Machine Learning in Python®: Essential Techniques for Predictive Analysis
By Michael Bowles

CHAPTER

2

Understand the Problem by Understanding the Data

A new data set (problem) is a wrapped gift. It's full of promise and anticipation at the miracles you can wreak once you've solved it. But it remains a mystery until you've opened it. This chapter is about opening up your new data set so you can see what's inside, get an appreciation for what you'll be able to do with the data, and start thinking about how you'll approach model building with it.

This chapter has two purposes. One is to familiarize you with data sets that will be used later as examples of different types of problems to be solved using the algorithms you'll learn in Chapter 4, "Penalized Linear Regression," and Chapter 6, "Ensemble Methods." The other purpose is to demonstrate some of the tools available in Python for data exploration.

The chapter uses a simple example to review some basic problem structure, nomenclature, and characteristics of a machine learning data set. The language introduced in this section will be used throughout the rest of the book. After establishing some common language, the chapter goes one by one through several different types of function approximation problems. These problems illustrate common variations of machine learning problems so that you'll know how to recognize the variants when you see them and will know how to handle them (and will have code examples for them).

The Anatomy of a New Problem

The algorithms covered in this book start with a matrix (or table) full of numbers and perhaps some character variables. The example in Table 2-1 establishes some nomenclature and represents a small machine learning data set in a two-dimensional table. The table will give you a mental image of a data set so that references to "columns corresponding to attributes" or rows corresponding to individual examples will be familiar. In this example, the predictive analytics problem is to predict how much money individuals will spend buying books online over the next year.

Table 2-1: Data for a Machine Learning Problem

USERID	ATTRIBUTE 1	ATTRIBUTE 2	ATTRIBUTE 3	LABELS
001	6.5	Male	12	$120
004	4.2	Female	17	$270
007	5.7	Male	3	$75
008	5.8	Female	8	$600

The data are arranged into rows and columns. Each row represents an individual case (also called an *instance*, *example*, or *observation*). The columns in Table 2-1 are given designations that indicate the roles they will play in the machine learning problem. The columns designated as attributes will be used to make predictions of the dollars spent on books. In the column designated as labels, you'll see how much each customer spent last year on books.

NOTE **Machine learning data sets are most commonly arranged with columns corresponding to a single attribute and rows corresponding to a single observation, but not always. For example, some text mining literature arranges the matrix the other way around—with columns corresponding to an observation and rows corresponding to an attribute.**

In Table 2-1, a row represents an individual customer, and the data in the row all pertain to that individual. The first column is called UserID and contains an identifier that is unique for each row (case). A unique identifier may or may not be present in your problem. For instance, websites typically tag site visitors with a user ID that is associated with them for the duration of their visit. If a user does not register with the site, the same user gets a different ID with each visit. The ID is usually assigned to each observation, which will be the subject of the prediction you're going to build. Columns 2, 3, and 4 are called Attributes instead of being given more specific names like Height or Gender. The point is

to highlight their role in the prediction process. Attributes are data available about the case that will be used to make predictions.

Labels are the things you want to predict. In this example, UserID is a simple number, Attribute 1 is height, Attribute 2 is gender, and Attribute 3 is how many books the person read last year. The column under Labels contains how much money the individual spent on books online last year. What are the roles that these different categories of data will play? What use does a machine learning algorithm make of user ID, attributes, and labels? The short answer is this: You ignore the user ID. You use the attributes to predict the labels.

The unique ID is for bookkeeping purposes and allows you to refer back to the other data available for the specific case. Generally, the unique ID does not get used directly in a machine learning algorithm. Attributes are the things that you've chosen to use for making predictions. Labels are observed outcomes that the machine learning algorithm will use to build a predictive model.

User ID doesn't usually get used for making predictions because it is too specific. It pertains to only a single example. The trick with machine learning is to build a model that generalizes to new cases (not merely memorizing past cases). To achieve that, the algorithm must be derived so that it is forced to pay attention to more than one row of data. One possible exception to excluding user ID is when the user ID is numeric and assigned in the order that users are signed up. Basically, it's indicating signup date in that case and can be useful because users with close IDs signed up at similar times and can be considered as a group on that basis.

The process of building a predictive model is called *training*. The way the process proceeds depends on the algorithm, and later chapters cover the details, but it is often iterative. The algorithm postulates a predictive relationship between the attributes and the labels, observes the mistakes that it makes, and makes some correction, and then iterates on that process until a sound model is achieved. A number of technicalities are addressed later, but that's the basic idea.

WHAT'S IN A NAME?

Attributes and labels go by a variety of names, and new machine learners can get tripped up by the name switching from one author to another or even one paragraph to another from a single author.

Attributes (the variables being used to make predictions) are also known as the following:

- **Predictors**
- **Features**

- Independent variables
- Inputs

Labels are also known as the following:

- Outcomes
- Targets
- Dependent variables
- Responses

Different Types of Attributes and Labels Drive Modeling Choices

The attributes shown in Table 2-1 come in two different types: numeric variables and categorical (or factor) variables. Attribute 1 (height) is a numeric variable and is the most usual type of attribute. Attribute 2 is gender and is indicated by the entry Male or Female. This type of attribute is called a *categorical* or *factor* variable. Categorical variables have the property that there's no order relation between the various values. There's no sense to Male < Female (despite centuries of squabbling). Categorical variables can be two-valued, like Male Female, or multivalued, like states (AL, AK, AR . . . WY). Other distinctions can be drawn regarding attributes (integer versus float, for example), but they do not have the same impact on machine learning algorithms. The reason for this is that many machine learning algorithms take numeric attributes only; they cannot handle categorical or factor variables. Penalized regression algorithms deal only with numeric attributes. The same is true for support vector machines, kernel methods, and K-nearest neighbors. Chapter 4 will cover methods for converting categorical variables to numeric variables. The nature of the variables will shape your algorithm choices and the direction you take in developing a predictive model, so it's one of the things you need to pay attention to when you face a new problem.

A similar dichotomy arises for the labels. The labels shown in Table 2-1 are numeric: the amount of money that the individual spent on books online last year. In other problems, though, the labels may also be categorical. For example, if the job with Table 2-1 were to predict which individuals would spend more than $200 next year the problem would change, and the problem approach would change. The new problem of predicting which customers would spend more than $200 would have new labels. The new labels would take one of two values. Table 2-2 shows the relationship between the labels given in Table 2-1 and new labels based on the logical proposition Spending > $200. The new labels shown in Table 2-2 take one of two values—True or False.

Table 2-2: Numeric Targets versus Categorical Targets

TABLE 1 LABELS	>$200 ?
$120	False
$270	True
$75	False
$600	True

When the labels are numeric, the problem is called a *regression problem*. When the labels are categorical, the problem is called a *classification problem*. If the categorical target takes only two values, the problem is called a *binary classification problem*. If it takes more than two values, the problem is called a *multiclass classification problem*.

In many cases, the choice of problem type is up to the designer. You've just seen that this example problem can be converted from a regression problem to a binary classification problem by the simple transformation of the labels. These are tradeoffs that you may might to make as part of your attack on a problem. For example, classification targets might better support a decision between two courses of action.

The classification problem might also be simpler than the regression problem. Consider, for instance, the difference in complexity between a topographic map with a single contour line (say the 100-foot contour line) and a topographic map with contour lines every 10 feet. The single contour divides the map into the areas that are higher than 100 feet and those that are lower and contains considerably less information than the more detailed contour map. A classifier is trying to compute a single dividing contour without regard for behavior distant from the decision boundary, whereas regression is trying to draw the whole map.

Things to Notice about Your New Data Set

You'll want to ascertain a number of other features of the data set as part of your initial inspection of the data. The following is a checklist and a sequence of things to learn about your data set to familiarize yourself with the data and to formulate the predictive model development steps that you want to follow. These are simple things to check and directly impact your next steps. In addition, the process gets you moving around the data and learning its properties.

Items to Check

Number of rows and columns

Number of categorical variables and number of unique values for each

Missing values

Summary statistics for attributes and labels

One of the first things to check is the size and shape of the data. Read the data into a list of lists; then the dimension of the outer list is the number of rows, and the dimension of one of the inner lists is the number of columns. The next section shows the concrete application of this to one of the data sets that you'll see used later to illustrate the properties of an algorithm that will be developed.

The next step in the process is to determine how many missing values there are in each row. The reason for doing it on a row-by-row basis is that the simplest way to deal with missing values is to throw away instances that aren't complete (examples with at least one missing value). In many situations, this can bias the results, but just a few incomplete examples will not make a material difference. By counting the rows with missing data (in addition to the total number of missing entries), you'll know how much of the data set you have to discard if you use the easy method.

If you have a large number of rows, as you might if you're collecting web data, the number you'll lose may be small compared to the number of rows of data you have available. If you're working on biological problems where the data are expensive and you have many attributes, you might not be able to afford to throw data out. In that case, you'll have to figure out some ways to fill in the missing values or use an algorithm that can deal with them. Filling them in is called *imputation*. The easiest way to impute the missing data is to fill in the missing entries using average values of the entries in each row. A more sophisticated method is to use one of the predictive methods covered in Chapters 4 and 6. To use a predictive method, you treat a column of attributes with missing values as though it were labels. Be sure to remove the original problem labels before undertaking this process.

The next several sections are going to go through the process outlined here and will introduce some methods for characterizing your data set to help you decide how to attack the modeling process.

Classification Problems: Detecting Unexploded Mines Using Sonar

This section steps through several checks that you might make on a classification problem as you begin digging into it. It starts with simple measurements of size and shape, reporting data types, counting missing values, and so forth. Then it moves on to statistical properties of the data and interrelationships between attributes and between attributes and the labels. The data set comes from the UC Irvine Data Repository [Ref 1.]. The data result from some experiments to determine if sonar can be used to detect unexploded mines left in harbors subsequent to military actions. The sonar signal is what's called a *chirped signal*. That means that the signal rises (or falls) in frequency over the duration of the

sound pulse. The measurements in the data set represent the power measurements collected in the sonar receiver at different points in the returned signal. For roughly half of the examples, the sonar is illuminating a rock, and for the other half a metal cylinder having the shape of a mine. The data set goes by the name of "Rocks versus Mines."

Physical Characteristics of the Rocks Versus Mines Data Set

The first thing to do with a new data set is to determine its size and shape. Listing 2-1 shows code for determining the size and shape of the "Rocks versus Mines" data set from the UC Irvine Data Repository: the rocks versus mines data. Later in this chapter, you'll learn more about this data set, and the book will use it for example purposes as the algorithms are introduced. The process for determining the number of rows and columns is pretty simple in this case. The file is comma delimited, with the data for one experiment occupying one line of text. This makes it a simple matter to read a line, split it on the comma delimiters, and stack the resulting lists into an outer list containing the whole data set.

