA 3 041219 may not be redistributed

$$\hat{P}(\text{delayed}|\text{Carrier} = \text{DL}, \text{Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA}, \text{Origin} = \text{DCA})$$
$$\propto (0.0958)(0.1609)(0.0307)(0.4215)(0.5211)(0.2) = 0.000021$$
$$\hat{P}(\text{ontime}|\text{Carrier} = \text{DL}, \text{Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA}, \text{Origin} = \text{DCA})$$
$$\propto (0.2040)(0.1048)(0.0519)(0.5779)(0.6478)(0.8) = 0.00033$$

The symbol $\propto$ means "is proportional to," reflecting the fact that this calculation deals only with the numerator in the naive Bayes formula (8.3). Comparing the numerators, it is therefore, more likely that the flight will be on time. Note that a record with such a combination of predictor values does not exist in the training set, and therefore we use the naive Bayes rather than the exact Bayes. To compute the actual probability, we divide each of the numerators by their sum:

$$\hat{P}(\text{delayed}|\text{Carrier} = \text{DL}, \text{Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA}, \text{Origin} = \text{DCA})$$
$$= \frac{0.000021}{0.000021 + 0.00033} = 0.058$$
$$\hat{P}(\text{on time}|\text{Carrier} = \text{DL}, \text{Day\_Week} = 7, \text{Dep\_Time} = 10, \text{Dest} = \text{LGA}, \text{Origin} = \text{DCA})$$
$$= \frac{0.00033}{0.000021 + 0.00033} = 0.942$$

Of course, we rely on software to compute these probabilities for any records of interest (in the training set, the validation set, or for scoring new data). Table 8.6 shows the predicted probability and class for the example flight, which coincide with our manual calculation.

TABLE 8.6 SCORING THE EXAMPLE FLIGHT (PROBABILITY AND CLASS)

code for scoring data using naive Bayes

```
# classify a specific flight by searching in the dataset
# for a flight with the same predictor values
df = pd.concat([pd.DataFrame({'actual': y_valid, 'predicted': y_valid_pred}),
                pd.DataFrame(predProb_valid, index=y_valid.index)], axis=1)
mask = ((X_valid.CARRIER_DL == 1) & (X_valid.DAY_WEEK_7 == 1) &
        (X_valid.CRS_DEP_TIME_10 == 1) &  (X_valid.DEST_LGA == 1) &
        (X_valid.ORIGIN_DCA == 1))

df[mask]
```

Output

```
      actual predicted         0         1
1225  ontime    ontime  0.057989  0.942011
```

Statistics.com PA 3/04/2019 may not be redistributed

TABLE 8.7 CONFUSION MATRICES FOR FLIGHT DELAY USING THE NAIVE BAYES CLASSIFIER

code for confusion matrices

```
# training
classificationSummary(y_train, y_train_pred, class_names=classes)

# validation
classificationSummary(y_valid, y_valid_pred, class_names=classes)
```

Output

```
Confusion Matrix (Accuracy 0.7955)

        Prediction
 Actual delayed  ontime
delayed      52     209
 ontime      61     998

Confusion Matrix (Accuracy 0.7821)

        Prediction
 Actual delayed  ontime
delayed      26     141
 ontime      51     663
```

Finally, to evaluate the performance of the naive Bayes classifier for our data, we can use the confusion matrix, gains and lift charts, and all the measures that were described in Chapter 5. For our example, the confusion matrices for the training and validation sets are shown in Table 8.7. We see that the overall accuracy level is around 80% for both the training and validation data. In comparison, a naive rule that would classify all 880 flights in the validation set as 'on time' would have missed the 172 delayed flights, also resulting in a 80% accuracy. Thus, by a simple accuracy measure, the naive Bayes model does no better than the naive rule. However, examining the gains and lift charts (Figure 8.1) shows the strength of the naive Bayes in capturing the delayed flights effectively, when the goal is ranking.

Statistics.com PA 3 041219 may not be redistributed

code for creating Figure 8.1

```
df = pd.DataFrame({'actual': 1 - y_valid.cat.codes, 'prob': predProb_valid[:, 0]})
df = df.sort_values(by=['prob'], ascending=False).reset_index(drop=True)

fig, ax = plt.subplots()
fig.set_size_inches(4, 4)
gainsChart(df.actual, ax=ax)
```

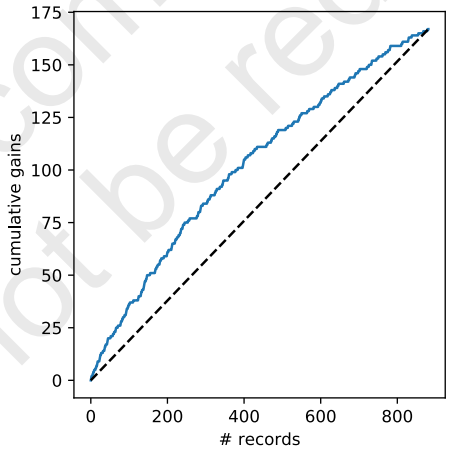

FIGURE 8.1 CUMULATIVE GAINS CHART OF NAIVE BAYES CLASSIFIER APPLIED TO FLIGHT DELAYS DATA

Statistics.com PA 3 041219 may not be redistributed

8.3 Advantages and Shortcomings of the Naive Bayes Classifier

The naive Bayes classifier's beauty is in its simplicity, computational efficiency, good classification performance, and ability to handle categorical variables directly. In fact, it often outperforms more sophisticated classifiers even when the underlying assumption of independent predictors is far from true. This advantage is especially pronounced when the number of predictors is very large.

Three main issues should be kept in mind, however. First, the naive Bayes classifier requires a very large number of records to obtain good results.

Second, where a predictor category is not present in the training data, naive Bayes assumes that a new record with that category of the predictor has zero probability. This can be a problem if this rare predictor value is important. One example is the binary predictor *Weather* in the flights delay dataset, which we did not use for analysis, and which denotes bad weather. When the weather was bad, all flights were delayed. Consider another example, where the outcome variable is *bought high-value life insurance* and a predictor category is *owns yacht*. If the training data have no records with *owns yacht* = 1, for any new records where *owns yacht* = 1, naive Bayes will assign a probability of 0 to the outcome variable *bought high-value life insurance*. With no training records with *owns yacht* = 1, of course, no data mining technique will be able to incorporate this potentially important variable into the classification model—it will be ignored. With naive Bayes, however, the absence of this predictor actively "outvotes" any other information in the record to assign a 0 to the outcome value (when, in this case, it has a relatively good chance of being a 1). The presence of a large training set (and judicious binning of continuous predictors, if required) helps mitigate this effect. A popular solution in such cases is to replace zero probabilities with non-zero values using a method called *smoothing* (e.g., Laplace smoothing can be applied by using argument *alpha* > 0 in function *MultinomialNB*; by default scikit-learn already uses a smothing parameter of 1).

Finally, good performance is obtained when the goal is *classification* or *ranking* of records according to their probability of belonging to a certain class. However, when the goal is to *estimate the probability of class membership (propensity)*, this method provides very biased results. For this reason, the naive Bayes method is rarely used in credit scoring (Larsen, 2005).

Statistics.com FAS 04219 may not be redistributed

SPAM FILTERING

Filtering spam in e-mail has long been a widely familiar application of data mining. Spam filtering, which is based in large part on natural language vocabulary, is a natural fit for a naive Bayesian classifier, which uses exclusively categorical variables. Most spam filters are based on this method, which works as follows:

1. Humans review a large number of e-mails, classify them as "spam" or "not spam," and from these select an equal (also large) number of spam e-mails and non-spam e-mails. This is the training data.
2. These e-mails will contain thousands of words; for each word, compute the frequency with which it occurs in the spam dataset, and the frequency with which it occurs in the non-spam dataset. Convert these frequencies into estimated probabilities (i.e., if the word "free" occurs in 500 out of 1000 spam e-mails, and only 100 out of 1000 non-spam e-mails, the probability that a spam e-mail will contain the word "free" is 0.5, and the probability that a non-spam e-mail will contain the word "free" is 0.1).
3. If the only word in a new message that needs to be classified as spam or not spam is "free," we would classify the message as spam, since the Bayesian posterior probability is 0.5/(0.5+01) or 5/6 that, given the appearance of "free," the message is spam.
4. Of course, we will have many more words to consider. For each such word, the probabilities described in Step 2 are calculated, and multiplied together, and formula (8.3) is applied to determine the naive Bayes probability of belonging to the classes. In the simple version, class membership (spam or not spam) is determined by the higher probability.
5. In a more flexible interpretation, the ratio between the "spam" and "not spam" probabilities is treated as a score for which the operator can establish (and change) a cutoff threshold—anything above that level is classified as spam.
6. Users have the option of building a personalized training database by classifying incoming messages as spam or not spam, and adding them to the training database. One person's spam may be another person's substance.

It is clear that, even with the "Naive" simplification, this is an enormous computational burden. Spam filters now typically operate at two levels—at servers (intercepting some spam that never makes it to your computer) and on individual computers (where you have the option of reviewing it). Spammers have also found ways to "poison" the vocabulary-based Bayesian approach, by including sequences of randomly selected irrelevant words. Since these words are randomly selected, they are unlikely to be systematically more prevalent in spam than in non-spam, and they dilute the effect of key spam terms such as "Viagra" and "free." For this reason, sophisticated spam classifiers also include variables based on elements other than vocabulary, such as the number of links in the message, the vocabulary in the subject line, determination of whether the "From:" e-mail address is the real originator (anti-spoofing), use of HTML and images, and origination at a dynamic or static IP address (the latter are more expensive and cannot be set up quickly).

Statistics.com PA 2014 1219 may not be reproduced

PROBLEMS

8.1 Personal Loan Acceptance. The file *UniversalBank.csv* contains data on 5000 customers of Universal Bank. The data include customer demographic information (age, income, etc.), the customer's relationship with the bank (mortgage, securities account, etc.), and the customer response to the last personal loan campaign (Personal Loan). Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign. In this exercise, we focus on two predictors: Online (whether or not the customer is an active user of online banking services) and Credit Card (abbreviated CC below) (does the customer hold a credit card issued by the bank), and the outcome Personal Loan (abbreviated Loan below).

Partition the data into training (60%) and validation (40%) sets.

a. Create a pivot table for the training data with Online as a column variable, CC as a row variable, and Loan as a secondary row variable. The values inside the table should convey the count. Use the pandas dataframe methods *melt()* and *pivot()*.

b. Consider the task of classifying a customer who owns a bank credit card and is actively using online banking services. Looking at the pivot table, what is the probability that this customer will accept the loan offer? [This is the probability of loan acceptance (Loan = 1) conditional on having a bank credit card (CC = 1) and being an active user of online banking services (Online = 1)].

c. Create two separate pivot tables for the training data. One will have Loan (rows) as a function of Online (columns) and the other will have Loan (rows) as a function of CC.

d. Compute the following quantities [P(A | B) means "the probability of A given B"]:

i. P(CC = 1 | Loan = 1) (the proportion of credit card holders among the loan acceptors)

ii. P(Online = 1 | Loan = 1)

iii. P(Loan = 1) (the proportion of loan acceptors)

iv. P(CC = 1 | Loan = 0)

v. P(Online = 1 | Loan = 0)

vi. P(Loan = 0)

e. Use the quantities computed above to compute the naive Bayes probability P(Loan = 1 | CC = 1, Online = 1).

f. Compare this value with the one obtained from the pivot table in (b). Which is a more accurate estimate?

g. Which of the entries in this table are needed for computing P(Loan = 1 | CC = 1, Online = 1)? In Python, run naive Bayes on the data. Examine the model output on training data, and find the entry that corresponds to P(Loan = 1 | CC = 1, Online = 1). Compare this to the number you obtained in (e).

8.2 Automobile Accidents. The file *accidentsFull.csv* contains information on 42,183 actual automobile accidents in 2001 in the United States that involved one of three levels of injury: NO INJURY, INJURY, or FATALITY. For each accident, additional information is recorded, such as day of week, weather conditions, and road type. A firm might be interested in developing a system for quickly classifying the severity of an accident based on initial reports and associated data in the system (some of which rely on GPS-assisted reporting).

Statistics.com DATA 04/2019 may not be redistributed

Our goal here is to predict whether an accident just reported will involve an injury (MAX_SEV_IR = 1 or 2) or will not (MAX_SEV_IR = 0). For this purpose, create a dummy variable called INJURY that takes the value "yes" if MAX_SEV_IR = 1 or 2, and otherwise "no."

a. Using the information in this dataset, if an accident has just been reported and no further information is available, what should the prediction be? (INJURY = Yes or No?) Why?

b. Select the first 12 records in the dataset and look only at the response (INJURY) and the two predictors WEATHER_R and TRAF_CON_R.

i. Create a pivot table that examines INJURY as a function of the two predictors for these 12 records. Use all three variables in the pivot table as rows/columns.

ii. Compute the exact Bayes conditional probabilities of an injury (INJURY = Yes) given the six possible combinations of the predictors.

iii. Classify the 12 accidents using these probabilities and a cutoff of 0.5.

iv. Compute manually the naive Bayes conditional probability of an injury given WEATHER_R = 1 and TRAF_CON_R = 1.

v. Run a naive Bayes classifier on the 12 records and two predictors using scikit-learn. Check the model output to obtain probabilities and classifications for all 12 records. Compare this to the exact Bayes classification. Are the resulting classifications equivalent? Is the ranking (= ordering) of observations equivalent?

c. Let us now return to the entire dataset. Partition the data into training (60%) and validation (40%).

i. Assuming that no information or initial reports about the accident itself are available at the time of prediction (only location characteristics, weather conditions, etc.), which predictors can we include in the analysis? (Use the data descriptions page from www.dataminingbook.com.)

ii. Run a naive Bayes classifier on the complete training set with the relevant predictors (and INJURY as the response). Note that all predictors are categorical. Show the confusion matrix.

iii. What is the overall error for the validation set?

iv. What is the percent improvement relative to the naive rule (using the validation set)?

v. Examine the conditional probabilities in the pivot tables. Why do we get a probability of zero for P(INJURY = No | SPD_LIM = 5)?

Statistics.com PA 3 04/12/19 may not be redistributed

CHAPTER 9

Classification and Regression Trees

Statistics.com DAS 041219 may not be redistributed

This chapter describes a flexible data-driven method that can be used for both classification (called *classification tree*) and prediction (called *regression tree*). Among the data-driven methods, trees are the most transparent and easy to interpret. Trees are based on separating records into subgroups by creating splits on predictors. These splits create logical rules that are transparent and easily understandable, for example, "IF Age $<$ 55 AND Education $>$ 12 THEN class = 1." The resulting subgroups should be more homogeneous in terms of the outcome variable, thereby creating useful prediction or classification rules. We discuss the two key ideas underlying trees: *recursive partitioning* (for constructing the tree) and *pruning* (for cutting the tree back). In the context of tree construction, we also describe a few metrics of homogeneity that are popular in tree algorithms, for determining the homogeneity of the resulting subgroups of records. We explain that limiting tree size is a useful strategy for avoiding overfitting and show how it is done. We also describe alternative strategies for avoiding overfitting. As with other data-driven methods, trees require large amounts of data. However, once constructed, they are computationally cheap to deploy even on large samples. They also have other advantages such as being highly automated, robust to outliers, and able to handle missing values. In addition to prediction and classification, we describe how trees can be used for dimension reduction. Finally, we introduce *random forests* and *boosted trees*, which combine results from multiple trees to improve predictive power.

