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.

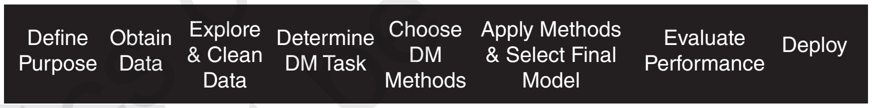

FIGURE 2.1 SCHEMATIC OF THE DATA MODELING PROCESS

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.

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.

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."

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.

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.

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.)

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 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.

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

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
```

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.

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 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.

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 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 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.

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

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.

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 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

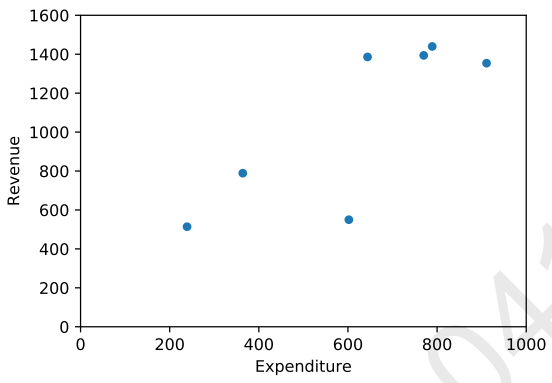

FIGURE 2.2 SCATTER PLOT FOR ADVERTISING AND SALES DATA

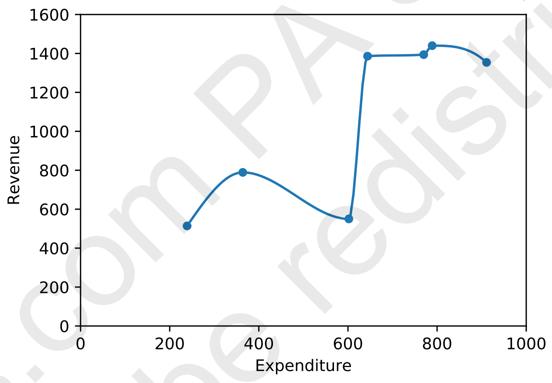

FIGURE 2.3 OVERFITTING: THIS FUNCTION FITS THE DATA WITH NO ERROR

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

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 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.

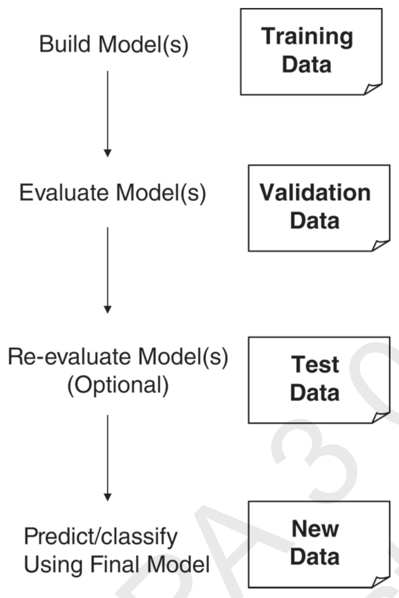

FIGURE 2.4 THREE DATA PARTITIONS AND THEIR ROLE IN THE DATA MINING PROCESS

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

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 "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

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.

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
```

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.

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
```

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.

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 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 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

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/

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

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

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

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 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.

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.

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

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

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.