Listing 2-1: Sizing Up a New Data Set—rockVmineSummaries.py (Output: outputRocksVMinesSummaries.txt)

```
__author__ = 'mike_bowles'
import urllib2
import sys

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

sys.stdout.write("Number of Rows of Data = " + str(len(xList)) + '\n')
sys.stdout.write("Number of Columns of Data = " + str(len(xList[1])))


Output:
Number of Rows of Data = 208
Number of Columns of Data = 61
```

As you can see in the sample output, this data set has 208 rows (lines) and 61 columns (fields per line). What difference does this make? The number of rows and columns has several impacts on how you proceed. First, the overall size gives you a rough idea of how long your training times are going to be. For a small data set like the rocks versus mines data, training time will be less than a minute, which will facilitate iterating through the process of training and tweaking. If the data set grows to 1,000 x 1,000, the training times will grow to a fraction of a minute for penalized linear regression and a few minutes for an ensemble method. As the data set gets to several tens of thousands of rows and columns, the training times will expand to 3 or 4 hours for penalized linear regression and 12 to 24 hours for an ensemble method. The larger training times will have an impact on your development time because you'll iterate a number of times.

The second important observation regarding row and column counts is that if the data set has many more columns than rows, you may be more likely to get the best prediction with penalized linear regression and vice versa. Chapter 3, "Predictive Model Building: Balancing Performance, Complexity, and Big Data," and the examples you'll run later will give you a better understanding of why that's true.

The next step in the checklist is to determine how many of the columns of data are numeric versus categorical. Listing 2-2 shows code to accomplish this for the rocks versus mine data set. The code runs down each column and adds up the number of entries that are numeric (int or float), the number of entries that are nonempty strings, and the number that are empty. The result is that the first 60 columns contain all numeric values and the last column contains all strings. The string values are the labels. Generally, categorical variables are presented as strings, as in this example. In some cases, binary-valued categorical variables are presented as a 0,1 numeric variable.

Listing 2-2: Determining the Nature of Attributes—rockVmineContents.py (Output: outputRocksVMinesContents.txt)

```
__author__ = 'mike_bowles'
import urllib2
import sys

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
```

```
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])

type = [0]*3
colCounts = []

for col in range(ncol):
    for row in xList:
        try:
            a = float(row[col])
            if isinstance(a, float):
                type[0] += 1
        except ValueError:
            if len(row[col]) > 0:
                type[1] += 1
            else:
                type[2] += 1

    colCounts.append(type)
    type = [0]*3

sys.stdout.write("Col#" + '\t' + "Number" + '\t' +
                 "Strings" + '\t ' + "Other\n")
iCol = 0
for types in colCounts:
    sys.stdout.write(str(iCol) + '\t\t' + str(types[0]) + '\t\t' +
                     str(types[1]) + '\t\t' + str(types[2]) + "\n")
    iCol += 1


Output:
Col#   Number  Strings Other
 0      208      0      0
 1      208      0      0
 2      208      0      0
 3      208      0      0
 4      208      0      0
 5      208      0      0
 6      208      0      0
 7      208      0      0
 8      208      0      0
 9      208      0      0
10      208      0      0
11      208      0      0
 .        .      .      .
 .        .      .      .
 .        .      .      .
```

continues

continued

```
54      208      0       0
55      208      0       0
56      208      0       0
57      208      0       0
58      208      0       0
59      208      0       0
60      0      208       0
```

Statistical Summaries of the Rocks versus Mines Data Set

After determining which attributes are categorical and which are numeric, you'll want some descriptive statistics for the numeric variables and a count of the unique categories in each categorical attribute. Listing 2-3 gives some examples of these two procedures.

Listing 2-3: Summary Statistics for Numeric and Categorical Attributes—rVMSummaryStats.py (Output: outputSummaryStats.txt)

```
__author__ = 'mike_bowles'
import urllib2
import sys
import numpy as np

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []
labels = []

for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])

type = [0]*3
colCounts = []

#generate summary statistics for column 3 (e.g.)
col = 3
colData = []
for row in xList:
    colData.append(float(row[col]))

colArray = np.array(colData)
```

```
colMean = np.mean(colArray)
colsd = np.std(colArray)
sys.stdout.write("Mean = " + '\t' + str(colMean) + '\t\t' +
            "Standard Deviation = " + '\t ' + str(colsd) + "\n")


#calculate quantile boundaries
ntiles = 4

percentBdry = []

for i in range(ntiles+1):
    percentBdry.append(np.percentile(colArray, i*(100)/ntiles))

sys.stdout.write("\nBoundaries for 4 Equal Percentiles \n")
print(percentBdry)
sys.stdout.write(" \n")


#run again with 10 equal intervals
ntiles = 10

percentBdry = []

for i in range(ntiles+1):
    percentBdry.append(np.percentile(colArray, i*(100)/ntiles))

sys.stdout.write("Boundaries for 10 Equal Percentiles \n")
print(percentBdry)
sys.stdout.write(" \n")


#The last column contains categorical variables

col = 60
colData = []
for row in xList:
    colData.append(row[col])

unique = set(colData)
sys.stdout.write("Unique Label Values \n")
print(unique)

#count up the number of elements having each value

catDict = dict(zip(list(unique),range(len(unique))))

catCount = [0]*2

for elt in colData:
    catCount[catDict[elt]] += 1
```

continues

continued

```
sys.stdout.write("\nCounts for Each Value of Categorical Label \n")
print(list(unique))
print(catCount)

Output:
Mean =    0.053892307      Standard Deviation =     0.046415983

Boundaries for 4 Equal Percentiles
 [0.0057999999999999996, 0.024375000000000001, 0.044049999999999999,
 0.064500000000000002, 0.4264]

Boundaries for 10 Equal Percentiles
[0.00579999999999, 0.0141, 0.022740000000, 0.0278699999999,
0.0362200000000, 0.0440499999999, 0.050719999999, 0.0599599999999,
0.0779400000000, 0.10836, 0.4264]
Unique Label Values
set(['R', 'M'])

Counts for Each Value of Categorical Label
['R', 'M']
[97, 111]
```

The first section of the code picks up one column of numeric data, and then generates some statistics for it. The first step is to calculate the mean and standard deviation for the chosen attribute. Knowing these will undergird your intuition as you're developing models.

The next section of code looks for outliers. Here's how that works. Suppose that you're trying to determine whether you've got an outlier in the following list of numbers = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 4]. This example is constructed to have an outlier. The last number (4) is clearly out of scale with the rest of the numbers.

One way to reveal this sort of mismatch is to divide a set of numbers into percentiles. For example, the 25th percentile contains the smallest 25 percent of the data. The 50th percentile contains the smallest 50 percent of the data. The easiest way to visualize forming these groupings is to imagine that the data are sorted into numeric order. The numbers in the preceding list are arranged in numeric order. That makes it easy to see where the percentile boundaries go. Some often used percentiles are given special names. The percentiles defined by dividing the set into equal quarters, fifths, and tenths are called respectively *quartiles*, *quintiles*, and *deciles*.

With the preceding list, it's easy to define the quartiles because the list is ordered and there are eight elements in the list. The first quartile contains 0.1 and 0.15 and so on. Notice how wide these quartiles are. The first quartile has a range of 0.5 (0.15–0.1). The second quartile is roughly the same. However, the last quartile has a range of 4.6, which is 100 times larger than the range of the other quartiles.

You can see similar behavior in the quartile boundaries that are calculated in Listing 2-3. First the program calculates the quartiles. That shows that the upper quartile is much wider than the others. To be more certain, the decile boundaries are also calculated and similarly demonstrate that the upper decile is unusually wide. Some widening is normal because distributions often thin out in the tails.

Visualization of Outliers Using Quantile-Quantile Plot

One way to study outliers in more detail is to plot the distribution of the data in question relative to some reasonable distributions to see whether the relative numbers match up. Listing 2-4 shows how to use the Python function `probplot` to help determine whether the data has outliers or not. The resulting plot shows how the boundaries associated with empirical percentiles in the data compare to the boundaries for the same percentiles of a Gaussian distribution. If the data being analyzed comes from a Gaussian distribution, the point being plotted will lie on a straight line. Figure 2-1 shows that a couple of points from column 4 of the rocks versus mines data are very far from the line. That means that the tails of the rocks versus mines data contain more examples than the tails of a Gaussian density.

Listing 2-4: Quantile-Quantile Plot for 4th Rocks versus Mines Attribute—qqplotAttribute.py

```
__author__ = 'mike bowles'
import numpy as np
import pylab
import scipy.stats as stats
import urllib2
import sys

target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []
labels = []

for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])
```

continues

continued

```
type = [0]*3
colCounts = []

#generate summary statistics for column 3 (e.g.)
col = 3
colData = []
for row in xList:
    colData.append(float(row[col]))


stats.probplot(colData, dist="norm", plot=pylab)
pylab.show()
```

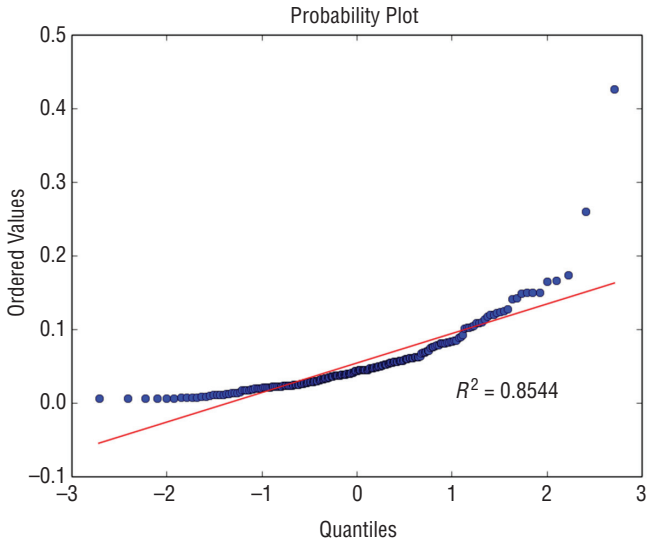

Figure 2-1: Quantile-quantile plot of attribute 4 from rocks versus mines data

What do you do with this information? Outliers may cause trouble either for model building or prediction. After you've trained a model on this data set, you can look at the errors your model makes and see whether the errors are correlated with these outliers. If they are, you can then take steps to correct them. For example, you can replicate the poor-performing examples to force them to be more heavily represented. You can segregate them out and train on them as a separate class. You can also edit them out of the data if they represent an abnormality that won't be present in the data your model will see when deployed. A reasonable process for this might be to generate quartile boundaries during the exploration phase and note potential outliers to get a feel for how much of a problem you might (or might not) have with it. Then when you're evaluating

performance data, use quantile-quantile (Q-Q) plots to determine which points to call outliers for use in your error analysis.