Python

In this chapter, we will use `pandas` for data handling, `scikit-learn` for the models, and `matplotlib` and `pydotplus` for visualization. We will also make

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

use of the utility functions from the Python Utilities Functions Appendix. Use the following import statements for the Python code in this chapter.

import required functionality for this chapter

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pylab as plt
from dmba import plotDecisionTree, classificationSummary, regressionSummary
```

9.1 INTRODUCTION

If one had to choose a classification technique that performs well across a wide range of situations without requiring much effort from the analyst while being readily understandable by the consumer of the analysis, a strong contender would be the tree methodology developed by Breiman et al. (1984). We discuss this classification procedure first, then in later sections we show how the procedure can be extended to prediction of a numerical outcome. The program that Breiman et al. created to implement these procedures was called CART (Classification And Regression Trees). A related procedure is called C4.5.

What is a classification tree? Figure 9.1 shows a tree for classifying bank customers who receive a loan offer as either acceptors or nonacceptors, based on information such as their income, education level, and average credit card expenditure.

Tree Structure

We have two types of nodes in a tree: decision (=splitting) nodes and terminal nodes. Nodes that have successors are called *decision nodes* because if we were to use a tree to classify a new record for which we knew only the values of the predictor variables, we would "drop" the record down the tree so that at each decision node, the appropriate branch is taken until we get to a node that has no successors. Such nodes are called the *terminal nodes* (or *leaves* of the tree), and represent the partitioning of the data by predictors.

It is useful to note that the type of trees grown by Python's *DecisionTreeClassifier()* method, also known as CART or *binary trees*, have the property that the number of terminal nodes is exactly one more than the number of decision nodes.

When using the *export_graphviz()* function from scikit-learn, nodes are represented as boxes. The function has a large number of arguments that allow

Statistics.com PA 3 04219 may not be redistributed

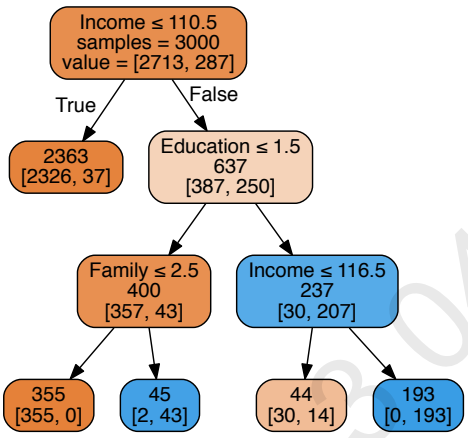

FIGURE 9.1 EXAMPLE OF A TREE FOR CLASSIFYING BANK CUSTOMERS AS LOAN ACCEPTORS OR NONACCEPTORS

Statistics.com PA 3041219 may not be redistributed

controlling the final graph. We use the utility function *plotDecisionTree* from the appendix for plotting the graphs in this chapter. With the chosen settings, all nodes contain information about the number of records in that node (samples), the distribution of the classes, and the majority class of that node. In addition, we color the nodes by the average value of the node for regression or purity of node for classification.

For decision nodes, the name of the predictor variable chosen for splitting and its splitting value appear at the top. Of the two child nodes connected below a decision node, the left box is for records that meet the splitting condition ("True"), while the right box is for records that do not meet it ("False").

Decision Rules

One of the reasons that tree classifiers are very popular is that they provide easily understandable decision rules (at least if the trees are not too large). Consider the tree in the example. The *terminal nodes* are colored orange or blue corresponding to a nonacceptor (0) or acceptor (1) classification. The condition at the top of each splitting node gives the predictor and its splitting value for the split (e.g. *Income* $\leq$110.5 in the top node). *samples*= shows the number of records in that node, and *values*= shows the counts of the two classes (0 and 1) in that node; the labels are only shown in the top node. This tree can easily be translated into a set of rules for classifying a bank customer. For example, the bottom-left node under the "Family" decision node in this tree gives us the following rule:

IF(*Income* > 110.5) AND (*Education* ≤ 1.5) AND (*Family* ≤ 2.5)
THEN *Class* $= 0$ (nonacceptor).

Classifying a New Record

To classify a new record, it is "dropped" down the tree. When it has dropped all the way down to a terminal node, we can assign its class simply by taking a "vote" (or average, if the outcome is numerical) of all the training data that belonged to the terminal node when the tree was grown. The class with the highest vote is assigned to the new record. For instance, a new record reaching the leftmost terminal node in Figure 9.1, which has a majority of records that belong to the 0 class, would be classified as "nonacceptor." Alternatively, we can convert the number of class 0 records in the node to a proportion (propensity) and then compare the proportion to a user-specified cutoff value. In a binary classification situation (typically, with a success class that is relatively rare and of particular interest), we can also establish a lower cutoff to better capture those rare successes (at the cost of lumping in more failures as successes). With a lower cutoff, the votes for the *success* class only need attain that lower cutoff level for the entire terminal node to be classified as a *success*. The cutoff therefore determines the proportion of votes needed for determining the terminal node class. See Chapter 5 for further discussion of the use of a cutoff value in classification, for cases where a single class is of interest.

In the following sections, we show how trees are constructed and evaluated.

9.2 Classification Trees

The key idea underlying tree construction is *recursive partitioning* of the space of the predictor variables. A second important issue is avoiding over-fitting. We start by explaining recursive partitioning (Section 9.2) and then describe approaches for evaluating and fine-tuning trees while avoiding overfitting (Section9.3).

Recursive Partitioning

Let us denote the outcome variable by Y and the input (predictor) variables by $X_1, X_2, X_3, \ldots, X_p$. In classification, the outcome variable will be a categorical variable. Recursive partitioning divides up the p-dimensional space of the X predictor variables into nonoverlapping multidimensional rectangles. The predictor variables here are considered to be continuous, binary, or ordinal. This division is accomplished recursively (i.e., operating on the results of prior divisions). First, one of the predictor variables is selected, say X_i, and a value of X_i, say s_i, is chosen to split the p-dimensional space into two parts: one part

Statistics.com PA 5/04/19 may not be redistributed

TABLE 9.1 LOT SIZE, INCOME, AND OWNERSHIP OF A RIDING MOWER FOR 24 HOUSEHOLDS

Household Number	Income ($000s)	Lot Size (000s ft^2)	Ownership of Riding Mower
1	60.0	18.4	Owner
2	85.5	16.8	Owner
3	64.8	21.6	Owner
4	61.5	20.8	Owner
5	87.0	23.6	Owner
6	110.1	19.2	Owner
7	108.0	17.6	Owner
8	82.8	22.4	Owner
9	69.0	20.0	Owner
10	93.0	20.8	Owner
11	51.0	22.0	Owner
12	81.0	20.0	Owner
13	75.0	19.6	Nonowner
14	52.8	20.8	Nonowner
15	64.8	17.2	Nonowner
16	43.2	20.4	Nonowner
17	84.0	17.6	Nonowner
18	49.2	17.6	Nonowner
19	59.4	16.0	Nonowner
20	66.0	18.4	Nonowner
21	47.4	16.4	Nonowner
22	33.0	18.8	Nonowner
23	51.0	14.0	Nonowner
24	63.0	14.8	Nonowner

that contains all the points with $X_i < s_i$ and the other with all the points with $X_i \geq s_i$. Then, one of these two parts is divided in a similar manner by again choosing a predictor variable (it could be X_i or another variable) and a split value for that variable. This results in three (multidimensional) rectangular regions. This process is continued so that we get smaller and smaller rectangular regions. The idea is to divide the entire X-space up into rectangles such that each rectangle is as homogeneous or "pure" as possible. By *pure*, we mean containing records that belong to just one class. (Of course, this is not always possible, as there may be records that belong to different classes but have exactly the same values for every one of the predictor variables.)

Let us illustrate recursive partitioning with an example.

Example 1: Riding Mowers

We again use the riding-mower example presented in Chapter 3. A riding-mower manufacturer would like to find a way of classifying families in a city into those likely to purchase a riding mower and those not likely to buy one. A pilot random sample of 12 owners and 12 nonowners in the city is undertaken. The data are shown and plotted in Table 9.1 and Figure 9.2.

If we apply the classification tree procedure to these data, the procedure will choose *Income* for the first split with a splitting value of 60. The (X_1, X_2) space

Statistics.com PA 3 04219 may not be redistributed

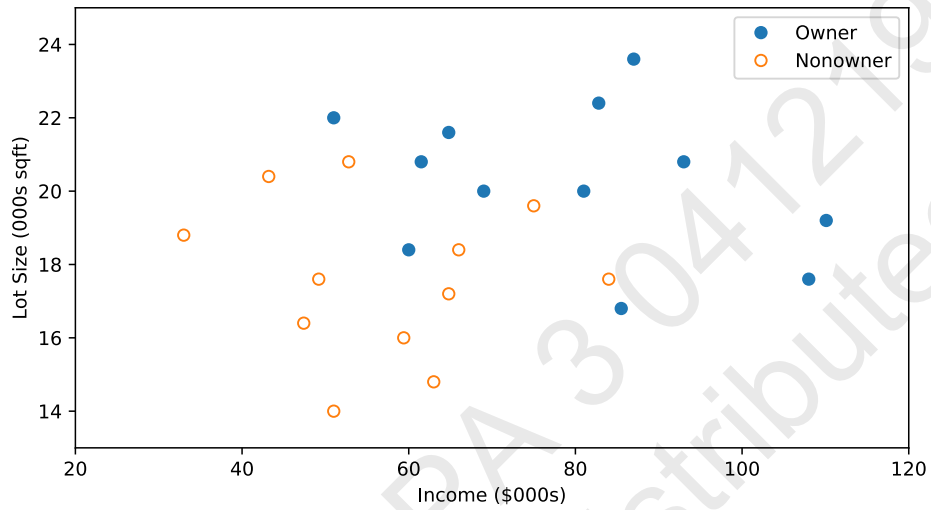

FIGURE 9.2 SCATTER PLOT OF LOT SIZE VS. INCOME FOR 24 OWNERS AND NONOWNERS OF RIDING MOWERS

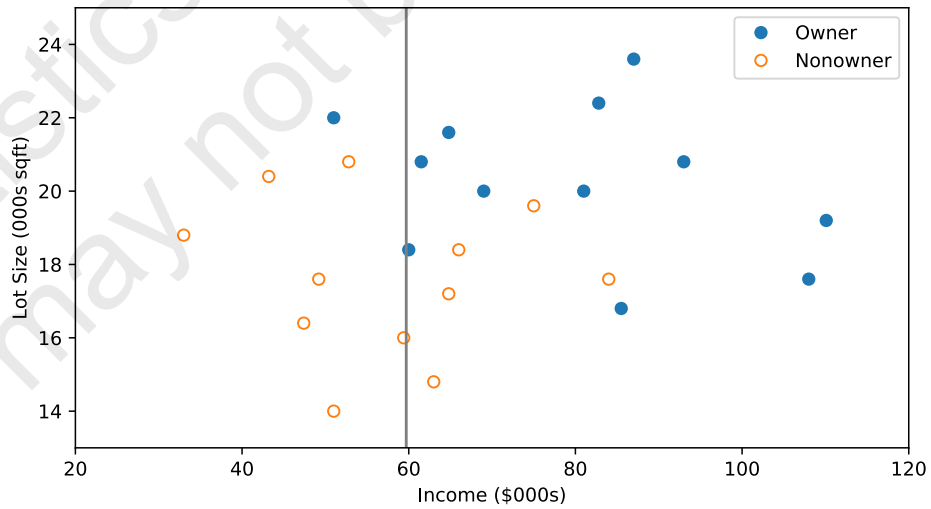

FIGURE 9.3 SPLITTING THE 24 RECORDS BY INCOME VALUE OF 59.7

Statistics.com PA 3041219 may not be redistributed

is now divided into two rectangles, one with Income ≤ 59.7 and the other with Income > 59.7. This is illustrated in Figure 9.3.

Notice how the split has created two rectangles, each of which is much more homogeneous than the rectangle before the split. The left rectangle contains points that are mostly nonowners (seven nonowners and one owner) and the right rectangle contains mostly owners (11 owners and five nonowners).

How was this particular split selected? The algorithm examined each predictor variable (in this case, Income and Lot Size) and all possible split values for each variable to find the best split. What are the possible split values for a variable? They are simply the midpoints between pairs of consecutive values for the predictor. The possible split points for Income are $\{38.1, 45.3, 50.1, \ldots, 109.5\}$ and those for Lot Size are $\{14.4, 15.4, 16.2, \ldots, 23\}$. These split points are ranked according to how much they reduce impurity (heterogeneity) in the resulting rectangle. A pure rectangle is one that is composed of a single class (e.g., owners). The reduction in impurity is defined as overall impurity before the split minus the sum of the impurities for the two rectangles that result from a split.

Categorical Predictors The previous description used numerical predictors; however, categorical predictors can also be used in the recursive partitioning context. To handle categorical predictors, the split choices for a categorical predictor are all ways in which the set of categories can be divided into two subsets. For example, a categorical variable with four categories, say $\{a, b, c, d\}$, can be split in seven ways into two subsets: $\{a\}$ and $\{b, c, d\}$; $\{b\}$ and $\{a, c, d\}$; $\{c\}$ and $\{a, b, d\}$; $\{d\}$ and $\{a, b, c\}$; $\{a, b\}$ and $\{c, d\}$; $\{a, c\}$ and $\{b, d\}$; and finally $\{a, d\}$ and $\{b, c\}$. When the number of categories is large, the number of splits becomes very large. As with k-nearest-neighbors, a predictor with m categories ($m > 2$) should be factored into m dummies (not $m - 1$).

Normalization Whether predictors are numerical or categorical, it does not make any difference if they are standardized (normalized) or not.

Measures of Impurity

There are a number of ways to measure impurity. The two most popular measures are the *Gini index* and an *entropy measure*. We describe both next. Denote the m classes of the response variable by $k = 1, 2, \ldots, m$.

The Gini impurity index for a rectangle A is defined by

$$I(A) = 1 - \sum_{k=1}^{m} p_k^2,$$

where p_k is the proportion of records in rectangle A that belong to class k. This measure takes values between 0 (when all the records belong to the same class)

Statistics.com PA 5841219 may not be reproduced

and $(m-1)/m$ (when all m classes are equally represented). Figure 9.4 shows the values of the Gini index for a two-class case as a function of p_k. It can be seen that the impurity measure is at its peak when $p_k = 0.5$ (i.e., when the rectangle contains 50% of each of the two classes).

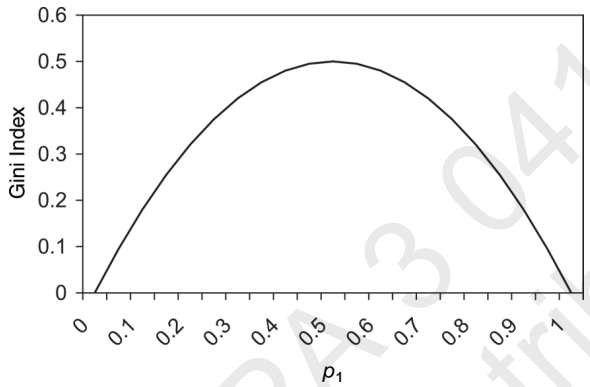

FIGURE 9.4 VALUES OF THE GINI INDEX FOR A TWO-CLASS CASE AS A FUNCTION OF THE PROPORTION OF RECORDS IN CLASS 1 (p_1)

A second impurity measure is the entropy measure. The entropy for a rectangle A is defined by

$$\text{entropy}(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

(to compute $\log_2(x)$ in Python, use function *math.log2(x)*). This measure ranges between 0 (most pure, all records belong to the same class) and $\log_2(m)$ (when all m classes are represented equally). In the two-class case, the entropy measure is maximized (like the Gini index) at $p_k = 0.5$.

Let us compute the impurity in the riding mower example before and after the first split (using Income with the value of 59.7). The unsplit dataset contains 12 owners and 12 nonowners. This is a two-class case with an equal number of records from each class. Both impurity measures are therefore at their maximum value: Gini = 0.5 and entropy = $\log_2(2) = 1$. After the split, the left rectangle contains seven nonowners and one owner. The impurity measures for this rectangle are:

$$\text{gini\_left} = 1 - (7/8)^2 - (1/8)^2 = 0.219$$
$$\text{entropy\_left} = -(7/8)\log_2(7/8) - (1/8)\log_2(1/8) = 0.544$$

The right node contains 11 owners and five nonowners. The impurity measures of the right node are therefore

Statistics.com DA 3 041219 may not be redistributed

$$\text{gini\_right} = 1 - (11/16)^2 - (5/16)^2 = 0.430$$
$$\text{entropy\_right} = -(11/16)\log_2(11/16) - (5/16)\log_2(5/16) = 0.896$$

The combined impurity of the two nodes created by the split is a weighted average of the two impurity measures, weighted by the number of records in each:

$$\text{gini} = (8/24)(0.219) + (16/24)(0.430) = 0.359$$
$$\text{entropy} = (8/24)(0.544) + (16/24)(0.896) = 0.779$$

Thus, the Gini impurity index decreased from 0.5 before the split to 0.359 after the split. Similarly, the entropy impurity measure decreased from 1 before the split to 0.779 after the split.

By comparing the reduction in impurity across all possible splits in all possible predictors, the next split is chosen. If we continue splitting the mower data, the next split is on the Lot Size variable at the value 21.4. Figure 9.5 shows that once again the tree procedure has astutely chosen to split a rectangle to increase the purity of the resulting rectangles. The lower-left rectangle, which contains records with Income $\leq$ 59.7 and Lot Size $\leq$ 21.4, has all records that are nonowners; whereas the upper-left rectangle, which contains records with Income $\leq$ 59.7 and Lot Size $>$ 21.4, consists exclusively of a single owner. In other words, the two left rectangles are now "pure."

We can see how the recursive partitioning is refining the set of constituent rectangles to become purer as the algorithm proceeds. The final stage of the recursive partitioning is shown in Figure 9.6.

Notice that each rectangle is now pure: it contains data points from just one of the two classes.

The reason the method is called a *classification tree algorithm* is that each split can be depicted as a split of a node into two successor nodes. The first split is shown as a branching of the root node of a tree in Figure 9.7. The full-grown tree is shown in Figure 9.9.

Statistics.com PA 3 041219 may not be redistributed

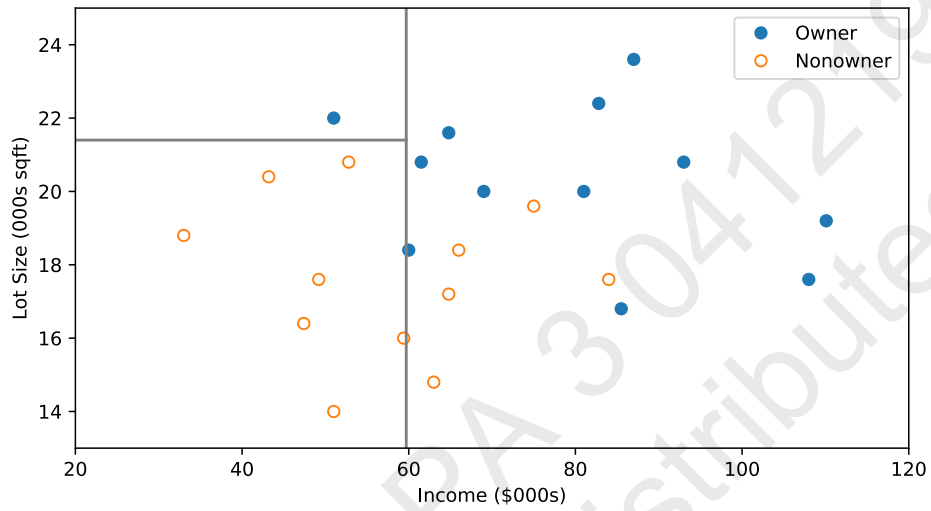

FIGURE 9.5 SPLITTING THE 24 RECORDS FIRST BY INCOME VALUE OF 59.7 AND THEN LOT SIZE VALUE OF 21.4

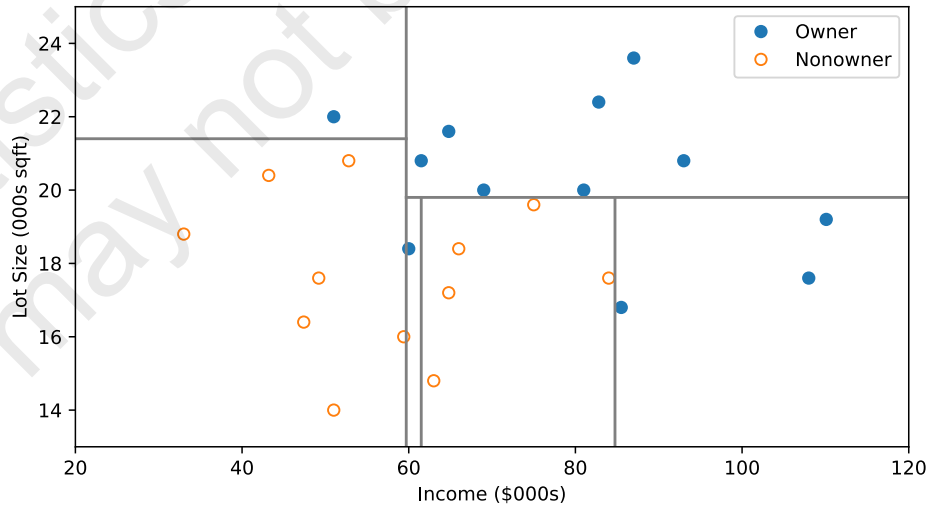

FIGURE 9.6 FINAL STAGE OF RECURSIVE PARTITIONING; EACH RECTANGLE CONSISTING OF A SINGLE CLASS (OWNERS OR NONOWNERS)

Statistics.com PA 3041219 may not be redistributed

code for running and plotting classification tree