Statistical Characterization of Categorical Attributes

The process just described applies to numeric attributes. But what about categorical attributes? You want to check to see how many categories they have and how many examples there are from each category. You want to learn these things for a couple of reasons. The gender attribute has two possible values (Male and Female), but if the attribute had been the state of the United States, there would have been 50 possible categories. As the number of attributes grows, the complexity of dealing with them mounts. Most binary tree algorithms, which are the basis for ensemble methods, have a cutoff on how many categories they can handle. The popular Random Forests package written by Breiman and Cutler (the inventors of the algorithm) has a cutoff of 32 categories. If an attribute has more than 32 categories, you'll need to aggregate them.

You'll see later that training involves taking a random subset of the data and training a series of models on it. Suppose, for instance, that the category is the state of the United States and that Idaho has only two examples. A random draw of training examples might not get any from Idaho. You need to see those kinds of problems before they occur so that you can address them. In the case of the two Idaho examples, you might merge them with Montana or Wyoming, you might duplicate them, or you might manage the random draw so that you ensure getting Idaho examples (a procedure called *stratified sampling*).

How to Use Python Pandas to Summarize the Rocks Versus Mines Data Set

The Python package Pandas can help automate the process of data inspection and handling. It proves particularly useful for the early stages of data inspection and preprocessing. The Pandas package makes it possible to read data into a specialized data structure called a *data frame*. The data frame is modeled after the CRAN-R data structure of the same name.

NOTE **The Pandas package can be difficult to install because it has a number of dependencies that need to be correctly versioned and each of those has to be correctly matched to one another (and so on). An easy way around this hurdle is to use the Anaconda Python distribution available for free download from Continuum Analytics (`http://continuum.io`). The installation procedures are easy to follow and result in compatible installations of a wide variety of packages for data analysis and machine learning.**

You can think of a data frame as a table or matrix-like structure as in Table 2-1. The data frame is oriented with a row representing a single case (experiment, example, measurement) and columns representing particular attributes. The structure is matrix-like, but not a matrix because the elements in various columns may be of different types. Formally, a matrix is defined over a field (like the real numbers, binary numbers, complex numbers), and all the entries in a matrix are elements from that field. For statistical problems, the matrix is too confining because statistical samples typically have a mix of different types.

The simple example in Table 2-1 has real values in the Attribute 1 column, categorical variables in the Attribute 2 column, and integer variables in the Attribute 3 column. Within a column, the entries are all the same type, but they differ from one column to the next. The data frame structure enables access to individual elements through an index roughly similar to addressing an entry in a Python Numpy array or a list of lists. Similarly, index slicing can be used to address an entry in the array. In addition, the Pandas data frame enables addressing rows and columns by means of their names. This turns out to be very handy, particularly for a small to medium number of columns. (A search on "Pandas introduction" will give you a number of links that can guide you through the basics of using Pandas.)

Listing 2-5 show how simple it is to read in the rocks versus mines CSV file from the UC Irvine Data Repository website. The output shown as part of the listing is truncated from the actual output. You can get the full version by running the code for yourself.

Listing 2-5: Using Python Pandas to Read and Summarize Data—pandasReadSummarize.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#print head and tail of data frame
print(rocksVMines.head())
print(rocksVMines.tail())

#print summary of data frame
summary = rocksVMines.describe()
print(summary)

Output (truncated):

       V0      V1      V2      ...      V57      V58      V59  V60
```

```
0  0.0200  0.0371  0.0428     ...  0.0084  0.0090  0.0032    R
1  0.0453  0.0523  0.0843     ...  0.0049  0.0052  0.0044    R
2  0.0262  0.0582  0.1099     ...  0.0164  0.0095  0.0078    R
3  0.0100  0.0171  0.0623     ...  0.0044  0.0040  0.0117    R
4  0.0762  0.0666  0.0481     ...  0.0048  0.0107  0.0094    R

[5 rows x 61 columns]

         V0      V1      V2     ...      V57     V58     V59  V60
203  0.0187  0.0346  0.0168     ...   0.0115  0.0193  0.0157    M
204  0.0323  0.0101  0.0298     ...   0.0032  0.0062  0.0067    M
205  0.0522  0.0437  0.0180     ...   0.0138  0.0077  0.0031    M
206  0.0303  0.0353  0.0490     ...   0.0079  0.0036  0.0048    M
207  0.0260  0.0363  0.0136     ...   0.0036  0.0061  0.0115    M

[5 rows x 61 columns]



                V0          V1         ...          V58         V59
count  208.000000  208.000000         ...   208.000000  208.000000
mean     0.029164    0.038437         ...     0.007941    0.006507
std      0.022991    0.032960         ...     0.006181    0.005031
min      0.001500    0.000600         ...     0.000100    0.000600
25%      0.013350    0.016450         ...     0.003675    0.003100
50%      0.022800    0.030800         ...     0.006400    0.005300
75%      0.035550    0.047950         ...     0.010325    0.008525
max      0.137100    0.233900         ...     0.036400    0.043900
```

After reading in the file, the first section of the program prints out head and tail. Notice that all the heads have R labels, and the tails have M labels. With this data set, the Rs all come first and the Ms second. Note things like that during your inspection of the data. You'll see in later sections that determining the quality of your models requires sampling the data. Structure in the way the data are stored might need to be factored into your approach for doing subsequent sampling. The last bit of the code snippet prints out summaries of the real-valued columns in the data set.

Pandas makes it possible to automate the steps of calculating mean, variance, and quantiles. Notice that the summary produced by the `describe` function is itself a data frame so that you can automate the process of screening for attributes that have outliers. To do that, you can compare the differences between the various quantiles and raise a flag if any of the differences for an attribute are out of scale with the other differences for the same attributes. The attributes that are shown in the output indicate that several of them have outliers. It would be worth looking to determine how many rows are involved in the outliers. They might all come from a handful of examples. This can point out data that needs to be inspected more closely.

Visualizing Properties of the Rocks versus Mines Data Set

Visualizations can sometimes give you insights into your data that would be difficult to see in tables of numbers. This section introduces several that you may find useful. Some of the visualizations take slightly different forms for classification problems than for regression problems. You'll see the regression variants of the methods in the sections covering the abalone data set and the wine quality data set.

Visualizing with Parallel Coordinates Plots

One visualization that is useful for problems with more than a few attributes is called a *parallel coordinates plot*. Figure 2-2 depicts the construction of a parallel coordinates plot. The vector of numbers on the right-hand side of the figure represents a row of attribute data from a machine learning data set. The parallel coordinates plot of that vector of numbers is shown in the line plot in Figure 2-2. The line plots the value of each attribute versus its index. The parallel coordinates plot for the whole data set has a line for each row of attributes in the data set. Color-coding based on the labels can help you see some types of systematic relationships between the attribute values and the labels. Plot the real-valued attributes from a row versus the index of the attribute. (Search "parallel coordinates" and check out the Wikipedia page for some more examples.)

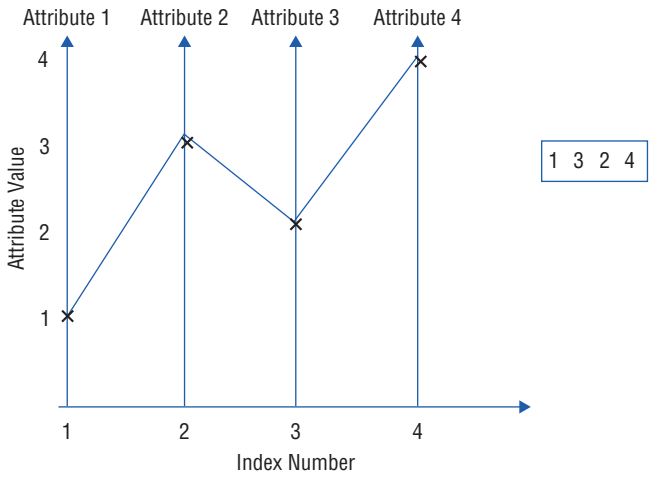

Figure 2-2: Constructing a parallel coordinates plot

Listing 2-6 shows how this process works for the rocks versus mines data set. Figure 2-3 shows the resulting plotted line graphs. The lines are color coded according to their labels: blue for R (rock), and red for M (mine). Sometimes a plot of this type will show clear areas of separation between the classes. The famous "Iris data" show very clear separation that machine learning algorithms will exploit for classification purposes. For the rocks versus mines data set, no extremely clear separation is evident in the line plot, but there are some areas where the blues and reds are separated. Along the bottom of the plot, the blues stand out a bit, and in the range of attribute indices from 30 to 40, the blues are somewhat higher than the reds. These kinds of insights can help in interpreting and confirming predictions made by your trained model.

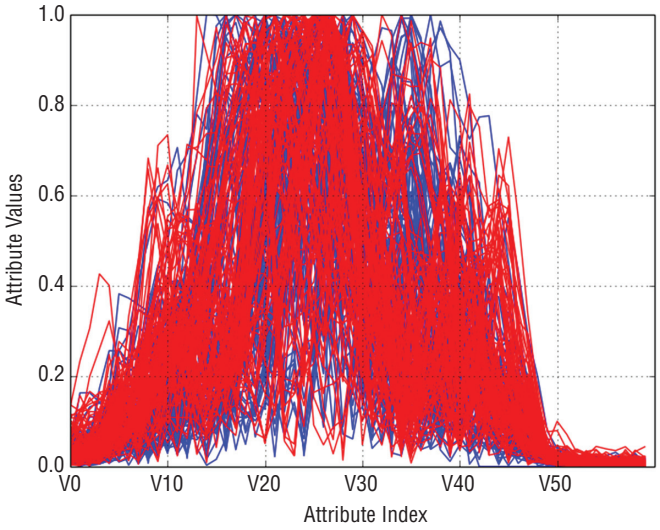

Figure 2-3: Parallel coordinates graph of rocks versus mines attributes

Listing 2-6: Parallel Coordinates Graph for Real Attribute Visualization—linePlots.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

for i in range(208):
    #assign color based on "M" or "R" labels
    if rocksVMines.iat[i,60] == "M":
        pcolor = "red"
```

continues

continued

```
    else:
        pcolor = "blue"

    #plot rows of data as if they were series data
    dataRow = rocksVMines.iloc[i,0:60]
    dataRow.plot(color=pcolor)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

Visualizing Interrelationships between Attributes and Labels

Another question you might ask of the data is how the various attributes relate to one another. One quick way to get an idea of pair-wise relationships is to cross-plot the attributes with the labels. Listing 2-7 shows what's required to generate cross-plots for a couple of representative pairs of attributes. These cross-plots (also called *scatter plots*) show you how closely related the pairs of variables are.

Listing 2-7: Cross Plotting Pairs of Attributes—corrPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
dataRow2 = rocksVMines.iloc[1,0:60]
dataRow3 = rocksVMines.iloc[2,0:60]

plot.scatter(dataRow2, dataRow3)


plot.xlabel("2nd Attribute")
plot.ylabel(("3rd Attribute"))
plot.show()

dataRow21 = rocksVMines.iloc[20,0:60]

plot.scatter(dataRow2, dataRow21)


plot.xlabel("2nd Attribute")
plot.ylabel(("21st Attribute"))
plot.show()
```

Figures 2-4 and 2-5 show the scatter plots for two pairs of attributes from the rocks versus mines data set. The rocks versus mines attributes are samples from sonar returns. The sonar signal is called a *chirped* waveform because it's a pulse that starts at low frequency and rises higher over the duration of the pulse. The attributes in the rocks versus mines data set are time samples of the sound waves that bounce off the rock or mine. These returned acoustic signals bear the same relationship between time and frequency as the outgoing transmission. The 60 attributes in the rocks versus mines data are samples of the return taken at 60 different times (and therefore 60 different frequencies). You'd expect that adjacent attributes would be more correlated than attributes separated in time from one another because there's not much difference in frequency between adjacent time samples.

This intuition is borne out in Figures 2-4 and 2-5. The points in the scatter plot in Figure 2-4 are more closely grouped around a straight line than those in Figure 2-5. If you want to develop your intuition about the relation between numeric correlation and the shape of the scatter plot, just search "correlation" and have a look at the Wikipedia page that comes up. That shows some scatter plots and the associated numeric correlation. Basically, if the points in the scatter plot lie along a thin straight line, the two variables are highly correlated; if they form a ball of points, they're uncorrelated.

You can apply the same principle to plotting the correlation between each of the attributes and the target. For a problem where the targets are real numbers (a regression problem), the plots look much the same as Figures 2-4 and 2-5. The rocks versus mines data set is a classification problem. The targets are two-valued. You can follow the same general procedure.

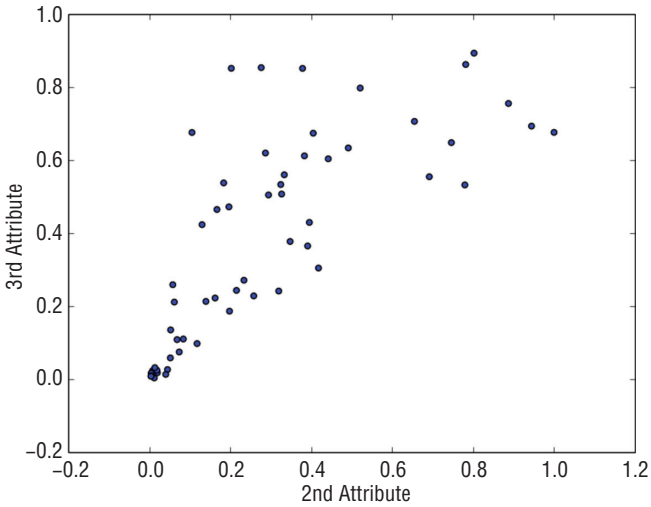

Figure 2-4: Cross-plot of rocks versus mines attributes 2 and 3

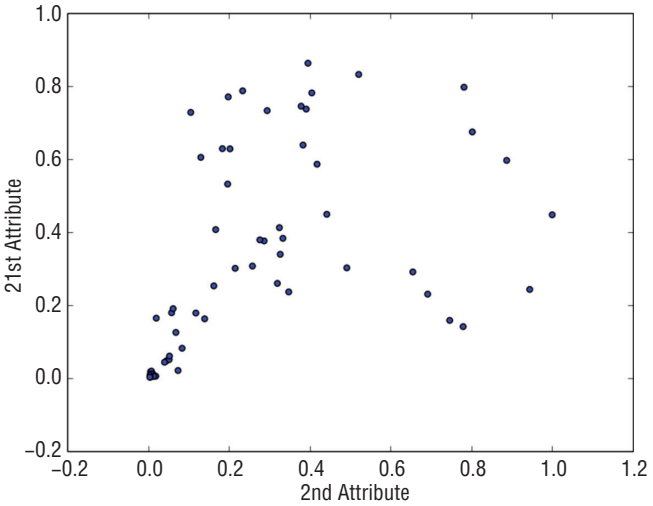

Figure 2-5: Cross- plot of rocks versus mines attributes 2 and 21

Listing 2-8 shows the code for plotting a scatter plot between the targets and attribute 35. The idea of using attribute 35 for the example showing correlation with the target came from the parallel coordinates graph in Figure 2-3. That graph shows some separation between the rocks and mines (red lines and blue lines) around index value 35. The correlation between the target and one of the attributes around that index value should also show some separation. Figures 2-6 and 2-7 plot the results.

Listing 2-8: Correlation between Classification Target and Real Attributes—targetCorr.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
from random import uniform
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#change the targets to numeric values
target = []
for i in range(208):
    #assign 0 or 1 target value based on "M" or "R" labels
    if rocksVMines.iat[i,60] == "M":
        target.append(1.0)
```

```
    else:
        target.append(0.0)

#plot 35th attribute
dataRow = rocksVMines.iloc[0:208,35]
plot.scatter(dataRow, target)

plot.xlabel("Attribute Value")
plot.ylabel("Target Value")
plot.show()

#
#To improve the visualization, this version dithers the points a little
# and makes them somewhat transparent
target = []
for i in range(208):

#assign 0 or 1 target value based on "M" or "R" labels
    # and add some dither

    if rocksVMines.iat[i,60] == "M":
        target.append(1.0 + uniform(-0.1, 0.1))
    else:
        target.append(0.0 + uniform(-0.1, 0.1))

    #plot 35th attribute with semi-opaque points
dataRow = rocksVMines.iloc[0:208,35]
plot.scatter(dataRow, target, alpha=0.5, s=120)

plot.xlabel("Attribute Value")
plot.ylabel("Target Value")
plot.show()
```

The plots show what happens if you make a list corresponding to the list of R or M targets but with the substitution of 1 for M and 0 for R. Then you can plot a scatter plot as shown in Figure 2-6. Figure 2-6 highlights a common problem with cross-plots. When one of the variables being plotted takes on a small number of values, the points get plotted on top of one another. If there are a lot of them, you just get a thick dark line, and you don't get a feel for how the points are distributed along the line.

The code in Listing 2-8 generates a second plot with two small changes to overcome this problem. A small random number is added to each of the points and takes a small number of discrete values (the targets in this case). The target values are either 0 or 1 by construction. In the code, you'll see that the added random number is uniformly distributed between –0.1 and 0.1. That spreads the points apart, but not so far as to confuse the two lines. Second, the points are plotted with alpha=0.5 in order that the points are only partially opaque.

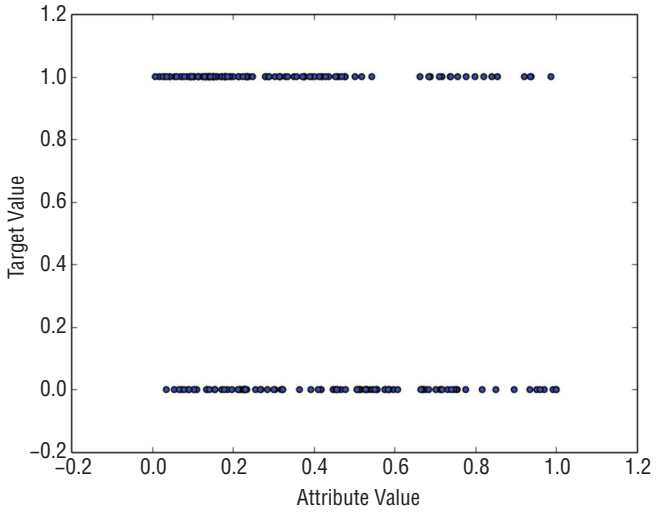

Figure 2-6: Target-attribute cross-plot

Then any overplotting shows up as a darkened region in the scatter plot. You may have to adjust these numbers a little to make the plot show you what you need to know.

Figure 2-7 shows the effect of these two alterations. Notice the somewhat higher concentration of attribute 35 on the left end of the upper band of points, whereas the points are more uniformly spread from right to left in the lower band. The upper band of points corresponds to mines. The lower band corresponds to rocks. You could build a classifier for this problem by testing whether attribute 35 is greater than or less than 0.5. If it is greater than 0.5 call it a rock,

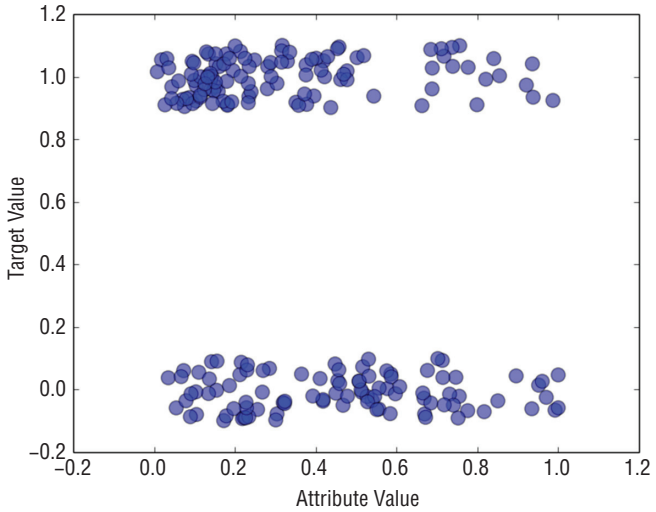

Figure 2-7: Target-attribute cross-plot with point dither and partial opacity

and if it is less than 0.5, call it a mine. The examples where attribute 35 is less than 0.5 contain a higher concentration of mines than rock, and the examples where attribute 35 is less than 0.5 contain a lower density, so you'd get better performance than you would with random guessing.

NOTE **You'll see much more systematic approaches to building classifiers in Chapters 5, "Building Predictive Models Using Penalized Linear Methods," and Chapter 7, "Building Ensemble Models with Python." They'll use all the attributes instead of just one or two. However, when you look at what they're using to make their decisions, you can refer back to these types of studies to help you gain confidence that what they're doing is sensible.**

The degree of correlation between two attributes (or an attribute and a target) can be quantified using Pearson's correlation coefficient. Pearson's correlation coefficient is defined for two equal length vectors u and v, as follows (see Equations 2-1 and 2-2). First subtract the mean value of u from all the elements of u (see Equation 2-3) and do the same for v.

$$u = \begin{matrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{matrix}$$

Equation 2-1: Elements of a vector u

$$\overline{u} = avg(u)$$

Equation 2-2: Average values of the entries in u

$$\Delta u = \begin{matrix} u_1 - \overline{u} \\ u_2 - \overline{u} \\ \vdots \\ u_n - \overline{u} \end{matrix}$$

Equation 2-3: Subtract the average from each element in u.

For the second vector v, define a vector Δv in the same way as Δu was defined corresponding to the vector u.

Then Pearson's correlation between u and v is shown in Equation 2-4.

$$corr(u,v) = \frac{\Delta u^T * \Delta v}{\sqrt{(\Delta u^T * \Delta u) * (\Delta v^T * \Delta v)}}$$

Equation 2-4: Definition of Pearson's correlation coefficient

Listing 2-9 shows a Python implementation of this function to calculate correlation for the pairs of attributes plotted in Figures 2-3 and 2-5. The correlation numbers agree with plotted data. The attributes that have close index numbers have relatively higher correlations than those that are separated further.

Listing 2-9: Pearson's Correlation Calculation for Attributes 2 versus 3 and 2 versus 21—corrCalc.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from math import sqrt
import sys
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
dataRow2 = rocksVMines.iloc[1,0:60]
dataRow3 = rocksVMines.iloc[2,0:60]
dataRow21 = rocksVMines.iloc[20,0:60]

mean2 = 0.0; mean3 = 0.0; mean21 = 0.0
numElt = len(dataRow2)
for i in range(numElt):
    mean2 += dataRow2[i]/numElt
    mean3 += dataRow3[i]/numElt
    mean21 += dataRow21[i]/numElt

var2 = 0.0; var3 = 0.0; var21 = 0.0
for i in range(numElt):
    var2 += (dataRow2[i] - mean2) * (dataRow2[i] - mean2)/numElt
    var3 += (dataRow3[i] - mean3) * (dataRow3[i] - mean3)/numElt
    var21 += (dataRow21[i] - mean21) * (dataRow21[i] - mean21)/numElt

corr23 = 0.0; corr221 = 0.0
for i in range(numElt):

corr23 += (dataRow2[i] - mean2) * \
              (dataRow3[i] - mean3) / (sqrt(var2*var3) * numElt)
    corr221 += (dataRow2[i] - mean2) * \
                (dataRow21[i] - mean21) / (sqrt(var2*var21) * numElt)

sys.stdout.write("Correlation between attribute 2 and 3 \n")
print(corr23)
sys.stdout.write(" \n")
```

```
sys.stdout.write("Correlation between attribute 2 and 21 \n")
print(corr221)
sys.stdout.write(" \n")


Output:
Correlation between attribute 2 and 3
0.770938121191

Correlation between attribute 2 and 21
0.466548080789
```