```
mower_df = pd.read_csv('RidingMowers.csv')
# use max_depth to control tree size (None = full tree)
classTree = DecisionTreeClassifier(random_state=0, max_depth=1)
classTree.fit(mower_df.drop(columns=['Ownership']), mower_df['Ownership'])
print("Classes: {}".format(', '.join(classTree.classes_)))
plotDecisionTree(classTree, feature_names=mower_df.columns[:2], class_names=classTree.classes_))
```

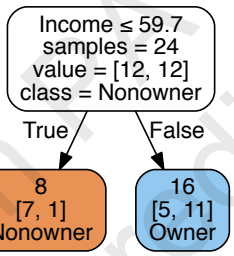

FIGURE 9.7 TREE REPRESENTATION OF FIRST SPLIT (CORRESPONDS TO FIGURE 9.3)

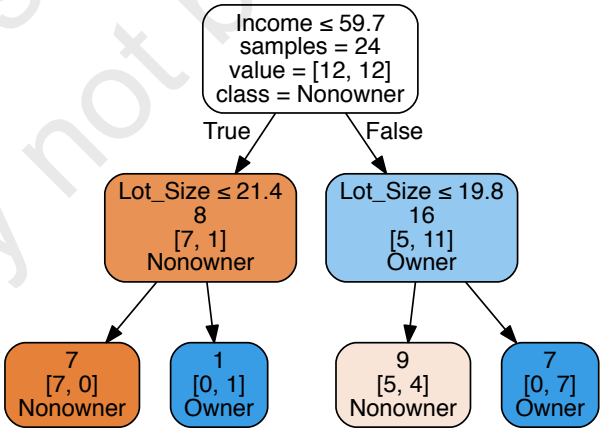

FIGURE 9.8 TREE REPRESENTATION OF FIRST THREE SPLITS.

Statistics.com ID 3041219 may not be redistributed

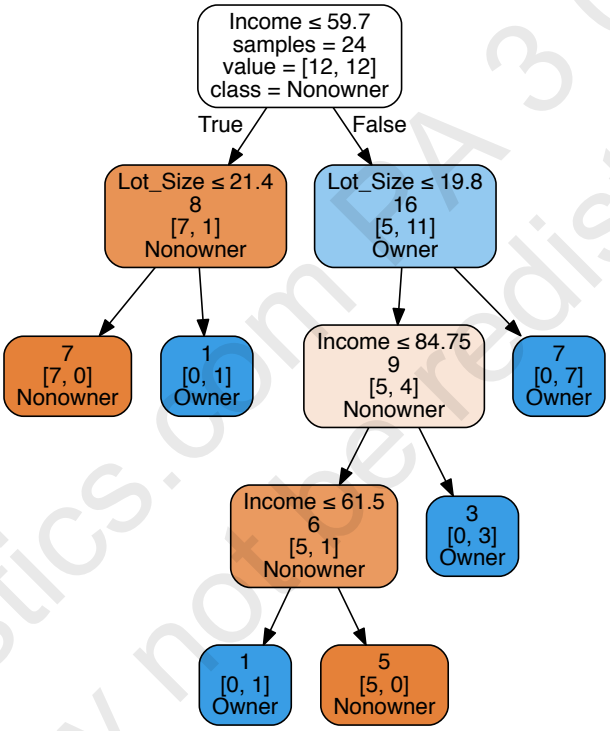

FIGURE 9.9 TREE REPRESENTATION AFTER ALL SPLITS (CORRESPONDS TO FIGURE 9.6). THIS IS THE FULL GROWN TREE

Statistics.com LLC 3 041219
may not be redistributed

9.3 Evaluating the Performance of a Classification Tree

We have seen with previous methods that the modeling job is not completed by fitting a model to training data; we need out-of-sample data to assess and tune the model. This is particularly true with classification and regression trees, for two reasons:

- Tree structure can be quite unstable, shifting substantially depending on the sample chosen.
- A fully-fit tree will invariably lead to overfitting.

To visualize the first challenge, potential instability, imagine that we partition the data randomly into two samples, A and B, and we build a tree with each. If there are several predictors of roughly equal predictive power, you can see that it would be easy for samples A and B to select different predictors for the top level split, just based on which records ended up in which sample. And a different split at the top level would likely cascade down and yield completely different sets of rules. So we should view the results of a single tree with some caution.

To illustrate the second challenge, overfitting, let's examine another example.

Example 2: Acceptance of Personal Loan

Universal Bank is a relatively young bank that is growing rapidly in terms of overall customer acquisition. The majority of these customers are liability customers with varying sizes of relationship with the bank. The customer base of asset customers is quite small, and the bank is interested in growing this base rapidly to bring in more loan business. In particular, it wants to explore ways of converting its liability (deposit) customers to personal loan customers.

A campaign the bank ran for liability customers showed a healthy conversion rate of over 9% successes. This has encouraged the retail marketing department to devise smarter campaigns with better target marketing. The goal of our analysis is to model the previous campaign's customer behavior to analyze what combination of factors make a customer more likely to accept a personal loan. This will serve as the basis for the design of a new campaign.

Our predictive model will be a classification tree. To assess the accuracy of the tree in classifying new records, we start with the tools and criteria discussed in Chapter 5—partitioning the data into training and validation sets, and later introduce the idea of *cross-validation*.

The bank's dataset includes data on 5000 customers. The data include customer demographic information (age, income, etc.), customer response to the last personal loan campaign (*Personal Loan*), and the customer's relationship with the bank (mortgage, securities account, etc.). Table 9.2 shows a sample of the

Statistics.com PA - 04/12/19 may not be reproduced

Statistics.com PA 3041219
may not be redistributed

TABLE 9.2 SAMPLE OF DATA FOR 20 CUSTOMERS OF UNIVERSAL BANK

ID	Age	Professional Experience	Income	Family Size	CC Avg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online Banking	Credit Card
1	25	1	49	4	1.60	UG	0	No	Yes	No	No	No
2	45	19	34	3	1.50	UG	0	No	Yes	No	No	No
3	39	15	11	1	1.00	UG	0	No	No	No	No	No
4	35	9	100	1	2.70	Grad	0	No	No	No	No	No
5	35	8	45	4	1.00	Grad	0	No	No	No	No	Yes
6	37	13	29	4	0.40	Grad	155	No	No	No	Yes	No
7	53	27	72	2	1.50	Grad	0	No	No	No	Yes	No
8	50	24	22	1	0.30	Prof	0	No	No	No	No	Yes
9	35	10	81	3	0.60	Grad	104	No	No	No	Yes	No
10	34	9	180	1	8.90	Prof	0	Yes	No	No	No	No
11	65	39	105	4	2.40	Prof	0	No	No	No	No	No
12	29	5	45	3	0.10	Grad	0	No	No	No	Yes	No
13	48	23	114	2	3.80	Prof	0	No	Yes	No	No	No
14	59	32	40	4	2.50	Grad	0	No	No	No	Yes	No
15	67	41	112	1	2.00	UG	0	No	Yes	No	No	No
16	60	30	22	1	1.50	Prof	0	No	No	No	Yes	Yes
17	38	14	130	4	4.70	Prof	134	Yes	No	No	No	No
18	42	18	81	4	2.40	UG	0	No	No	No	No	No
19	46	21	193	2	8.10	Prof	0	Yes	No	No	No	No
20	55	28	21	1	0.50	Grad	0	No	Yes	No	No	Yes

bank's customer database for 20 customers, to illustrate the structure of the data. Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign.

After randomly partitioning the data into training (3000 records) and validation (2000 records), we use the training data to construct a tree. The result is shown in Figure 9.12 – this is the default tree produced by *DecisionTreeClassifier()* for these data. Although it is difficult to see the exact splits, we note that the top decision node refers to all the records in the training set, of which 2704 customers did not accept the loan and 296 customers accepted the loan. The "class=0" in the top node represents the majority class (nonacceptors). The first split, which is on the Income variable, generates left and right child nodes. To the left is the child node with customers who have income of 110.5 or lower. Customers with income greater than 110.5 go to the right. The splitting process continues; where it stops depends on the parameter settings of the algorithm. The eventual classification of customer appears in the terminal nodes. Of the 43 terminal nodes, 24 lead to classification of "nonacceptor" and 19 lead to classification of "acceptor."

The default tree has no restrictions on the maximum depth of the tree (or number of leaf nodes), nor on the magnitude of decrease in impurity measure. These default values for the parameters controlling the size of the tree lead to a fully grown tree, with pure leaf nodes.

Let us assess the performance of this full tree with the validation data. Each record in the validation data is "dropped down" the tree and classified according to the terminal node it reaches. These predicted classes can then be compared to the actual outcome via a confusion matrix. When a particular class is of interest, a lift chart is useful for assessing the model's ability to capture those records. Table 9.3 shows the confusion matrices for the full grown tree. We see that the training data are perfectly classified (accuracy=1), whereas the validation data have a lower accuracy (0.98).

Sensitivity Analysis Using Cross Validation

Due to the issue of tree instability, it is possible that the performance results will differ markedly when using a different partitioning into training and validation data. It is therefore useful to use cross-validation to evaluate the variability in performance on different data partitioning (see Section 2.5 in Chapter 2). Table 9.4 shows code and the result of applying Python's *cross_val_score* method to perform 5-fold cross-validation on the full grown (default) tree. We see that the validation accuracy changes significantly across different folds from 0.972 to 0.992.

Statistics.com PA 3 04.2019 may not be redistributed

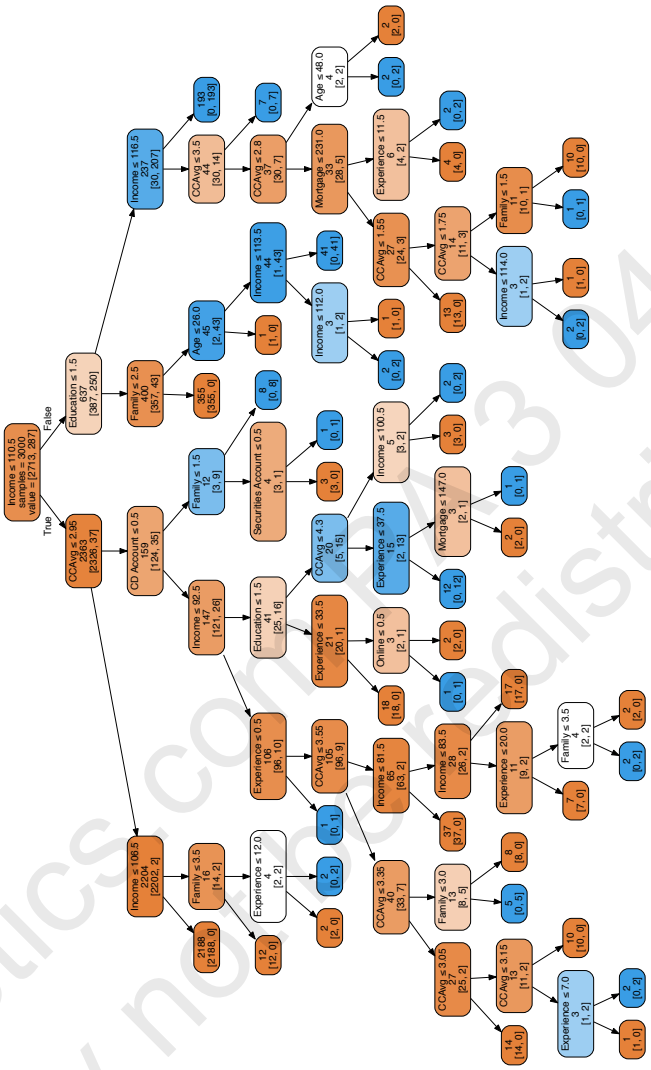

FIGURE 9.10 A FULL TREE FOR THE LOAN ACCEPTANCE DATA USING THE TRAINING SET (3000 RECORDS)

Statistics content © 2013 04/2019 may not be redistributed

code for creating a full-grown classification tree

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)

X = bank_df.drop(columns=['Personal Loan'])
y = bank_df['Personal Loan']
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

fullClassTree = DecisionTreeClassifier(random_state=1)
fullClassTree.fit(train_X, train_y)

plotDecisionTree(fullClassTree, feature_names=train_X.columns)
```