Visualizing Attribute and Label Correlations Using a Heat Map

Calculating the correlations and printing them or drawing cross-plots works fine for a few correlations, but it is difficult to get a grasp of a large table of numbers, and it is difficult to squeeze all the cross-plots onto a page if the problem has 100 attributes.

One way to check correlations with a large number of attributes is to calculate the Pearson's correlation coefficient for pairs of attributes, arrange those correlations into a matrix where the ij-th entry is the correlation between the ith attribute and the jth attribute, and then plot them in a heat map. Listing 2-10 gives the code to make this plot. Figure 2-8 shows the plot. The light areas along the diagonal confirm that attributes close to one another in index have relatively high correlations. As mentioned earlier, this is due to the way in which the data are generated. Close indices are sampled at short time intervals from one another and consequently have similar frequencies. Similar frequencies reflect off the targets similarly (and so on).

Listing 2-10: Presenting Attribute Correlations Visually—sampleCorrHeatMap.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
corMat = DataFrame(rocksVMines.corr())

#visualize correlations using heatmap
plot.pcolor(corMat)
plot.show()
```

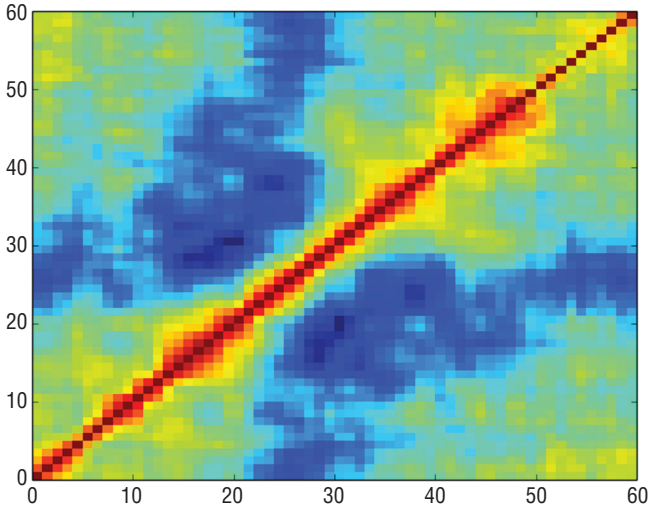

Figure 2-8: Heat map showing attribute cross-correlations

Perfect correlation (correlation = 1) between attributes means that you may have made a mistake and included the same thing twice. Very high correlation between a set of attributes (pairwise correlations > 0.7) is known as *multicollinearity* and can lead to unstable estimates. Correlation with the targets is a different matter. Having an attribute that's correlated with the target generally indicates a predictive relation.

Summarizing the Process for Understanding Rocks versus Mines Data Set

In the process of understanding the rocks versus mines data set, this section has introduced a number of tools for you to use to gain understanding and intuition about your data sets. The section has gone into some detail to make their derivation and use clear. The next sections will use several of these same tools to inspect the other data sets that the book will use to develop machine learning algorithms. Since you're now familiar with the tools for doing data inspection, the next sections will comment on the tools only to the extent that they need to be modified because of the different nature of a problem.

Real-Valued Predictions with Factor Variables: How Old Is Your Abalone?

Most of the tools you've seen used for understanding the problem of detecting unexploded mines can be applied to regression problems. Predicting the age of an abalone, given physical measurements, provides an example of such a

problem. The abalone attributes also include an attribute that is a factor variable, which will illustrate the differences involved with factor variables.

The abalone data set poses the problem of predicting the age of an abalone by taking several measurements. It is possible to get a precise reading on the age of an abalone by slicing the shell and counting growth rings, much like gauging the age of a tree by counting rings. The problem for scientists studying abalone populations is that it is expensive and time-consuming to slice the shells and count the rings under a microscope. It would be more convenient and economical to be able to make simple physical measurements like length, width, weight, and so forth and then to use a predictive model to process the measurements and make an accurate determination of the age of the abalone. There are a myriad of scientific applications for predictive analytics, and one of the benefits of studying machine learning is being able to contribute to an interesting array of different problems.

The data for this problem are available through the UC Irvine Data Repository. The URL for this data set is `http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data`. This data set is in the form of a comma-delimited file with no column headers. The names of the columns are in a separate file. Listing 2-11 reads the abalone data set into a Pandas data frame and runs through some of the same analyses that you saw in "Classification Problems: Detecting Unexploded Mines Using Sonar." For the rocks versus mines data set, the column names were somewhat generic because of the nature of the data. For the abalone data set, the different columns of data have meanings that can be critical to cultivating an intuitive understanding of your progress toward an acceptable model. For this reason, you'll see in the code that the column names have been copy-pasted into the code and attached to the data set to help you make sense of what subsequent machine learning algorithms are doing to make predictions. The columns of data available for building a predictive model are Sex, Length, Diameter, Height, Whole Weight, Shucked Weight, Viscera Weight, Shell Weight, and Rings. The last column, Rings, is measured by the laborious process of sawing the shell and counting under a microscope. This is the usual arrangement for a supervised learning problem. You've got a special data set for which the answer is known so as to build a model that will generate predictions when the answer is not known.

In addition to showing the code for producing the summaries, Listing 2-11 shows the printed output from the summarization. The first section prints the head and tail of the data set. Only the head is shown in the output to save space. When you run the code for yourself, you'll see both. Most of the data frame is filled with floating-point numbers. The first column, which contains the gender of the animal, contains the letters M (male), F (female), and I (indeterminate). The gender of an abalone is not determined at birth, but after it has matured a little. Therefore, the gender is indeterminate for younger abalones. The gender of the abalone is a three-valued categorical variable. Categorical attributes require special attention. Some algorithms only deal with real-valued attributes (for

example, support vector machines, K-nearest neighbors, and penalized linear regression, which is introduced in Chapter 4). Chapter 4 discusses techniques for translating categorical variables into real-valued variables so that you can employ these algorithms. Listing 2-11 also shows the column-by-column statistical summaries for the real-valued attributes.

Listing 2-11: Read and Summarize the Abalone Data Set—abaloneSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole weight','Shucked weight', 'Viscera weight',
                   'Shell weight', 'Rings']


print(abalone.head())
print(abalone.tail())

#print summary of data frame
summary = abalone.describe()
print(summary)

#box plot the real-valued attributes
#convert to array for plot routine
array = abalone.iloc[:,1:9].values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges"))
show()

#the last column (rings) is out of scale with the rest
# - remove and replot
array2 = abalone.iloc[:,1:8].values
boxplot(array2)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges"))
show()

#removing is okay but renormalizing the variables generalizes better.
#renormalize columns to zero mean and unit standard deviation
#this is a common normalization and desirable for other operations
# (like k-means clustering or k-nearest neighbors
```

```
abaloneNormalized = abalone.iloc[:,1:9]


for i in range(8):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]

abaloneNormalized.iloc[:,i:(i + 1)] = (
                    abaloneNormalized.iloc[:,i:(i + 1)] - mean) / sd


array3 = abaloneNormalized.values
boxplot(array3)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Printed Output: (partial)
  Sex  Length  Diameter  Height  Whole wt  Shucked wt  Viscera wt
0   M   0.455     0.365   0.095    0.5140      0.2245      0.1010
1   M   0.350     0.265   0.090    0.2255      0.0995      0.0485
2   F   0.530     0.420   0.135    0.6770      0.2565      0.1415
3   M   0.440     0.365   0.125    0.5160      0.2155      0.1140
4   I   0.330     0.255   0.080    0.2050      0.0895      0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7
     Sex  Length  Diameter  Height  Whole weight  Shucked weight
4172   F   0.565     0.450   0.165        0.8870          0.3700
4173   M   0.590     0.440   0.135        0.9660          0.4390
4174   M   0.600     0.475   0.205        1.1760          0.5255
4175   F   0.625     0.485   0.150        1.0945          0.5310
4176   M   0.710     0.555   0.195        1.9485          0.9455

      Viscera weight  Shell weight  Rings
4172          0.2390        0.2490     11
4173          0.2145        0.2605     10
4174          0.2875        0.3080      9
4175          0.2610        0.2960     10
4176          0.3765        0.4950     12
            Length     Diameter       Height    Whole wt  Shucked wt
count  4177.000000  4177.000000  4177.000000 4177.000000 4177.000000
mean      0.523992     0.407881     0.139516    0.828742    0.359367
std       0.120093     0.099240     0.041827    0.490389    0.221963
min       0.075000     0.055000     0.000000    0.002000    0.001000
25%       0.450000     0.350000     0.115000    0.441500    0.186000
50%       0.545000     0.425000     0.140000    0.799500    0.336000
```