TABLE 9.3 CONFUSION MATRICES AND ACCURACY FOR THE DEFAULT (FULL) CLASSIFICATION TREE, ON THE TRAINING AND VALIDATION SETS OF THE PERSONAL LOAN DATA

code for classifying the validation data using a tree and computing the confusion matrices and accuracy for the training and validation data

```
classificationSummary(train_y, fullClassTree.predict(train_X))
classificationSummary(valid_y, fullClassTree.predict(valid_X))
```

Output

```
# full tree: training
Confusion Matrix (Accuracy 1.0000)

       Prediction
Actual    0    1
     0 2727    0
     1    0  273

# full tree: validation
Confusion Matrix (Accuracy 0.9790)

       Prediction
Actual    0    1
     0 1790   17
     1   25  168
```

TABLE 9.4 ACCURACY OF THE DEFAULT (FULL) CLASSIFICATION TREE, ON FIVE VALIDATION FOLDS USING 5-FOLD CROSS-VALIDATION

code for computing validation accuracy using 5-fold cross-validation on the full tree

```
treeClassifier = DecisionTreeClassifier(random_state=1)

scores = cross_val_score(treeClassifier, train_X, train_y, cv=5)
print('Accuracy scores of each fold: ', [f'{acc:.3f}' for acc in scores])
```

Output

```
Accuracy scores of each fold:  ['0.985', '0.972', '0.992', '0.987', '0.992']
```

Statistics.com PA 3 041219 may not be redistributed

9.4 Avoiding Overfitting

One danger in growing deep trees on the training data is overfitting. As discussed in Chapter 5, overfitting will lead to poor performance on new data. If we look at the overall error at the various sizes of the tree, it is expected to decrease as the number of terminal nodes grows until the point of overfitting. Of course, for the training data the overall error decreases more and more until it is zero at the maximum level of the tree. However, for new data, the overall error is expected to decrease until the point where the tree fully models the relationship between class and the predictors. After that, the tree starts to model the noise in the training set, and we expect the overall error for the validation set to start increasing. This is depicted in Figure 9.11. One intuitive reason a large tree may overfit is that its final splits are based on very small numbers of records. In such cases, class difference is likely to be attributed to noise rather than predictor information.

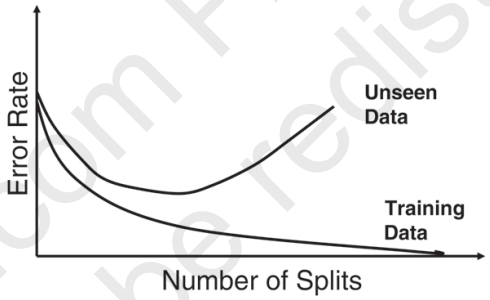

FIGURE 9.11 ERROR RATE AS A FUNCTION OF THE NUMBER OF SPLITS FOR TRAINING VS. VALIDATION DATA: OVERFITTING

Stopping Tree Growth

One can think of different criteria for stopping the tree growth before it starts overfitting the data. Examples are tree depth (i.e., number of splits), minimum number of records in a terminal node, and minimum reduction in impurity. In Python's *DecisionTreeClassifier()*, we can control the depth of the tree, the minimum number of records in a node needed in order to split, the mimimum number of records in a terminal node, etc. The problem is that it is not simple to determine what is a good stopping point using such rules.

Let us return to the personal loan example, this time restricting tree depth, the minimum number of records in a node required for splitting, and the minimum impurity decrease required.[1] The code and resulting tree are shown in

[1]The parameter values chosen are the default values used by the R function *rpart()*

Statistics.com PA 3 04.12.19 may not be redistributed

code for creating a smaller classification tree

```
smallClassTree = DecisionTreeClassifier(max_depth=30, min_samples_split=20,
                        min_impurity_decrease=0.01, random_state=1)
smallClassTree.fit(train_X, train_y)

plotDecisionTree(smallClassTree, feature_names=train_X.columns)
```

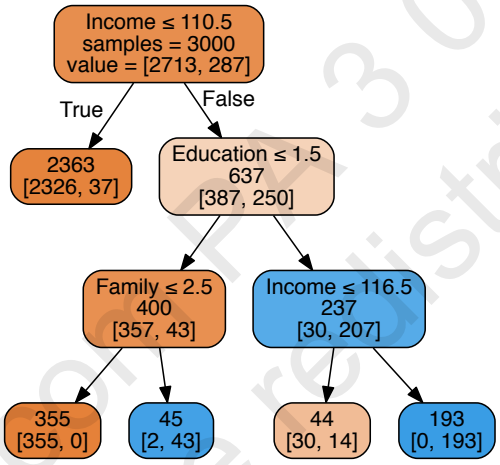

FIGURE 9.12 SMALLER CLASSIFICATION TREE FOR THE LOAN ACCEPTANCE DATA USING THE TRAINING SET (3000 RECORDS)

Statistics.com PA 3 041219 may not be redistributed

Figure 9.12. The resulting tree has 4 terminal nodes, where two are pure but two are not. The smallest terminal node has 44 records.

Table 9.5 displays the confusion matrices for the training and validation sets of the smaller tree. Compared to the full grown tree, we see that the full tree has higher training accuracy: it is 100% accurate in classifying the training data, which means it has completely pure terminal nodes. The smaller tree achieves the same validation performance, but importantly, it has a smaller gap in performance between training and validation, whereas the full tree's training-vs-validation performance gap is bigger. The main reason is that the full-grown tree overfits the training data (to perfect accuracy!).

TABLE 9.5 CONFUSION MATRICES AND ACCURACY FOR THE SMALL CLASSIFICATION TREE, ON THE TRAINING AND VALIDATION SETS OF THE PERSONAL LOAN DATA

code for classifying the validation data using a small tree and computing the confusion matrices and accuracy for the training and validation data

```
classificationSummary(train_y, smallClassTree.predict(train_X))
classificationSummary(valid_y, smallClassTree.predict(valid_X))
```

Output

```
# small tree: training
Confusion Matrix (Accuracy 0.9823)

       Prediction
Actual    0    1
     0 2711    2
     1   51  236

# small tree: validation
Confusion Matrix (Accuracy 0.9770)

       Prediction
Actual    0    1
     0 1804    3
     1   43  150
```

Statistics.com PA 3 041219 may not be redistributed

Fine-tuning Tree Parameters

As mentioned earlier, the challenge with using tree growth stopping rules such as maximum tree depth is that it is not simple to determine what is a good stopping point using such rules. A solution is to use grid search over combinations of different parameter values. For example, we might want to search for trees with {maximum depths in the range of [5, 30], minimum number of records in terminal nodes between [20, 100], impurity criterion decrease in the range [0.001, 0.01]}. We can use an exhaustive grid search to find the combination that leads to the tree with smallest error (highest accuracy).

If we use the training set to find the tree with the lowest error among all the trees in the searched range, measuring accuracy on that same training data, then we will be overfitting the training data. If we use the validation set to measure accuracy, then, with the numerous grid search trials, we will be overfitting the validation data! A solution is therefore to use cross-validation on the training set, and, after settling on the best tree, use that tree with the validation data to evaluate likely actual performance with new data. This will help detect and avoid possible overfitting.

In Python, an exhaustive grid search of this type can be achieved using *GridSearchCV()*. Table 9.6 shows the code and result of using this method with

TABLE 9.6 EXHAUSTIVE GRID SEARCH TO FINE TUNE METHOD PARAMETERS

code for using GridSearchCV to fine tune method parameters

```
# Start with an initial guess for parameters
param_grid = {
    'max_depth': [10, 20, 30, 40],
    'min_samples_split': [20, 40, 60, 80, 100],
    'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
}
gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5,
                          n_jobs=-1)  # n_jobs=-1 will utilize all available CPUs
gridSearch.fit(train_X, train_y)
print('Initial score: ', gridSearch.best_score_)
print('Initial parameters: ', gridSearch.best_params_)

# Adapt grid based on result from initial grid search
param_grid = {
    'max_depth': list(range(2, 16)),  # 14 values
    'min_samples_split': list(range(10, 22)), # 11 values
    'min_impurity_decrease': [0.0009, 0.001, 0.0011], # 3 values
}
gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5,
                          n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Improved score: ', gridSearch.best_score_)
print('Improved parameters: ', gridSearch.best_params_)

bestClassTree = gridSearch.best_estimator_
```

Output

```
Initial score:  0.988
Initial parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.001,
                      'min_samples_split': 20}

Improved score:  0.9883333333333333
Improved parameters:  {'max_depth': 5, 'min_impurity_decrease': 0.0011,
                       'min_samples_split': 13}
```

Statistics.com PA 3 041219 may not be redistributed

code for plotting and evaluating performance of fine-tuned classification tree

Evaluating performance

```
# fine-tuned tree: training
> classificationSummary(train_y, bestClassTree.predict(train_X))
Confusion Matrix (Accuracy 0.9900)

       Prediction
Actual    0    1
     0 2703   10
     1   20  267

# fine-tuned tree: validation
> classificationSummary(valid_y, bestClassTree.predict(valid_X))
Confusion Matrix (Accuracy 0.9825)

       Prediction
Actual    0    1
     0 1793   14
     1   21  172
```

Plotting tree

```
plotDecisionTree(bestClassTree, feature_names=train_X.columns)
```

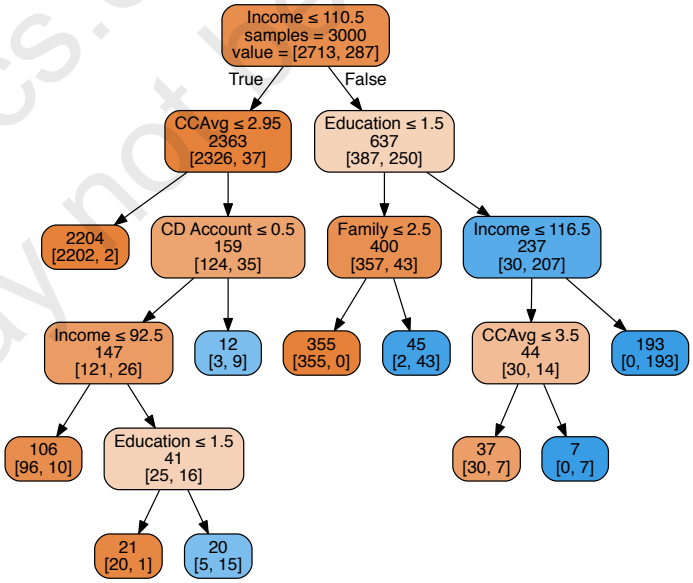

FIGURE 9.13 FINE-TUNED CLASSIFICATION TREE FOR THE LOAN ACCEPTANCE DATA USING THE TRAINING SET (3000 RECORDS)

Statistics.com PA 3 041219 may not be redistributed

5-fold cross-validation. The resulting parameter combination is *max_depth* 5, *min_impurity_decrease* 0.0011, and *min_samples_split* 13. The final tree and the model performance on the training and validation sets is shown in Figure 9.13.

The exhaustive grid search can quickly become very time consuming. In our example, the first grid search assessed $4 \times 5 \times 5 = 100$ combinations and the second $14 \times 11 \times 3 = 462$ combinations. With even more tune-able parameters the number of possible combinations can become very large. In this case it is better to use *RandomizedSearchCV()* which will randomly sample all possible combinations while limiting the total number of tries (parameter *n_iter*). With *RandomizedSearchCV()*, it is also possible to sample parameter from distributions (e.g., any number between 0 and 1) without listing them individually.