continues

continued

```
75%         0.615000      0.480000      0.165000     1.153000     0.502000
max         0.815000      0.650000      1.130000     2.825500     1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

As an alternative to the listing of the statistical summaries, Listing 2-11 generates box plots for each of the real-valued columns of data. The first of these is shown in Figure 2-9. In Figure 2-9, the statistical summaries are represented by box plots, which are also called *box and whisker* plots. These plots show a small rectangle with a red line through it. The red line marks the median value (or 50th percentile) for the column of data. The top and bottom of the rectangle mark the 25th percentile and the 75th percentile, respectively. You can compare the numbers in the printed summary to the levels in the box plot to confirm this. Above and below the box, you'll see small horizontal ticks, the so-called whiskers. These are drawn in at levels that are 1.4 times the interquartile spacing above and below the box. Interquartile spacing is the difference between the 75th percentile and the 25th percentile. In other words, the space between the top of the box and the upper whisker is 1.4 times the height of the box. The 1.4x spacing for the whisker is adjustable; see the box plot documentation. You'll

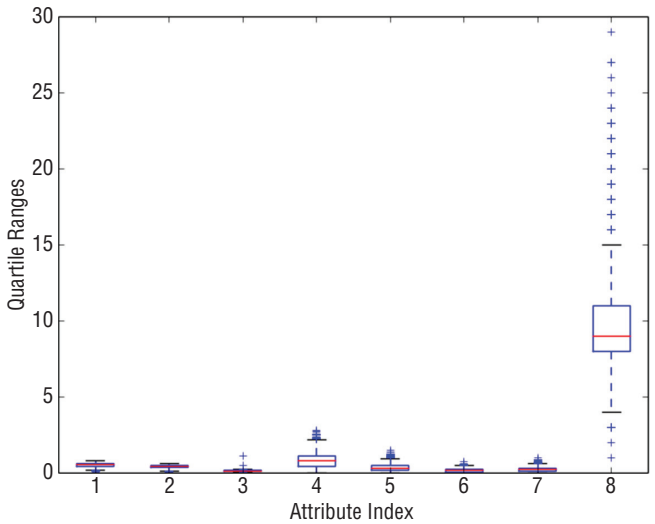

Figure 2-9: Box plot of real-valued attributes from abalone data set

notice that in some cases the whiskers are closer than the 1.4x spacing. For these cases the data values do not extend all the way to the calculated whisker locations. In these cases, the whisker is placed at the most extreme data point. In other cases, the data extend for a considerable distance beyond the calculated whisker locations. These points can be considered outliers.

The box plot in Figure 2-9 is a faster, more visual way to identify outliers than the printed data, but the scale on the rings attributes (the rightmost box plot) causes the other attributes to be compressed (making them hard to see). One way to deal with this is to simply eliminate the larger-scale attributes. The result of that is shown in Figure 2-10. But that approach is unsatisfying because it doesn't automate or scale very well.

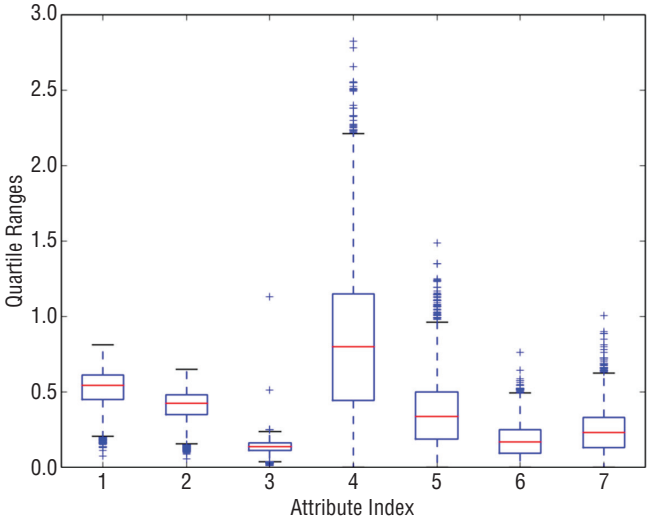

Figure 2-10: Box plot of real-valued attributes from the abalone data set

The last section of the code in Listing 2-11 normalizes all the data columns before box plotting. *Normalization* in this case means centering and scaling each column so that a unit of attribute number 1 means the same thing as a unit of attribute number 2. A number of algorithms and operations in data science require this type of normalization. For example, K-means clustering builds clusters based on vector distance between rows of data. Distance is measured by subtracting one point from another and squaring. If the units are different, the numeric distances are different. The distance to the grocery store can be 1 if measured in miles or 5,280 if measured in feet. The normalization indicated in Listing 2-11 adjusts the variables so that they all have 0 mean and a standard deviation of 1. This is a very common normalization. The calculations for the normalization make use of the numbers generated by the `summary()` function. The results are plotted in Figure 2-11.

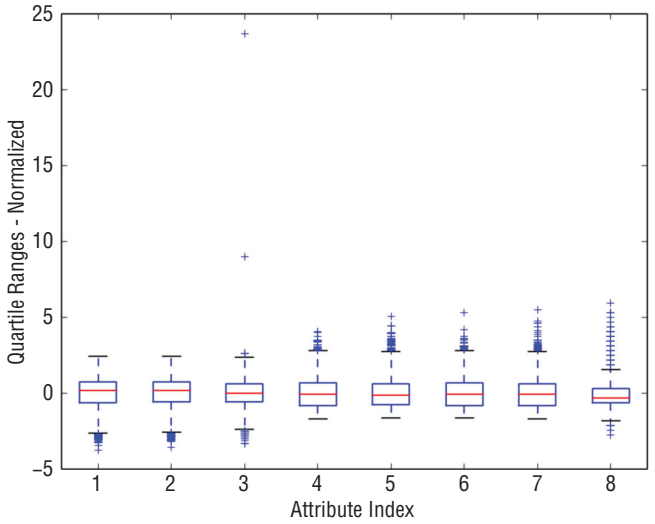

Figure 2-11: Box plot of normalized abalone attributes

Notice that normalizing to standard deviation of 1.0 does not mean that the data all fit between –1.0 and +1.0. It more or less places the lower and upper edges of the boxes at –1.0 and +1.0, but much of the data are outside these boundaries.

Parallel Coordinates for Regression Problems—Visualize Variable Relationships for Abalone Problem

The next step is to get some ideas about the relationship among the attributes and between attributes and labels. For the rocks versus mines data, the color-coded parallel coordinates plot portrayed these relationships graphically. That approach needs some modification to work for the abalone problem. Rocks versus mines was a classifier problem. The parallel coordinates plot for that problem color-coded the lines representing rows of data according to their true classification. That helps to visualize the relationship between prediction and predictors. The abalone problem is a regression problem, so the color-coding in this example needs to be shades of color corresponding to higher or lower target values. To assign shades of color to real values, the real values need to be compressed into the interval [0.0, 1.0]. Listing 2-12 uses the min and max values generated by the `summary()` function from Pandas to accomplish this. Figure 2-12 shows the results.

Listing 2-11: Parallel Coordinate Plot for Abalone Data—abaloneParallelPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
```

```
import matplotlib.pyplot as plot
from math import exp

target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole Wt', 'Shucked Wt',
                   'Viscera Wt', 'Shell Wt', 'Rings']
#get summary to use for scaling
summary = abalone.describe()
minRings = summary.iloc[3,7]
maxRings = summary.iloc[7,7]
nrows = len(abalone.index)

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = abalone.iloc[i,1:8]
    labelColor = (abalone.iloc[i,8] - minRings) / (maxRings - minRings)
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()

#renormalize using mean and standard variation, then compress
# with logit function
meanRings = summary.iloc[1,7]
sdRings = summary.iloc[2,7]

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = abalone.iloc[i,1:8]
    normTarget = (abalone.iloc[i,8] - meanRings)/sdRings
    labelColor = 1.0/(1.0 + exp(-normTarget))
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

The parallel coordinates plot in Figure 2-12 illustrates a direct relationship between abalone age (number of shell rings) and the attributes available for predicting age. The color scale used to produce this plot ranges from very dark reddish brown through lighter shades, yellow, light blue, and very dark blue. The box plot in Figure 2-11 shows that the maximum and minimum values are widely separated from the bulk of the data. This has the effect of compressing the scale so that most of the data are mid-range on the color scale. Nonetheless,

Figure 2-12 indicates significant correlation between each of the attributes and the number of rings measured for each of the examples. Similar shades of color are grouped together at similar values of several of the attributes. This correlation suggests that you'll be able to build an accurate predictive model. Contrary to the generally favorable correlation between attributes and target, some faint blue lines are mixed among the darker orange areas of the graph, indicating that there are some examples that will be difficult to correctly predict.

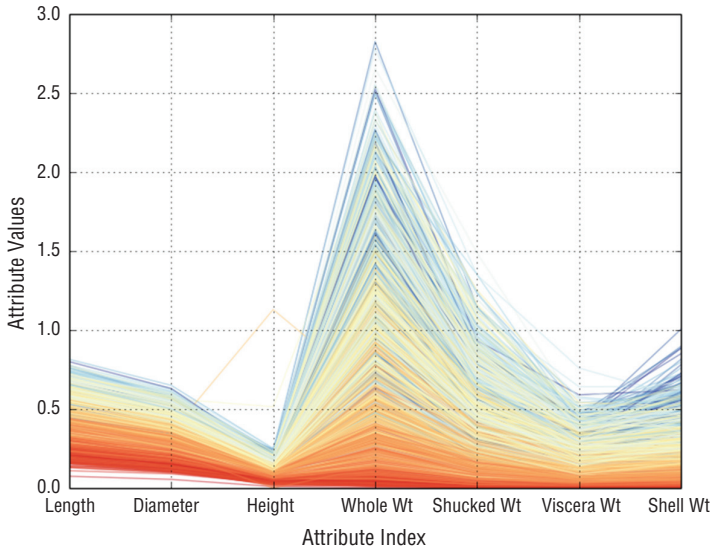

Figure 2-12: Color-coded parallel coordinate plot for abalone

Changing the color mapping can help you visualize relationships at different levels of target values. The last section of the code in Listing 2-11 uses the normalization that you saw used in the box plot graphs. That normalization doesn't make all the values fit between 0 and 1. For one thing, the resulting values take as many negative values as positive ones. The program in Listing 2-11 employs the logit transform to get values in (0, 1). The logit transform is given by the expression shown in Equation 2-5. The plot for this function is given in Figure 2-13.

$$logit\ transform(x) = 1 / (1 + e^{-x})$$

Equation 2-5: Using logit transform for soft range compression

The plot for this function is given in Figure 2-13. As you can see, the logit transform maps large negative values to 0 (almost) and large positive numbers to 1 (almost); it maps 0 to 0.5. You'll see the logit function again in Chapter 4, where it plays a critical role relating a linear function to a probability.

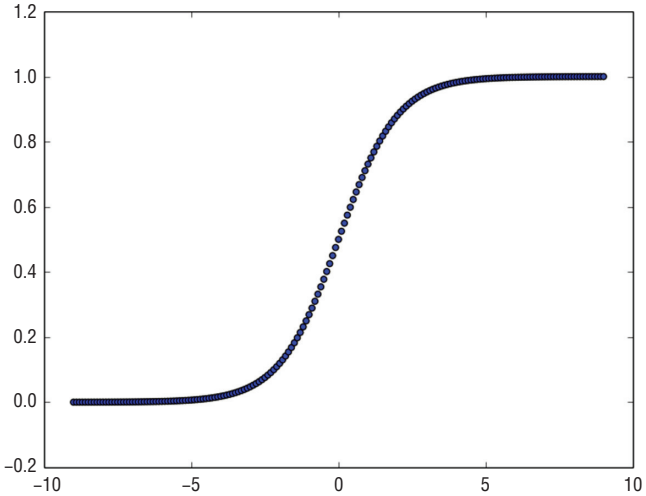

Figure 2-13: Graph of the logit function

Figure 2-14 shows the results of these steps. These transformations have resulted in better usage of the full range of colors available. Notice that there are several darker blue lines (corresponding to specimens with large numbers of rings) mixed in among lighter blue examples, and even yellow and light red specimens for the graphs in the area of Whole Weight and Shucked Weight.

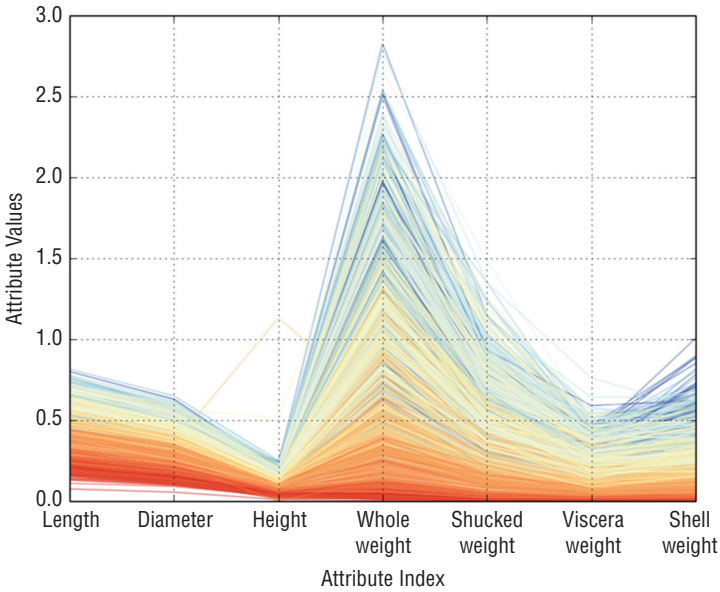

Figure 2-14: Parallel coordinate plot for the abalone data

That suggests that those attributes might not be enough to correctly predict the ages (number of rings) in the older specimens. Fortunately, some of the other attributes (Diameter and Shell Weight) do a better job of correctly ordering the dark blue lines. Those observations will prove helpful when you're analyzing the prediction errors later.

How to Use Correlation Heat Map for Regression—Visualize Pair-Wise Correlations for the Abalone Problem

The last step is to have a look at the correlations between the various attributes and between the attributes and the targets. Listing 2-12 shows the code for generating a correlation heat map and a correlation matrix for the abalone data. These calculations follow the same method outlined for the rocks versus mines data, but with one important difference: Because the abalone problem calls for making real number predictions, the correlation calculations can include the targets in the correlation matrix.

Listing 2-12: Correlation Calculations for Abalone Data—abaloneCorrHeat.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole weight', 'Shucked weight',
                   'Viscera weight', 'Shell weight', 'Rings']

#calculate correlation matrix
corMat = DataFrame(abalone.iloc[:,1:9].corr())
#print correlation matrix
print(corMat)

#visualize correlations using heatmap
plot.pcolor(corMat)
plot.show()
```

```
                  Length  Diameter    Height  Whole Wt  Shucked Wt
Length          1.000000  0.986812  0.827554  0.925261    0.897914
Diameter        0.986812  1.000000  0.833684  0.925452    0.893162
Height          0.827554  0.833684  1.000000  0.819221    0.774972
Whole weight    0.925261  0.925452  0.819221  1.000000    0.969405
Shucked weight  0.897914  0.893162  0.774972  0.969405    1.000000
Viscera weight  0.903018  0.899724  0.798319  0.966375    0.931961
```

```
Shell weight     0.897706   0.905330   0.817338    0.955355    0.882617
Rings            0.556720   0.574660   0.557467    0.540390    0.420884

                 Viscera weight  Shell weight      Rings
Length                 0.903018      0.897706   0.556720
Diameter               0.899724      0.905330   0.574660
Height                 0.798319      0.817338   0.557467
Whole weight           0.966375      0.955355   0.540390
Shucked weight         0.931961      0.882617   0.420884
Viscera weight         1.000000      0.907656   0.503819
Shell weight           0.907656      1.000000   0.627574
Rings                  0.503819      0.627574   1.000000
```