Other Methods for Limiting Tree Size

Several other methods for limiting tree size have been used widely. They are not implemented in Python at this writing, but we note them here for completeness.

CHAID CHAID stands for chi-squared automatic interaction detection, a recursive partitioning method that predates classification and regression tree (CART) procedures by several years and is widely used in database marketing applications to this day. It uses a well-known statistical test (the chi-square test for independence) to assess whether splitting a node improves the purity by a statistically significant amount. In particular, at each node, we split on the predictor with the strongest association with the outcome variable. The strength of association is measured by the p-value of a chi-squared test of independence. If for the best predictor the test does not show a significant improvement, the split is not carried out, and the tree is terminated. A more general class of trees based on this idea is called *conditional inference trees* (see Hothorn et al., 2006).

Pruning The idea behind pruning is to recognize that a very large tree is likely to overfit the training data, and that the smallest branches, which are the last to be grown and are furthest from the trunk, are likely fitting noise in the training data. They may actually contribute to error in fitting new data, so they are lopped off. Pruning the full-grown tree is the basis of the popular CART method (developed by Breiman et al., implemented in multiple data mining software packages such as R's `rpart` package, SAS Enterprise Miner, CART, and MARS) and C4.5 (developed by Quinlan and implemented in packages such as IBM SPSS Modeler). In C4.5, the training data are used both for growing and pruning the tree. In CART, the innovation is to use the validation data to prune back the tree that is grown from training data.

More specifically, the CART algorithm uses a cost-complexity function that balances tree size (complexity) and misclassification error (cost) in order

Statistics.com PA 3 04/12/16 may not be redistributed

to choose tree size. The cost complexity of a tree is equal to its misclassification error (based on the training data) plus a penalty factor for the size of the tree. For a tree T that has $L(T)$ terminal nodes, the cost complexity can be written as

$$CC(T) = \text{err}(T) + \alpha L(T),$$

where $\text{err}(T)$ is the fraction of training records that are misclassified by tree T and α is a penalty factor ("complexity parameter") for tree size. When $\alpha = 0$, there is no penalty for having too many nodes in a tree, and this yields the full-grown unpruned tree. When we increase α to a very large value the penalty cost component swamps the misclassification error component of the cost complexity criterion, and the result is simply the tree with the fewest terminal nodes: namely, the tree with one node. So there is a range of trees, from tiny to large, corresponding to a range of α, from large to small. From this sequence of trees it seems natural to choose the one that gave the lowest misclassification error on the validation dataset. Alternatively, rather than relying on a single validation partition, we can use k-fold cross-validation for choosing α.

9.5 Classification Rules from Trees

As described in Section 9.1, classification trees provide easily understandable *classification rules* (if the trees are not too large). Each terminal node is equivalent to a classification rule. Returning to the example, the left-most terminal node in the fine-tuned tree (Figure 9.13) gives us the rule

IF (*Income* $\leq$ 110.5) AND (*CCAvg* $\leq$ 2.95)
THEN *Class* = 0.

However, in many cases, the number of rules can be reduced by removing redundancies. For example, consider the rule from the second-from-bottom-left-most terminal node in Figure 9.13:

IF (*Income* $\leq$ 110.5) AND (*CCAvg* $>$ 2.95) AND (*CD.Account* $\leq$ *0.5*)
AND (*Income* $\leq$ 92.5)
THEN *Class* = 0

This rule can be simplified to

IF (*Income* $\leq$ 92.5) AND (*CCAvg* $>$ 2.95) AND (*CD.Account* $\leq$ *0.5*)
THEN *Class* = 0

This transparency in the process and understandability of the algorithm that leads to classifying a record as belonging to a certain class is very advantageous in settings where the final classification is not the only thing of interest. Berry and Linoff (2000) give the example of health insurance underwriting, where the

Statistics.com PA 3 04/12/19 may not be redistributed

insurer is required to show that coverage denial is not based on discrimination. By showing rules that led to denial (e.g., income $<$ \$20K AND low credit history), the company can avoid law suits. Compared to the output of other classifiers, such as discriminant functions, tree-based classification rules are easily explained to managers and operating staff. Their logic is certainly far more transparent than that of weights in neural networks!

9.6 Classification Trees for More Than Two Classes

Classification trees can be used with an outcome variable that has more than two classes. In terms of measuring impurity, the two measures presented earlier (the Gini impurity index and the entropy measure) were defined for m classes and hence can be used for any number of classes. The tree itself would have the same structure, except that its terminal nodes would take one of the m–class labels.

9.7 Regression Trees

The tree method can also be used for a numerical outcome variable. Regression trees for prediction operate in much the same fashion as classification trees. The outcome variable (Y) is a numerical variable in this case, but both the principle and the procedure are the same: Many splits are attempted, and for each, we measure "impurity" in each branch of the resulting tree. The tree procedure then selects the split that minimizes the sum of such measures. To illustrate a regression tree, consider the example of predicting prices of Toyota Corolla automobiles (from Chapter 6). The dataset includes information on 1000 sold Toyota Corolla cars. (We use the first 1000 cars from the dataset *ToyotoCorolla.csv*). The goal is to find a predictive model of price as a function of 10 predictors (including mileage, horsepower, number of doors, etc.). A regression tree for these data was built using a training set of 600 records. The fine-tuned tree is shown in Figure 9.14. Code for training and evaluating the regression tree is given in Table 9.7.

We see that the top three levels of the regression tree are dominated by age of the car, its weight, and mileage. In the lower levels we can see that individual details of a car, like powered windows, lead to smaller adjustments of the price.

Three details differ between regression trees and classification trees: prediction, impurity measures, and evaluating performance. We describe these next.

Statistics.com DRAFT 3 041219 may not be redistributed

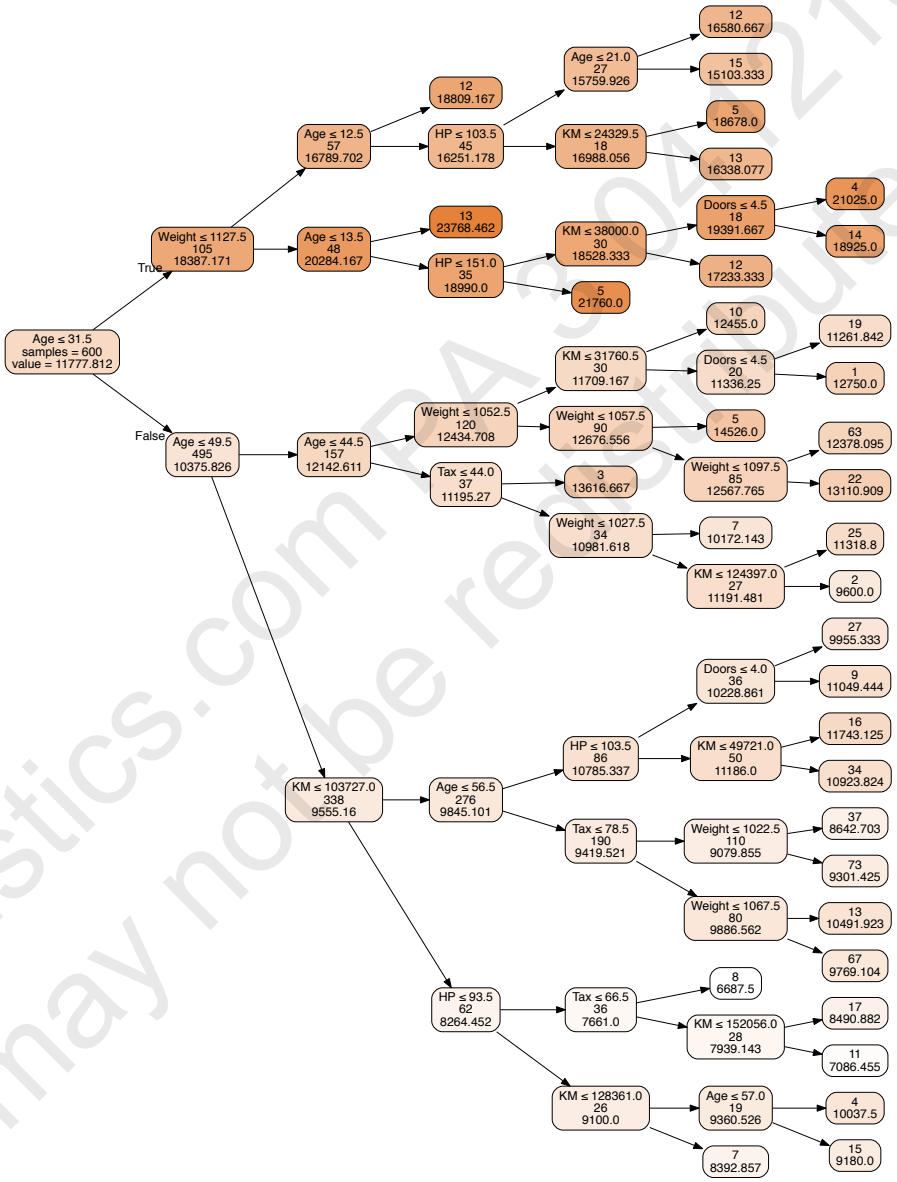

FIGURE 9.14 FINE-TUNED REGRESSION TREE FOR TOYOTA COROLLA PRICES

Statistics.com DA 3041 219
may not be reproduced

TABLE 9.7 TRAIN AND EVALUATE A FINE-TUNED REGRESSION TREE

code for building a regression tree

```
from sklearn.tree import DecisionTreeRegressor
toyotaCorolla_df = pd.read_csv('ToyotaCorolla.csv').iloc[:1000,:]
toyotaCorolla_df = toyotaCorolla_df.rename(columns={'Age_08_04': 'Age', 'Quarterly_Tax': 'Tax'})

predictors = ['Age', 'KM', 'Fuel_Type', 'HP', 'Met_Color', 'Automatic', 'CC',
              'Doors', 'Tax', 'Weight']
outcome = 'Price'

X = pd.get_dummies(toyotaCorolla_df[predictors], drop_first=True)
y = toyotaCorolla_df[outcome]

train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

# user grid search to find optimized tree
param_grid = {
    'max_depth': [5, 10, 15, 20, 25],
    'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
    'min_samples_split': [10, 20, 30, 40, 50],
}
gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Initial parameters: ', gridSearch.best_params_)

param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    'min_impurity_decrease': [0, 0.001, 0.002, 0.003, 0.005, 0.006, 0.007, 0.008],
    'min_samples_split': [14, 15, 16, 18, 20, ],
}
gridSearch = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=5, n_jobs=-1)
gridSearch.fit(train_X, train_y)
print('Improved parameters: ', gridSearch.best_params_)

regTree = gridSearch.best_estimator_

regressionSummary(train_y, regTree.predict(train_X))
regressionSummary(valid_y, regTree.predict(valid_X))
```

Output

```
Initial parameters:  {'max_depth': 10,'min_impurity_decrease': 0.001,'min_samples_split': 20}
Improved parameters: {'max_depth': 6,'min_impurity_decrease': 0.01,'min_samples_split': 12}

Regression statistics

                    Mean Error (ME) : 0.0000
     Root Mean Squared Error (RMSE) : 1058.8202
          Mean Absolute Error (MAE) : 767.7203
        Mean Percentage Error (MPE) : -0.8074
Mean Absolute Percentage Error (MAPE) : 6.8325

Regression statistics

                    Mean Error (ME) : 60.5241
     Root Mean Squared Error (RMSE) : 1554.9146
          Mean Absolute Error (MAE) : 1026.3487
        Mean Percentage Error (MPE) : -1.3082
Mean Absolute Percentage Error (MAPE) : 9.2311
```

Statistics.com PA 3041219 may not be redistributed

Prediction

Predicting the outcome value for a record is performed in a fashion similar to the classification case: We will use the truncated tree from Figure 9.14 to demonstrate how it works. The predictor information is used for "dropping" the record down the tree until reaching a terminal node. For instance, to predict the price of a Toyota Corolla with Age = 60, Mileage (KM) = 160,000, and Horse_Power (HP) = 100, we drop it down the tree and reach the node that has the value \$8392.857 (the lowest node). This is the price prediction for this car according to the tree. In classification trees, the value of the terminal node (which is one of the categories) is determined by the "voting" of the training records that were in that terminal node. In regression trees, the value of the terminal node is determined by the average outcome value of the training records that were in that terminal node. In the example above, the value \$8392.857 is the average price of the 7 cars in the training set that fall in the category of Age $>$ 49.5, KM $>$ 128361, and HP $>$ 93.5.

Measuring Impurity

We described two types of impurity measures for nodes in classification trees: the Gini index and the entropy-based measure. In both cases, the index is a function of the *ratio* between the categories of the records in that node. In regression trees, a typical impurity measure is the sum of the squared deviations from the mean of the terminal node. This is equivalent to the sum of the squared errors, because the mean of the terminal node is exactly the prediction. In the example above, the impurity of the node with the value \$8392.857 is computed by subtracting 8392.857 from the price of each of the 7 cars in the training set that fell in that terminal node, then squaring these deviations and summing them up. The lowest impurity possible is zero, when all values in the node are equal.

Evaluating Performance

As stated above, predictions are obtained by averaging the outcome values in the nodes. We therefore have the usual definition of predictions and errors. The predictive performance of regression trees can be measured in the same way that other predictive methods are evaluated (e.g., linear regression), using summary measures such as RMSE.

Statistics.com PA 15042019 may not be redistributed

9.8 Improving Prediction: Random Forests and Boosted Trees

Notwithstanding the transparency advantages of a single tree as described above, in a pure prediction application, where visualizing a set of rules does not matter, better performance is provided by several extensions to trees that combine results from multiple trees. These are examples of *ensembles* (see Chapter 13). One popular multitree approach is *random forests*, introduced by Breiman and Cutler.[2] Random forests are a special case of *bagging*, a method for improving predictive power by combining multiple classifiers or prediction algorithms. See Chapter 13 for further details on bagging.

Random Forests

The basic idea in random forests is to:

1. Draw multiple random samples, with replacement, from the data (this sampling approach is called the *bootstrap*).
2. Using a random subset of predictors at each stage, fit a classification (or regression) tree to each sample (and thus obtain a "forest").
3. Combine the predictions/classifications from the individual trees to obtain improved predictions. Use voting for classification and averaging for prediction.

The code and output in Table 9.8 illustrates applying a random forest in Python to the personal loan example. The validation accuracy of the random forest in this example (0.982) is similar to the single fine-tuned tree, and slightly higher than the single small tree that we fit earlier (0.977 - see Table 9.5).

Unlike a single tree, results from a random forest cannot be displayed in a tree-like diagram, thereby losing the interpretability that a single tree provides. However, random forests can produce "variable importance" scores, which measure the relative contribution of the different predictors. The importance score for a particular predictor is computed by summing up the decrease in the Gini index for that predictor over all the trees in the forest. Figure 9.15 shows the variable importance plot generated from the random forest model for the personal loan example. We see that Income and Education have the highest scores, with CCAvg being third. Importance scores for the other predictors are considerably lower.