Figure 2-15 shows the correlation heat map. In this map, red indicates high correlation, and blue represents weak correlation. The targets (the number of rings in the shell) are the last item, which is the top row of the heat map and the rightmost column. The blue values in those positions mean that the attributes are weakly correlated with the targets. The light blue corresponds to the correlation between the target and the shell weight. That confirms what you saw in the parallel coordinates plot. The reddish values in the other off-diagonal cell in Figure 2-15 indicate that the attributes are highly correlated with one another. This somewhat contradicts the picture given by the parallel coordinates map where visually the correspondence between the target and the attributes seemed fairly tight. Listing 2-12 shows the numeric values for correlation.

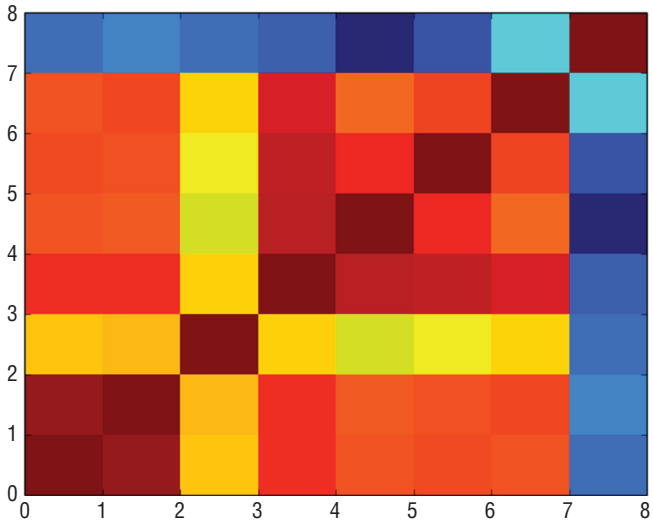

Figure 2-15: Correlation heat map for the abalone data

In this section you've seen how to modify the tools described for a classification problem (rocks versus mines) to a regression problem (abalone). The

modifications all stemmed from the basic difference between the two problem types—labels that are real numbers for a regression problem versus labels that are two-valued for a binary classification problem. The next section will conduct the same set of studies on a regression problem having all numeric attributes. Because it's a regression problem, the same tools used in this section for the abalone problem can be used. Because it has all numeric attributes, all of the attributes can be included in the studies, like correlation and plotting along the real number line.

Real-Valued Predictions Using Real-Valued Attributes: Calculate How Your Wine Tastes

The wine taste data set contains data for approximately 1,500 red wines. For each wine there are a number of measurements of chemical composition, including things like alcohol content, volatile acidity, and sulphites. Each wine also has a taste score determined by averaging the scores given by three professional wine tasters. The problem is to build a model that will incorporate the chemical measurements and predict taste scores to match those given by the human tasters.

Listing 2-13 shows the code for producing summaries of the wine data set. The code prints out a numeric summary of the data, which is included at the bottom of the listing. The code also generates a box plot of the normalized variables so that you can visualize the outliers in the data. Figure 2-16 shows the box plots. The numeric summaries and the box plots indicate numerous outlying values. This is something to keep in mind during training on this data set. When analyzing the performance of the trained models, these outlying examples will be one place to look to understand the source of errors in your models.

Listing 2-13: Wine Data Summary—wineSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
 "learning-databases/wine-quality/winequality-red.csv")
wine = pd.read_csv(target_url,header=0, sep=";")

print(wine.head())

#generate statistical summaries
```

```
summary = wine.describe()
print(summary)

wineNormalized = wine
ncols = len(wineNormalized.columns)

for i in range(ncols):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]

wineNormalized.iloc[:,i:(i + 1)] = \
        (wineNormalized.iloc[:,i:(i + 1)] - mean) / sd
array = wineNormalized.values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Output - [filename - wineSummary.txt]
   fixed acidity  volatil acid  citric acid  resid sugar  chlorides
0            7.4          0.70         0.00          1.9      0.076
1            7.8          0.88         0.00          2.6      0.098
2            7.8          0.76         0.04          2.3      0.092
3           11.2          0.28         0.56          1.9      0.075
4            7.4          0.70         0.00          1.9      0.076

   free sulfur dioxide  tot sulfur dioxide  density    pH  sulphates
0                   11                  34   0.9978  3.51       0.56
1                   25                  67   0.9968  3.20       0.68
2                   15                  54   0.9970  3.26       0.65
3                   17                  60   0.9980  3.16       0.58
4                   11                  34   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
       fixed acidity  volatile acidity  citric acid  residual sugar
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

        chlorides  free sulfur dioxide tot sulfur dioxide   density
```

continues

continued

```
count  1599.000000          1599.000000      1599.000000 1599.000000
mean      0.087467            15.874922        46.467792    0.996747
std       0.047065            10.460157        32.895324    0.001887
min       0.012000             1.000000         6.000000    0.990070
25%       0.070000             7.000000        22.000000    0.995600
50%       0.079000            14.000000        38.000000    0.996750
75%       0.090000            21.000000        62.000000    0.997835
max       0.611000            72.000000       289.000000    1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

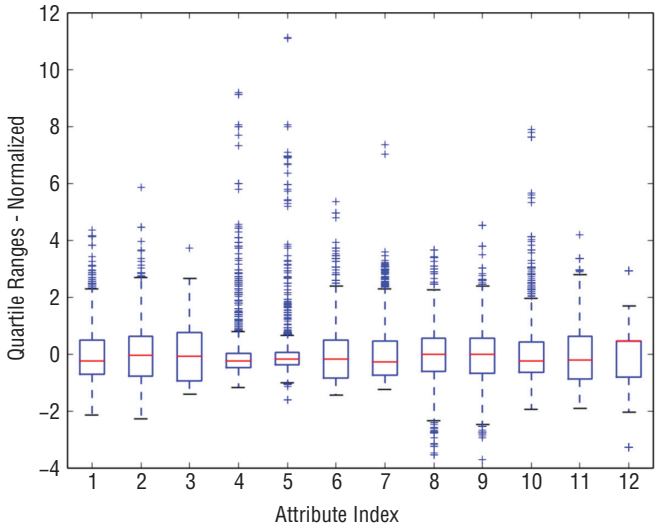

Figure 2-16: Attribute and target box plots of normalized wine data

A color-coded parallel coordinates plot for the wine data will give some idea of how well correlated the attributes are with the targets. Listing 2-14 shows the code for producing that plot. Figure 2-17 shows the resulting parallel coordinates plot. The plot in Figure 2-17 suffers from compressing the graph along the variable directions that have smaller scale values.