[2]For further details on random forests, see https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm.

Statistics.com P A 3 04.1.19 may not be redistributed

TABLE 9.8 RANDOM FOREST (PERSONAL LOAN EXAMPLE)

code for running a random forest, plotting variable importance plot, and computing accuracy

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)

X = bank_df.drop(columns=['Personal Loan'])
y = bank_df['Personal Loan']
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=1)

rf = RandomForestClassifier(n_estimators=500, random_state=1)
rf.fit(train_X, train_y)

# variable (feature) importance plot
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)

df = pd.DataFrame({'feature': train_X.columns, 'importance': importances, 'std': std})
df = df.sort_values('importance')
print(df)

ax = df.plot(kind='barh', xerr='std', x='feature', legend=False)
ax.set_ylabel('')
plt.show()

# confusion matrix for validation set
classificationSummary(valid_y, rf.predict(valid_X))
```

Output

```
              feature  importance       std
7   Securities Account    0.003964  0.004998
9              Online    0.006394  0.005350
10         CreditCard    0.007678  0.007053
6            Mortgage    0.034243  0.023469
1          Experience    0.035539  0.016061
0                 Age    0.036258  0.015858
8           CD Account    0.057917  0.043185
3              Family    0.111375  0.053146
4               CCAvg    0.172105  0.103011
5           Education    0.200772  0.101002
2              Income    0.333756  0.129227

Confusion Matrix (Accuracy 0.9820)

       Prediction
Actual    0    1
     0 1803    4
     1   32  161
```

Statistics.com PA 3 041219 may not be redistributed

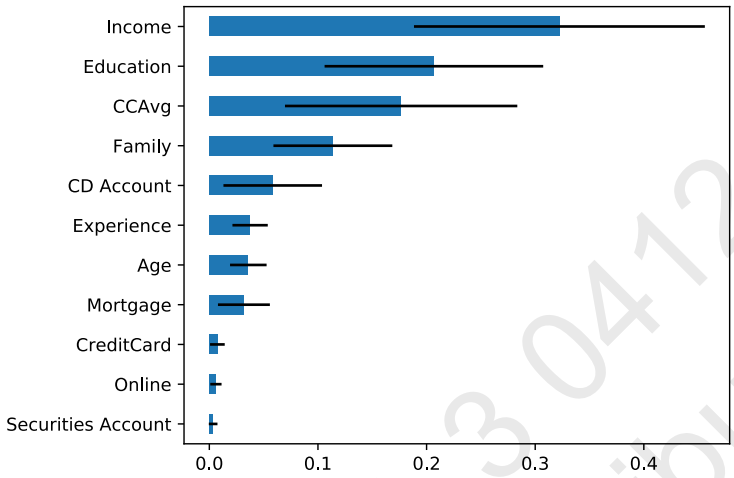

FIGURE 9.15 VARIABLE IMPORTANCE PLOT FROM RANDOM FOREST (PERSONAL LOAN EXAMPLE, FOR CODE SEE TABLE 9.8)

Boosted Trees

The second type of multitree improvement is *boosted trees*. Here a sequence of trees is fitted, so that each tree concentrates on misclassified records from the previous tree.

1. Fit a single tree.
2. Draw a sample that gives higher selection probabilities to misclassified records.
3. Fit a tree to the new sample.
4. Repeat Steps 2 and 3 multiple times.
5. Use weighted voting to classify records, with heavier weight for later trees.

Table 9.9 shows the result of running a boosted tree on the loan acceptance example that we saw earlier. We can see that compared to the performance of the single small tree (Table 9.5), the boosted tree has better performance on the validation data in terms of overall accuracy (0.9835) and especially in terms of correct classification of 1's—the rare class of special interest. Where does boosting's special talent for finding 1's come from? When one class is dominant (0's constitute over 90% of the data here), basic classifiers are tempted to classify cases as belonging to the dominant class, and the 1's in this case constitute most of the misclassifications with the single tree. The boosting algorithm concentrates on the misclassifications (which are mostly 1's), so it is naturally going to do well in reducing the misclassification of 1's (from 43 in the single small tree to 25 in the boosted tree, in the validation set).

Statistics.com PA 3 041219 may not be redistributed

TABLE 9.9 BOOSTED TREE: CONFUSION MATRIX FOR THE VALIDATION SET (LOAN DATA)

code for running boosted trees

```
boost = GradientBoostingClassifier()
boost.fit(train_X, train_y)
classificationSummary(valid_y, boost.predict(valid_X))
```

Output

```
Confusion Matrix (Accuracy 0.9835)

       Prediction
Actual    0    1
     0 1799    8
     1   25  168
```

9.9 ADVANTAGES AND WEAKNESSES OF A TREE

Tree methods are good off-the-shelf classifiers and predictors. They are also useful for variable selection, with the most important predictors usually showing up at the top of the tree. Trees require relatively little effort from users in the following senses: First, there is no need for transformation of variables (any monotone transformation of the variables will give the same trees). Second, variable subset selection is automatic since it is part of the split selection. In the loan example, note that out of the set of 14 predictors available, the small tree automatically selected only three predictors (Income, Education, and Family) and the fine-tuned tree selected five (Income, Education, Family, CCAvg, and CD Account).

Trees are also intrinsically robust to outliers, since the choice of a split depends on the *ordering* of values and not on the absolute *magnitudes* of these values. However, they are sensitive to changes in the data, and even a slight change can cause very different splits!

Unlike models that assume a particular relationship between the outcome and predictors (e.g., a linear relationship such as in linear regression and linear discriminant analysis), classification and regression trees are nonlinear and non-parametric. This allows for a wide range of relationships between the predictors and the outcome variable. However, this can also be a weakness: Since the splits are done on one predictor at a time, rather than on combinations of predictors, the tree is likely to miss relationships between predictors, in particular linear structures like those in linear or logistic regression models. Classification trees are useful classifiers in cases where horizontal and vertical splitting of the predictor space adequately divides the classes. But consider, for instance, a

Statistics.com PA 3 041219 may not be redistributed

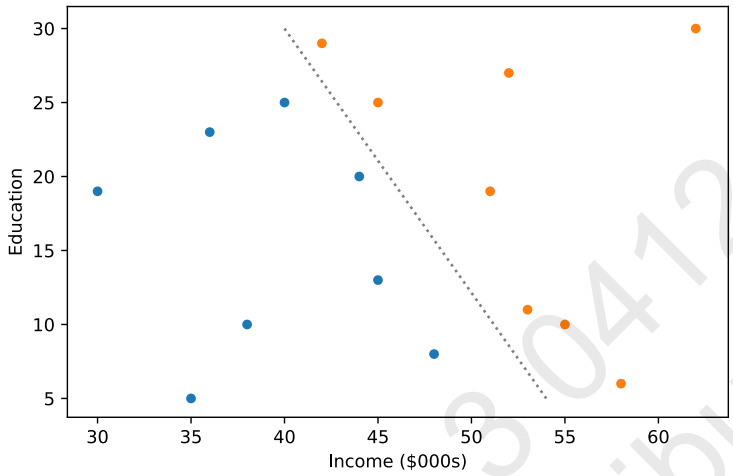

FIGURE 9.16 A TWO-PREDICTOR CASE WITH TWO CLASSES. THE BEST SEPARATION IS ACHIEVED WITH A DIAGONAL LINE, WHICH CLASSIFICATION TREES CANNOT DO

dataset with two predictors and two classes, where separation between the two classes is most obviously achieved by using a diagonal line (as shown in Figure 9.16). In such cases, a classification tree is expected to have lower performance than methods such as discriminant analysis. One way to improve performance is to create new predictors that are derived from existing predictors, which can capture hypothesized relationships between predictors (similar to interactions in regression models). Random forests are another solution in such situations.

Another performance issue with classification trees is that they require a large dataset in order to construct a good classifier. From a computational aspect, trees can be relatively expensive to grow, because of the multiple sorting involved in computing all possible splits on every variable. Methods for avoiding overfitting, such as cross-validation or pruning the data using the validation set, add further computation time.

Although trees are useful for variable selection, one challenge is that they "favor" predictors with many potential split points. This includes categorical predictors with many categories and numerical predictors with many different values. Such predictors have a higher chance of appearing in a tree. One simplistic solution is to combine multiple categories into a smaller set and bin numerical predictors with many values. Alternatively, some special algorithms avoid this problem by using a different splitting criterion [e.g., conditional inference trees in the R package `party`—see Hothorn et al. (2006)—and QUEST classification trees—see Loh and Shih (1997)].

An appealing feature of trees is that they handle missing data without having to impute values or delete records with missing values. Finally, a very important

Statistics.com PA 3 041219 may not be redistributed

practical advantage of trees is the transparent rules that they generate. Such transparency is often useful in managerial applications, though this advantage is lost in the ensemble versions of trees (random forests, boosted trees).

Statistics.com PA 3 041219
may not be redistributed

PROBLEMS

9.1 Competitive Auctions on eBay.com. The file *eBayAuctions.csv* contains information on 1972 auctions that transacted on eBay.com during May–June 2004. The goal is to use these data to build a model that will classify auctions as competitive or noncompetitive. A *competitive auction* is defined as an auction with at least two bids placed on the item auctioned. The data include variables that describe the item (auction category), the seller (his/her eBay rating), and the auction terms that the seller selected (auction duration, opening price, currency, day-of-week of auction close). In addition, we have the price at which the auction closed. The task is to predict whether or not the auction will be competitive.

Data Preprocessing. Convert variable *Duration* into a categorical variable. Split the data into training (60%) and validation (40%) datasets.

a. Fit a classification tree using all predictors. To avoid overfitting, set the minimum number of records in a terminal node to 50 and the maximum tree depth to 7. Write down the results in terms of rules. (*Note*: If you had to slightly reduce the number of predictors due to software limitations, or for clarity of presentation, which would be a good variable to choose?)

b. Is this model practical for predicting the outcome of a new auction?

c. Describe the interesting and uninteresting information that these rules provide.

d. Fit another classification tree (using a tree with a minimum number of records per terminal node = 50 and maximum depth = 7), this time only with predictors that can be used for predicting the outcome of a new auction. Describe the resulting tree in terms of rules. Make sure to report the smallest set of rules required for classification.

e. Plot the resulting tree on a scatter plot: Use the two axes for the two best (quantitative) predictors. Each auction will appear as a point, with coordinates corresponding to its values on those two predictors. Use different colors or symbols to separate competitive and noncompetitive auctions. Draw lines (you can sketch these by hand or use Python) at the values that create splits. Does this splitting seem reasonable with respect to the meaning of the two predictors? Does it seem to do a good job of separating the two classes?

f. Examine the lift chart and the confusion matrix for the tree. What can you say about the predictive performance of this model?

g. Based on this last tree, what can you conclude from these data about the chances of an auction obtaining at least two bids and its relationship to the auction settings set by the seller (duration, opening price, ending day, currency)? What would you recommend for a seller as the strategy that will most likely lead to a competitive auction?

9.2 Predicting Delayed Flights. The file *FlightDelays.csv* contains information on all commercial flights departing the Washington, DC area and arriving at New York during January 2004. For each flight, there is information on the departure and arrival airports, the distance of the route, the scheduled time and date of the flight, and so on. The variable that we are trying to predict is whether or not a flight is delayed. A delay is defined as an arrival that is at least 15 minutes later than scheduled.

Data Preprocessing. Transform variable day of week (DAY_WEEK) info a categorical variable. Bin the scheduled departure time into eight bins. Use these and all

Statistics.com PA 3 04/12/19 may not be distributed

other columns as predictors (excluding DAY_OF_MONTH). Partition the data into training (60%) and validation (40%) sets.

a. Fit a classification tree to the flight delay variable using all the relevant predictors. Do not include DEP_TIME (actual departure time) in the model because it is unknown at the time of prediction (unless we are generating our predictions of delays after the plane takes off, which is unlikely). Use a tree with maximum depth 8 and minimum impurity decrease = 0.01. Express the resulting tree as a set of rules.

b. If you needed to fly between DCA and EWR on a Monday at 7:00 AM, would you be able to use this tree? What other information would you need? Is it available in practice? What information is redundant?

c. Fit the same tree as in (a), this time excluding the Weather predictor. Display both the resulting (small) tree and the full-grown tree. You will find that the small tree contains a single terminal node.

i. How is the small tree used for classification? (What is the rule for classifying?)

ii. To what is this rule equivalent?

iii. Examine the full-grown tree. What are the top three predictors according to this tree?

iv. Why, technically, does the small tree result in a single node?

v. What is the disadvantage of using the top levels of the full-grown tree as opposed to the small tree?

vi. Compare this general result to that from logistic regression in the example in Chapter 10. What are possible reasons for the classification tree's failure to find a good predictive model?

9.3 Predicting Prices of Used Cars (Regression Trees). The file *ToyotaCorolla.csv* contains the data on used cars (Toyota Corolla) on sale during late summer of 2004 in the Netherlands. It has 1436 records containing details on 38 attributes, including Price, Age, Kilometers, HP, and other specifications. The goal is to predict the price of a used Toyota Corolla based on its specifications. (The example in Section 9.7 is a subset of this dataset).

Data Preprocessing. Split the data into training (60%), and validation (40%) datasets.

a. Run a full-grown regression tree (RT) with outcome variable Price and predictors Age_08_04, KM, Fuel_Type (first convert to dummies), HP, Automatic, Doors, Quarterly_Tax, Mfr_Guarantee, Guarantee_Period, Airco, Automatic_airco, CD_Player, Powered_Windows, Sport_Model, and Tow_Bar. Set random_state=1.

i. Which appear to be the three or four most important car specifications for predicting the car's price?

ii. Compare the prediction errors of the training and validation sets by examining their RMS error and by plotting the two boxplots. How does the predictive performance of the validation set compare to the training set? Why does this occur?

iii. How might we achieve better validation predictive performance at the expense of training performance?

Statistics.com PA 3 04 12 19
may not be redistributed

iv. Create a smaller tree by using GridSearchCV() with cv=5 to find a fine-tuned tree. Compared to the full-grown tree, what is the predictive performance on the validation set?

b. Let us see the effect of turning the price variable into a categorical variable. First, create a new variable that categorizes price into 20 bins. Now repartition the data keeping Binned_Price instead of Price. Run a classification tree with the same set of input variables as in the RT, and with Binned_Price as the output variable. As in the less deep regression tree, create a smaller tree by using GridSearchCV() with cv=5 to find a fine-tuned tree.

i. Compare the smaller tree generated by the CT with the smaller tree generated by RT. Are they different? (Look at structure, the top predictors, size of tree, etc.) Why?

ii. Predict the price, using the smaller RT and CT, of a used Toyota Corolla with the specifications listed in Table 9.10.

TABLE 9.10 SPECIFICATIONS FOR A PARTICULAR TOYOTA COROLLA

Variable	Value
Age_-08_-04	77
KM	117,000
Fuel_Type	Petrol
HP	110
Automatic	No
Doors	5
Quarterly_Tax	100
Mfg_Guarantee	No
Guarantee_Period	3
Airco	Yes
Automatic_airco	No
CD_Player	No
Powered_Windows	No
Sport_Model	No
Tow_Bar	Yes

iii. Compare the predictions in terms of the predictors that were used, the magnitude of the difference between the two predictions, and the advantages and disadvantages of the two methods.

Statistics.com PA 3041219 may not be redistributed

CHAPTER 13

Combining Methods: Ensembles and Uplift Modeling

Statistics.com DA-3 041219
may not be redistributed

In this chapter, we look at two useful approaches that combine methods for improving predictive power: *ensembles* and *uplift modeling*. An ensemble combines multiple supervised models into a "super-model." The previous chapters in this part of the book introduced different supervised methods for prediction and classification. Earlier, in Chapter 5, we learned about evaluating predictive performance, which can be used to compare several models and choose the best one. An ensemble is based on the powerful notion of *combining models*. Instead of choosing a single predictive model, we can combine several models to achieve improved predictive accuracy. In this chapter, we explain the underlying logic of why ensembles can improve predictive accuracy and introduce popular approaches for combining models, including simple averaging, bagging, and boosting.

In *uplift modeling*, we combine supervised modeling with A-B testing, which is a simple type of a randomized experiment. We describe the basics of A-B testing and how it is used along with predictive models in persuasion messaging not to predict outcomes but to predict who should receive which message or treatment.

Python

In this chapter, we will use `pandas` for data handling and `scikit-learn` for the models. We will also make use of the utility functions from the Python Utilities Functions Appendix. Use the following import statements for the Python code in this chapter.

Data Mining for Business Analytics: Concepts, Techniques, and Applications in Python, First Edition.
Galit Shmueli, Peter C. Bruce, Peter Gedeck, and Nitin R. Patel
© 2020 John Wiley & Sons, Inc. Published 2020 by John Wiley & Sons, Inc.

import required functionality for this chapter

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from dmba import classificationSummary
```

13.1 ENSEMBLES[1]

Ensembles played a major role in the million-dollar Netflix Prize contest that started in 2006. At the time, Netflix, the largest DVD rental service in the United States, wanted to improve their movie recommendation system (from www.netflixprize.com):

> Netflix is all about connecting people to the movies they love. To help customers find those movies, we've developed our world-class movie recommendation system: Cinematch[SM]…And while Cinematch is doing pretty well, it can always be made better.

In a bold move, the company decided to share a large amount of data on movie ratings by their users, and set up a contest, open to the public, aimed at improving their recommendation system:

> We provide you with a lot of anonymous rating data, and a prediction accuracy bar that is 10% better than what Cinematch can do on the same training data set.

During the contest, an active leader-board showed the results of the competing teams. An interesting behavior started appearing: Different teams joined forces to create combined, or *ensemble* predictions, which proved more accurate than the individual predictions. The winning team, called "BellKor's Pragmatic Chaos" combined results from the "BellKor" and "Big Chaos" teams alongside additional members. In a 2010 article in *Chance* magazine, the Netflix Prize winners described the power of their ensemble approach:

[1]This and subsequent sections in this chapter copyright © 2019 Datastats, LLC, and Galit Shmueli. Used by permission.

Statistics.com PA 3 041219 may not be redistributed

An early lesson of the competition was the value of combining sets of predictions from multiple models or algorithms. If two prediction sets achieved similar RMSEs, it was quicker and more effective to simply average the two sets than to try to develop a new model that incorporated the best of each method. Even if the RMSE for one set was much worse than the other, there was almost certainly a linear combination that improved on the better set.

Why Ensembles Can Improve Predictive Power

The principle of combining methods is popular for reducing risk. For example, in finance, portfolios are created for reducing investment risk. The return from a portfolio is typically less risky, because the variation is smaller than each of the individual components.

In predictive modeling, "risk" is equivalent to variation in prediction error. The more our prediction errors vary, the more volatile our predictive model. Consider predictions from two different models for a set of n records. $e_{1,i}$ is the prediction error for the ith record by method 1 and $e_{2,i}$ is the prediction error for the same record by method 2.

Suppose that each model produces prediction errors that are, on average, zero (for some records the model over-predicts and for some it under-predicts, but on average the error is zero):

$$E(e_{1,i}) = E(e_{2,i}) = 0.$$

If, for each record, we take an average of the two predictions: $\overline{y}_i = \frac{\hat{y}_{1,i}+\hat{y}_{2,i}}{2}$, then the expected mean error will also be zero:

$$\begin{aligned} E\left(y_i - \overline{y_i}\right) &= E\left(y_i - \frac{\hat{y}_{1,i} + \hat{y}_{2,i}}{2}\right) \\ &= E\left(\frac{y_i - \hat{y}_{1,i}}{2} + \frac{y_i - \hat{y}_{2,i}}{2}\right) = E\left(\frac{e_{1,i} + e_{2,i}}{2}\right) = 0. \end{aligned} \tag{13.1}$$

This means that the ensemble has the same mean error as the individual models. Now let us examine the variance of the ensemble's prediction errors:

$$\mathrm{V}ar\left(\frac{e_{1,i} + e_{2,i}}{2}\right) = \frac{1}{4}\left(\mathrm{V}ar(e_{1,i}) + \mathrm{V}ar(e_{2,i})\right) + \frac{1}{4} \times 2\mathrm{C}ov\left(e_{1,i}, e_{2,i}\right). \tag{13.2}$$

This variance can be lower than each of the individual variances $\mathrm{V}ar(e_{1,i})$ and $\mathrm{V}ar(e_{2,i})$ under some circumstances. A key component is the covariance (or equivalently, correlation) between the two prediction errors. The case of no correlation leaves us with a quantity that can be smaller than each of the individual variances. The variance of the average prediction error will be even smaller when the two prediction errors are negatively correlated.

Statistics.com PA 3 041219 may not be redistributed

In summary, using an average of two predictions can potentially lead to smaller error variance, and therefore better predictive power. These results generalize to more than two methods; you can combine results from multiple prediction methods or classifiers.

THE WISDOM OF CROWDS

In his book *The Wisdom of Crowds*, James Surowiecki recounts how Francis Galton, a prominent statistician from the 19th century, watched a contest at a county fair in England. The contest's objective was to guess the weight of an ox. Individual contest entries were highly variable, but the mean of all the estimates was surprisingly accurate—within 1% of the true weight of the ox. On balance, the errors from multiple guesses tended to cancel one another out. You can think of the output of a predictive model as a more informed version of these guesses. Averaging together multiple guesses will yield a more precise answer than the vast majority of the individual guesses. Note that in Galton's story, there were a few (lucky) individuals who scored better than the average. An ensemble estimate will not always be more accurate than all the individual estimates in all cases, but it will be more accurate most of the time.

Simple Averaging

The simplest approach for creating an ensemble is to combine the predictions, classifications, or propensities from multiple models. For example, we might have a linear regression model, a regression tree, and a k-NN algorithm. We use each of the three methods to score, say, a test set. We then combine the three sets of results.

The three models can also be variations that use the same algorithm. For example, we might have three linear regression models, each using a different set of predictors.

Combining Predictions In prediction tasks, where the outcome variable is numerical, we can combine the predictions from the different methods simply by taking an average. In the above example, for each record in the test set, we have three predictions (one from each model). The ensemble prediction is then the average of the three values.

One alternative to a simple average is taking the median prediction, which would be less affected by extreme predictions. Another possibility is computing a weighted average, where weights are proportional to a quantity of interest. For instance, weights can be proportional to the accuracy of the model, or if different data sources are used, the weights can be proportional to the quality of the data.

Statistics.com LLC 2011219 may not be redistributed

Ensembles for prediction are useful not only in cross-sectional prediction, but also in time series forecasting (see Chapters 16–18). In forecasting, the same approach of combining future forecasts from multiple methods can lead to more precise predictions. One example is the weather forecasting application Forecast.io (www.forecast.io), which describes their algorithm as follows:

> Forecast.io is backed by a wide range of data sources, which are aggregated together statistically to provide the most accurate forecast possible for a given location.

Combining Classifications In the case of classification, combining the results from multiple classifiers can be done using "voting": For each record, we have multiple classifications. A simple rule would be to choose the most popular class among these classifications. For example, we might use a classification tree, a naive Bayes classifier, and discriminant analysis for classifying a binary outcome. For each record we then generate three predicted classes. Simple voting would choose the most common class among the three.

As in prediction, we can assign heavier weights to scores from some models, based on considerations such as model accuracy or data quality. This would be done by setting a "majority rule" that is different from 50%.

Combining Propensities Similar to predictions, propensities can be combined by taking a simple (or weighted) average. Recall that some algorithms, such as naive Bayes (see Chapter 8), produce biased propensities and should therefore not be simply averaged with propensities from other methods.

Bagging

Another form of ensembles is based on averaging across multiple random data samples. *Bagging*, short for "bootstrap aggregating," comprises two steps:

1. Generate multiple random samples (by sampling with replacement from the original data)—this method is called "bootstrap sampling."
2. Running an algorithm on each sample and producing scores.

Bagging improves the performance stability of a model and helps avoid overfitting by separately modeling different data samples and then combining the results. It is therefore especially useful for algorithms such as trees and neural networks.

Boosting

Boosting is a slightly different approach to creating ensembles. Here the goal is to directly improve areas in the data where our model makes errors, by forcing the model to pay more attention to those records. The steps in boosting are:

Statistics.com PASS 04 12 19
may not be redistributed

1. Fit a model to the data.
2. Draw a sample from the data so that misclassified records (or records with large prediction errors) have higher probabilities of selection.
3. Fit the model to the new sample.
4. Repeat Steps 2–3 multiple times.

Bagging and Boosting in Python

Although bagging and boosting can be applied to any data mining method, most applications are for trees, where they have proved extremely effective. In Chapter 9, we described random forests, an ensemble based on bagged trees. We illustrated a random forest implementation for the personal loan example. `scikit-learn` has a variety of methods to combine classifiers (models for predicting categorical outcomes) and regressors (models for predicting numerical outcomes) using bagging or boosting. Here, we will demonstrate this for decision tree classifiers. Table 13.1 shows the Python code and output producing a bagged tree and a boosted tree for the personal loan data, and how they are used to generate classifications for the validation set. In this example we see that bagging and boosting both produce better validation accuracy than the single tree.

Advantages and Weaknesses of Ensembles

Combining scores from multiple models is aimed at generating more precise predictions (lowering the prediction error variance). The ensemble approach is most useful when the combined models generate prediction errors that are negatively associated, but it can also be useful when the correlation is low. Ensembles can use simple averaging, weighted averaging, voting, medians, etc. Models can be based on the same algorithm or on different algorithms, using the same sample or different samples. Ensembles have become a major strategy for participants in data mining contests, where the goal is to optimize some predictive measure. In that sense, ensembles also provide an operational way to obtain solutions with high predictive power in a fast way, by engaging multiple teams of "data crunchers" working in parallel and combining their results.

Ensembles that are based on different data samples help avoid overfitting. However, remember that you can also overfit the data with an ensemble if you tweak it (e.g., choosing the "best" weights when using a weighted average).

The major disadvantage of an ensemble is the resources that it requires: computationally, as well as in terms of software availability and the analyst's skill and time investment. Ensembles that combine results from different algorithms require developing each of the models and evaluating them. Boosting-type ensembles and bagging-type ensembles do not require such effort, but they do

Statistics.com PA C 041219
may not be redistributed

TABLE 13.1 EXAMPLE OF BAGGING AND BOOSTING CLASSIFICATION TREES (PERSONAL LOAN DATA)

code for bagging and boosing trees

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)

# split into training and validation
X = bank_df.drop(columns=['Personal Loan'])
y = bank_df['Personal Loan']
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.40, random_state=3)

# single tree
defaultTree = DecisionTreeClassifier(random_state=1)
defaultTree.fit(X_train, y_train)

classes = defaultTree.classes_
classificationSummary(y_valid, defaultTree.predict(X_valid), class_names=classes)

# bagging
bagging = BaggingClassifier(DecisionTreeClassifier(random_state=1),
                            n_estimators=100, random_state=1)
bagging.fit(X_train, y_train)

classificationSummary(y_valid, bagging.predict(X_valid), class_names=classes)

# boosting
boost = AdaBoostClassifier(DecisionTreeClassifier(random_state=1),
                           n_estimators=100, random_state=1)
boost.fit(X_train, y_train)

classificationSummary(y_valid, boost.predict(X_valid), class_names=classes)
```

Output

```
> # single tree
Confusion Matrix (Accuracy 0.9825)

       Prediction
Actual    0    1
     0 1778   15
     1   20  187

> # bagging
Confusion Matrix (Accuracy 0.9855)

       Prediction
Actual    0    1
     0 1781   12
     1   17  190

> # boosting
Confusion Matrix (Accuracy 0.9840)

       Prediction
Actual    0    1
     0 1779   14
     1   18  189
```

Statistics.com PA 3 041219 may not be redistributed

have a computational cost (although boosting can be parallelized easily). Ensembles that rely on multiple data sources require collecting and maintaining multiple data sources. And finally, ensembles are "blackbox" methods, in that the relationship between the predictors and the outcome variable usually becomes nontransparent.

13.2 Uplift (Persuasion) Modeling

Long before the advent of the Internet, sending messages directly to individuals (i.e., direct mail) held a big share of the advertising market. Direct marketing affords the marketer the ability to invite and monitor direct responses from consumers. This, in turn, allows the marketer to learn whether the messaging is paying off. A message can be tested with a small section of a large list and, if it pays off, the message can be rolled out to the entire list. With predictive modeling, we have seen that the rollout can be targeted to that portion of the list that is most likely to respond or behave in a certain way. None of this was possible with traditional media advertising (television, radio, newspaper, magazine).

Direct response also made it possible to test one message against another and find out which does better.

A-B Testing

A-B testing is the marketing industry's term for a standard scientific experiment in which results can be tracked for each individual. The idea is to test one treatment against another, or a treatment against a control. "Treatment" is simply the term for the intervention you are testing: In a medical trial it is typically a drug, device, or other therapy; in marketing it is typically an offering to a consumer—for example, an e-mail, or a web page shown to a consumer. A general display ad in a magazine would not generally qualify, unless it had a specific call to action that allowed the marketer to trace the action (e.g., purchase) to a given ad, plus the ability to split the magazine distribution randomly and provide a different offer to each segment.

An important element of A-B testing is random allocation—the treatments are assigned or delivered to individuals randomly. That way, any difference between treatment A and treatment B can be attributed to the treatment (unless it is due to chance).

Uplift

An A-B test tells you which treatment does better on average, but says nothing about which treatment does better for which individual. A classic example is in political campaigns. Consider the following scenario: The campaign director

Statistics.com PA 5 041219 may not be redistributed

for Smith, a Democratic Congressional candidate, would like to know which voters should be called to encourage to support Smith. Voters that tend to vote Democratic but are not activists might be more inclined to vote for Smith if they got a call. Active Democrats are probably already supportive of him, and therefore a call to them would be wasted. Calls to Republicans are not only wasteful, but they could be harmful.

Campaigns now maintain extensive data on voters to help guide decisions about outreach to individual voters. Prior to the 2008 Obama campaign, the practice was to make rule-based decisions based on expert political judgment. Since 2008, it has increasingly been recognized that, rather than relying on judgment or supposition to determine whether an individual should be called, it is best to use the data to develop a model that can predict whether a voter will respond positively to outreach.

Gathering the Data

US states maintain publicly available files of voters, as part of the transparent oversight process for elections. The voter file contains data such as name, address, and date of birth. Political parties have "poll-watchers" at elections to record who votes, so they have additional data on which elections voters voted in. Census data for neighborhoods can be appended, based on voter address. Finally, commercial demographic data can be purchased and matched to the voter data. Table 13.2 shows a small extract of data derived from the voter file for the US state of Delaware.[2] The actual data used in this problem are in the file *VoterPersuasion.csv* and contain 10,000 records and many additional variables beyond those shown in Table 13.2.

First, the campaign director conducts a survey of 10,000 voters to determine their inclination to vote Democratic. Then she conducts an experiment, randomly splitting the sample of 10,000 voters in half and mailing a message promoting Smith to half the list (treatment A), and nothing to the other half (treatment B). The control group that gets no message is essential, since other campaigns or news events might cause a shift in opinion. The goal is to measure the change in opinion after the message is sent out, relative to the no-message control group.

The next step is conducting a post-message survey of the same sample of 10,000 voters, to measure whether each voter's opinion of Smith has shifted in a positive direction. A binary variable, Moved_AD, will be added to the above data, indicating whether opinion has moved in a Democratic direction (1) or not (0).

[2]Thanks to Ken Strasma, founder of the microtargeting firm HaystaqDNA and director of targeting for the 2004 Kerry campaign and the 2008 Obama campaign, for these data.

Statistics.com PA 3 04/12/19 may not be redistributed

TABLE 13.2 DATA ON VOTERS (SMALL SUBSET OF VARIABLES AND RECORDS) AND DATA DICTIONARY

Voter	Age	NH_White	Comm_PT	H_F1	Reg_Days	PR_Pelig	E_Elig	Political_C
1	28	70	0	0	3997	0	20	1
2	23	67	3	0	300	0	0	1
3	57	64	4	0	2967	0	0	0
4	70	53	2	1	16620	100	90	1
5	37	76	2	0	3786	0	20	0

Data Dictionary

Age	Voter age in years
NH_White	Neighborhood average of % non-Hispanic white in household
Comm_PT	Neighborhood % of workers who take public transit
H_F1	Single female household (1 = yes)
Reg_Days	Days since voter registered at current address
PR_Pelig	Voted in what % of non-presidential primaries
E_Pelig	Voted in what % of any primaries
Political_C	Is there a political contributor in the home? (1 = yes)

Table 13.3 summarizes the results of the survey, by comparing the movement in a Democratic direction for each of the treatments. Overall, the message (Message = 1) is modestly effective.

Movement in a Democratic direction among those who got no message is 34.4%. This probably reflects the approach of the election, the heightening campaign activity, and the reduction in the "no opinion" category. It also illustrates the need for a control group. Among those who did get the message, the movement in a Democratic direction is 40.2%. So, overall, the lift from the message is 5.8%.

TABLE 13.3 RESULTS OF SENDING A PRO-DEMOCRATIC MESSAGE TO VOTERS

	#Voters	# Moved Dem.	% Moved Dem.
Message = 1 (message sent)	5000	2012	40.2%
Message = 0 (no message sent)	5000	1722	34.4%

We can now append two variables to the voter data shown earlier in Table 13.2: *message* [whether they received the message (1) or not (0)] and *Moved_AD* [whether they moved in a Democratic direction (1) or not (0)]. The augmented data are shown in Table 13.4.

A Simple Model

We can develop a predictive model with Moved_AD as the outcome variable, and various predictor variables, *including the treatment Message*. Any classification

Statistics.com PASS 04 12 19 may not be redistributed

TABLE 13.4 OUTCOME VARIABLE (MOVED_AD) AND TREATMENT VARIABLE (MESSAGE) ADDED TO VOTER DATA

Voter	Age	NH_White	Comm_PT	H_F1	Reg_Days	PR_Pelig	E_Elig	Political_C	Message	Moved_AD
1	28	70	0	0	3997	0	20	1	0	1
2	23	67	3	0	300	0	0	1	1	1
3	57	64	4	0	2967	0	0	0	0	0
4	70	53	2	1	16620	100	90	1	0	0
5	37	76	2	0	3786	0	20	0	1	0

TABLE 13.5 CLASSIFICATIONS AND PROPENSITIES FROM PREDICTIVE MODEL (SMALL EXTRACT)

Voter	Message	Actual Moved_AD	Predicted Moved_AD	Predicted Prob.
1	0	1	1	0.5975
2	1	1	1	0.5005
3	0	0	0	0.2235
4	0	0	0	0.3052
5	1	0	0	0.4140

method can be used; Table 13.5 shows the first few lines from the output of some predictive model used to predict Moved_AD.

However, our interest is not just how the message did overall, nor is it whether we can predict the probability that a voter's opinion will move in a favorable direction. Rather our goal is to predict how much (positive) impact the message will have on a specific voter. That way the campaign can direct its limited resources toward the voters who are the most persuadable—those for whom sending the message will have the greatest positive effect.

Modeling Individual Uplift

To answer the question about the message's impact on each voter, we need to model the effect of the message at the individual voter level. For each voter, uplift is defined as follows:

Uplift = increase in propensity of favorable opinion after receiving message

To build an uplift model, we follow the following steps to estimate the change in probability of "success" (propensity) that comes from receiving the treatment (the message):

1. Randomly split a data sample into treatment and control groups, conduct an A-B test, and record the outcome (in our example: Moved_AD)
2. Recombining the data sample, partition it into training and validation sets; build a predictive model with this outcome variable and include a predictor variable that denotes treatment status (in our example: Message). If logistic regression is used, additional interaction terms between treatment status and other predictors can be added as predictors to allow

Statistics.com PA 3 041219 may not be redistributed

TABLE 13.6 CLASSIFICATIONS AND PROPENSITIES FROM PREDICTIVE MODEL (SMALL EXTRACT) WITH MESSAGE VALUES REVERSED

Voter	Message	Actual Moved_AD	Predicted Moved_AD	Predicted Prob.
1	1	1	1	0.6908
2	0	1	1	0.3996
3	1	0	0	0.3022
4	1	0	0	0.3980
5	0	0	0	0.3194

the treatment effect to vary across records (in data-driven methods such as trees and KNN this happens automatically).

3. Score this predictive model to a partition of the data; you can use the validation partition. This will yield, for each validation record, its propensity of success given its treatment.
4. Reverse the value of the treatment variable and re-score the same model to that partition. This will yield for each validation record its propensity of success had it received the other treatment.
5. Uplift is estimated for each individual by P(Success | Treatment = 1) − P(Success | Treatment = 0)
6. For new data where no experiment has been performed, simply add a synthetic predictor variable for treatment and assign first a '1,' score the model, then a '0,' and score the model again. Estimate uplift for the new record(s) as above.

Continuing with the small voter example, the results from Step 3 were shown in Table 13.5—the right column shows the propensities from the model. Next, we re-train the predictive model, but with the values of the treatment variable Message reversed for each row. Table 13.6 shows the propensities with variable Message reversed (you can see the reversed values in column Message). Finally, in Step 5, we calculate the uplift for each voter.

Table 13.7 shows the uplift for each voter—the success (Moved_AD = 1) propensity given Message = 1 minus the success propensity given Message = 0.

TABLE 13.7 UPLIFT: CHANGE IN PROPENSITIES FROM SENDING MESSAGE VS. NOT SENDING MESSAGE

Voter	Prob. if Message = 1	Prob. if Message = 0	Uplift
1	0.6908	0.5975	0.0933
2	0.5005	0.3996	0.1009
3	0.3022	0.2235	0.0787
4	0.3980	0.3052	0.0928
5	0.4140	0.3194	0.0946

Statistics.com PA 3041219 may not be redistributed

Computing Uplift with Python

This entire process that we showed manually can be done using `scikit-learn`. One difference to our manual computation is that we used a logistic regression whereas here we will use a random forest classifier. Table 13.8 shows the result of implementing the uplift analysis in `scikit-learn` using a random forest. The output shows the two conditional probabilities, P(Success | Treatment = 1) and P(Success | Treatment = 0), estimated for each record. The difference between the values in Tables 13.7 and 13.8 are due to the use of two different predictive algorithms (logistic regression vs. random forests).

Using the Results of an Uplift Model

Once we have estimated the uplift for each individual, the results can be ordered by uplift. The message could then be sent to all those voters with a positive uplift, or, if resources are limited, only to a subset—those with the greatest uplift.

Uplift modeling is used mainly in marketing and, more recently, in political campaigns. It has two main purposes:

- To determine whether to send someone a persuasion message, or just leave them alone.
- When a message is definitely going to be sent, to determine which message, among several possibilities, to send.

Technically this amounts to the same thing—"send no message" is simply another category of treatment, and an experiment can be constructed with multiple treatments, for example, no message, message A, and message B. However, practitioners tend to think of the two purposes as distinct, and tend to focus on the first. Marketers want to avoid sending discount offers to customers who would make a purchase anyway, or renew a subscription anyway. Political campaigns, likewise, want to avoid calling voters who would vote for their candidate in any case. And both parties especially want to avoid sending messages or offers where the effect might be antagonistic—where the uplift is negative.

13.3 SUMMARY

In practice, the methods discussed in this book are often used not in isolation, but as building blocks in an analytic process whose goal is always to inform and provide insight.

In this chapter, we looked at two ways that multiple models are deployed. In ensembles, multiple models are weighted and combined to produce improved predictions. In uplift modeling, the results of A-B testing are folded into the

Statistics.com PA 3 04/12/19 may not be redistributed

TABLE 13.8 UPLIFT IN PYTHON APPLIED TO THE VOTERS DATA

code for uplift

```
voter_df = pd.read_csv('Voter-Persuasion.csv')

# Preprocess data frame
predictors = ['AGE', 'NH_WHITE', 'COMM_PT', 'H_F1', 'REG_DAYS',
              'PR_PELIG', 'E_PELIG', 'POLITICALC', 'MESSAGE_A']
outcome = 'MOVED_AD'

classes = list(voter_df.MOVED_AD.unique())

# Partition the data
X = voter_df[predictors]
y = voter_df[outcome]
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.40, random_state=1)

# Train a random forest classifier using the training set
rfModel = RandomForestClassifier(n_estimators=100, random_state=1)
rfModel.fit(X_train, y_train)

# Calculating the uplift
uplift_df = X_valid.copy()  # Need to create a copy to allow modifying data

uplift_df.MESSAGE_A = 1
predTreatment = rfModel.predict_proba(uplift_df)
uplift_df.MESSAGE_A = 0
predControl = rfModel.predict_proba(uplift_df)

upliftResult_df = pd.DataFrame({
    'probMessage': predTreatment[:,1],
    'probNoMessage': predControl[:,1],
    'uplift': predTreatment[:,1] - predControl[:,1],
    }, index=uplift_df.index)
upliftResult_df.head()
```

Output

```
      probMessage  probNoMessage  uplift
9953         0.77           0.62    0.15
3850         0.39           0.39    0.00
4962         0.20           0.14    0.06
3886         0.86           0.62    0.24
5437         0.10           0.28   -0.18
```

predictive modeling process as a predictor variable to guide choices not just about whether to send an offer or persuasion message, but also as to who to send it to.

Statistics.com PA 3 041219 may not be redistributed

PROBLEMS

13.1 Acceptance of Consumer Loan Universal Bank has begun a program to encourage its existing customers to borrow via a consumer loan program. The bank has promoted the loan to 5000 customers, of whom 480 accepted the offer. The data are available in file *UniversalBank.csv*. The bank now wants to develop a model to predict which customers have the greatest probability of accepting the loan, to reduce promotion costs and send the offer only to a subset of its customers.

We will develop several models, then combine them in an ensemble. The models we will use are (1) logistic regression, (2) k-nearest neighbors with $k = 3$, and (3) classification trees. Preprocess the data as follows:

- Bin the following variables so they can be used in Naive Bayes: Age (5 bins), Experience (10 bins), Income (5 bins), CC Average (6 bins), and Mortgage (10 bins).
- Education and Family can be used as is, without binning.
- Zip code can be ignored.
- Use one-hot-encoding to convert the categorical data into indicator variables
- Partition the data: 60% training, 40% validation.

a. Fit models to the data for (1) logistic regression, (2) k-nearest neighbors with $k = 3$, (3) classification trees, and (4) Naive Bayes. Use Personal Loan as the outcome variable. Report the validation confusion matrix for each of the models.

b. Create a data frame with the actual outcome, predicted outcome, and each of the models. Report the first 10 rows of this data frame.

c. Add two columns to this data frame for (1) a majority vote of predicted outcomes, and (2) the average of the predicted probabilities. Using the classifications generated by these two methods derive a confusion matrix for each method and report the overall accuracy.

d. Compare the error rates for the four individual methods and the two ensemble methods.

13.2 eBay Auctions—Boosting and Bagging Using the eBay auction data (file *eBayAuctions.csv*) with variable Competitive as the outcome variable, partition the data into training (60%) and validation (40%).

a. Run a classification tree, using the default settings of *DecisionTreeClassifier*. Looking at the validation set, what is the overall accuracy? What is the lift on the first decile?

b. Run a boosted tree with the same predictors (use *AdaBoostClassifier* with *DecisionTreeClassifier* as the base estimator). For the validation set, what is the overall accuracy? What is the lift on the first decile?

c. Run a bagged tree with the same predictors (use *BaggingClassifier*). For the validation set, what is the overall accuracy? What is the lift on the first decile?

d. Run a random forest (use *RandomForestClassifier*). Compare the bagged tree to the random forest in terms of validation accuracy and lift on first decile. How are the two methods conceptually different?

13.3 Predicting Delayed Flights (Boosting). The file *FlightDelays.csv* contains information on all commercial flights departing the Washington, DC area and arriving at New York during January 2004. For each flight there is information on the departure and

Statistics.com PA - 3041 2019 may not be redistributed

arrival airports, the distance of the route, the scheduled time and date of the flight, and so on. The variable that we are trying to predict is whether or not a flight is delayed. A delay is defined as an arrival that is at least 15 minutes later than scheduled.

Data Preprocessing. Transform variable day of week info a categorical variable. Bin the scheduled departure time into eight bins (in Python use function *pd.cut()* from the `pandas` package). Partition the data into training (60%) and validation (40%).

Run a boosted classification tree for delay. With the exception of setting `n_estimators=500` and `random_state=1`, use default setting for the DecisionTreeClassifier and the AdaBoostClassifier.

a. Compared with the single tree, how does the boosted tree behave in terms of overall accuracy?

b. Compared with the single tree, how does the boosted tree behave in terms of accuracy in identifying delayed flights?

c. Explain why this model might have the best performance over the other models you fit.

13.4 Hair Care Product—Uplift Modeling This problem uses the data set in *Hair-Care-Product.csv*, courtesy of SAS. In this hypothetical case, a promotion for a hair care product was sent to some members of a buyers club. Purchases were then recorded for both the members who got the promotion and those who did not.

a. What is the purchase propensity

i. among those who received the promotion?

ii. among those who did not receive the promotion?

b. Partition the data into training (60%) and validation (40%) and fit:

i. Uplift using a Random Forest.

ii. Uplift using k-NN.

c. Report the two models' recommendations for the first three members.

Statistics.com DA 3 04 12 19
may not be redistributed