To overcome this limitation, Listing 2-14 normalizes the wine data and re-plots it. Figure 2-18 shows the resulting parallel coordinates plot.

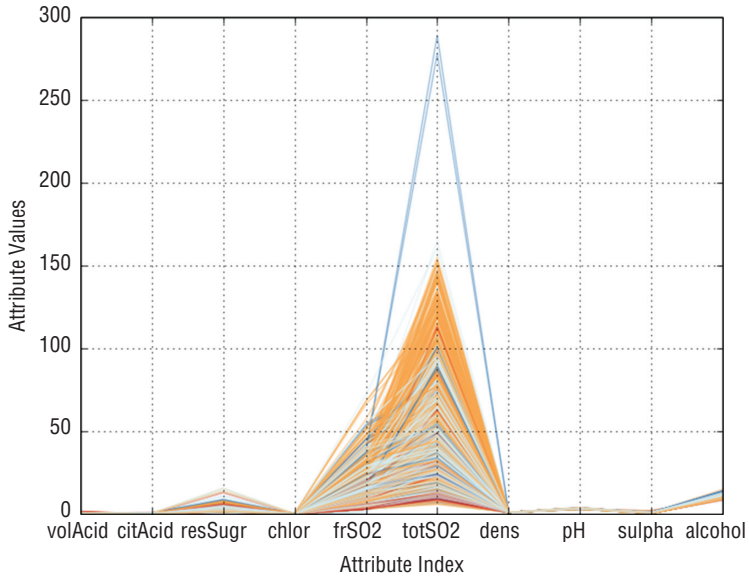

Figure 2-17: Parallel coordinate plot for wine data

Listing 2-14: Producing a Parallel Coordinate Plot for Wine Data—wineParallelPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot
from math import exp

target_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/
wine-quality/winequality-red.csv"
wine = pd.read_csv(target_url,header=0, sep=";")

#generate statistical summaries
summary = wine.describe()
nrows = len(wine.index)
tasteCol = len(summary.columns)
meanTaste = summary.iloc[1,tasteCol - 1]
sdTaste = summary.iloc[2,tasteCol - 1]
nDataCol = len(wine.columns) -1

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = wine.iloc[i,1:nDataCol]
    normTarget = (wine.iloc[i,nDataCol] - meanTaste)/sdTaste
    labelColor = 1.0/(1.0 + exp(-normTarget))
```

continues

continued

```
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()

wineNormalized = wine
ncols = len(wineNormalized.columns)

for i in range(ncols):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]
    wineNormalized.iloc[:,i:(i + 1)] =
    (wineNormalized.iloc[:,i:(i + 1)] - mean) / sd

#Try again with normalized values
for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = wineNormalized.iloc[i,1:nDataCol]
    normTarget = wineNormalized.iloc[i,nDataCol]
    labelColor = 1.0/(1.0 + exp(-normTarget))
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

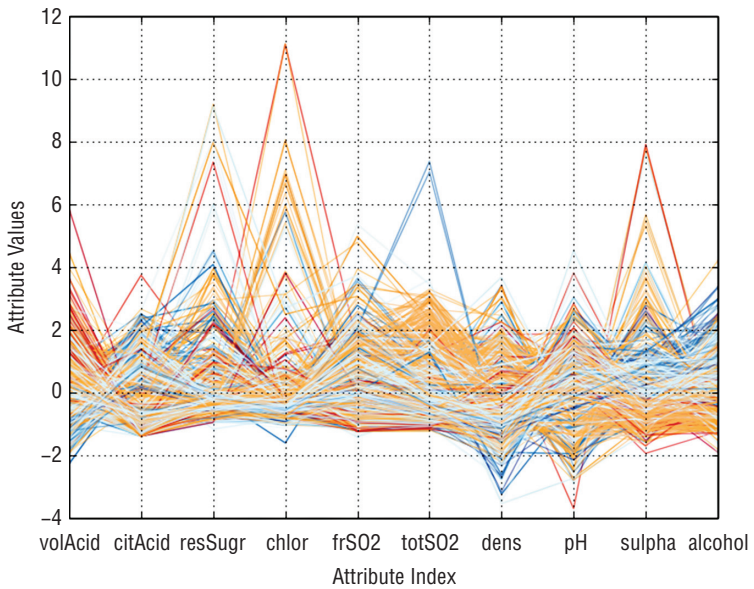

Figure 2-18: Parallel coordinates plot for normalized wine data

The plot of the normalized wine data gives a better simultaneous view of the correlation with the targets along all the coordinate directions. Figure 2-18 shows a clear correlation between several of the attributes. On the far right of the plot, dark blue lines (high taste scores) aggregate at high values of alcohol. On the far left, the dark red lines (low taste scores) aggregate at high values of volatile acidity. Those are the most obviously correlated attributes. The predictive models that you'll see in Chapters 5 and 7 will rank attributes on the basis of their importance in generating predictions. You'll see how these visual observations are supported by the predictive models.

Figure 2-19 shows the heat map of the correlations between attributes and other attributes and between the attributes and the target. In the heat map, hot colors correspond to high levels (the opposite of the color scale used in the parallel coordinates plots). The heat map for the wine data shows relatively high correlation between taste (the last column) and alcohol (the next-to-last column), and very low levels (high correlation but with negative sign) for several of the other attributes, including the first one (volatile acidity).

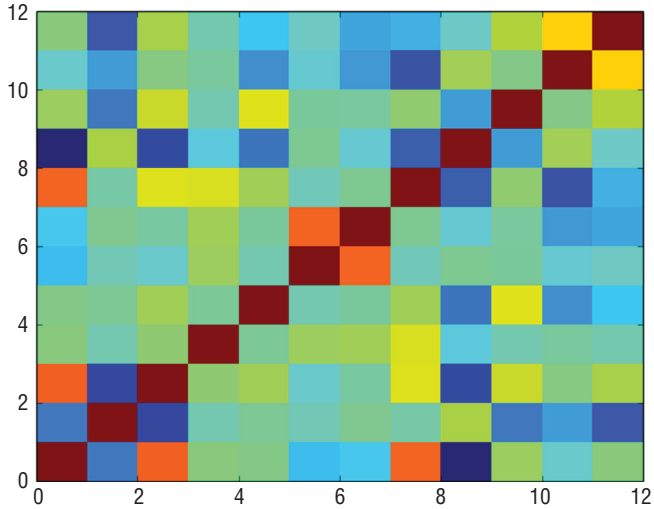

Figure 2-19: Correlation heat map for the wine data

Exploration of the wine data set was accomplished with tools that have already been explained and used. The wine data set shows off what these tools can reveal. Both the parallel coordinates plot and the correlation heat map show that high levels of alcohol go with high taste scores, while high levels of volatile acidity go with low taste scores. You'll see in Chapters 5 and 7 that the variable importance studies that come as part of predictive modeling will echo these findings. The wine data gives a good example of how far data exploration can

take you toward building and qualifying a predictive modeling. The next section will explore data for a multiclass classification problem.

Multiclass Classification Problem: What Type of Glass Is That?

Multiclass classifications are similar to binary classifications, with the difference that there are several possible discrete outcomes instead of just two. Recall that the problem of detecting unexploded mines involved two possible outcomes: that the object being illuminated by the sonar was a rock or that it was a mine. The problem of determining wine taste from measurements of chemical composition had several possible outcomes (taste scores from 3 to 8). But with the wine problem, an order relationship existed among the scores. A wine that had a score of 5 was better than one with a score of 3, but worse than one with a score of 8. With a multiclass problem, no sense of order exists among the outcomes. The glass problem described in this section provides an example of a multiclass problem.

In this section, the glass problem presents chemical compositions of various types of glass. The objective of the problem is to determine the use for the glass. The possible types of glass include glass from building windows, glass from vehicle windows, glass containers, and so on. The motivation for determining the type of glass is forensics. At the scene of an accident or a crime, there are fragments of glass, and determining their origin can help determine who is at fault or who committed the crime. Listing 2-15 shows the code for generating summaries of the glass data set. Figure 2-20 shows the box plot on the normalized data. The box plot shows a fair number of extreme values.

Listing 2-15: Summary of Glass Data Set—glassSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("https://archive.ics.uci.edu/ml/machine-"
              "learning-databases/glass/glass.data")

glass = pd.read_csv(target_url,header=None, prefix="V")
glass.columns = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si',
                 'K', 'Ca', 'Ba', 'Fe', 'Type']

print(glass.head())

#generate statistical summaries
summary = glass.describe()
```

```
print(summary)
ncol1 = len(glass.columns)

glassNormalized = glass.iloc[:, 1:ncol1]
ncol2 = len(glassNormalized.columns)
summary2 = glassNormalized.describe()

for i in range(ncol2):
    mean = summary2.iloc[1, i]
    sd = summary2.iloc[2, i]

glassNormalized.iloc[:,i:(i + 1)] = \
        (glassNormalized.iloc[:,i:(i + 1)] - mean) / sd

array = glassNormalized.values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Output:  [filename - ]
print(glass.head())
```

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0	0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0	0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0	0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0	0	1

```
print(summary) - Abridged
```

	Id	RI	Na	Mg	Al
count	214.000000	214.000000	214.000000	214.000000	214.000000
mean	107.500000	1.518365	13.407850	2.684533	1.444907
std	61.920648	0.003037	0.816604	1.442408	0.499270
min	1.000000	1.511150	10.730000	0.000000	0.290000
25%	54.250000	1.516523	12.907500	2.115000	1.190000
50%	107.500000	1.517680	13.300000	3.480000	1.360000
75%	160.750000	1.519157	13.825000	3.600000	1.630000
max	214.000000	1.533930	17.380000	4.490000	3.500000

	K	Ca	Ba	Fe	Type
count	214.000000	214.000000	214.000000	214.000000	214.000000
mean	0.497056	8.956963	0.175047	0.057009	2.780374
std	0.652192	1.423153	0.497219	0.097439	2.103739
min	0.000000	5.430000	0.000000	0.000000	1.000000
25%	0.122500	8.240000	0.000000	0.000000	1.000000
50%	0.555000	8.600000	0.000000	0.000000	2.000000
75%	0.610000	9.172500	0.000000	0.100000	3.000000
max	6.210000	16.190000	3.150000	0.510000	7.000000

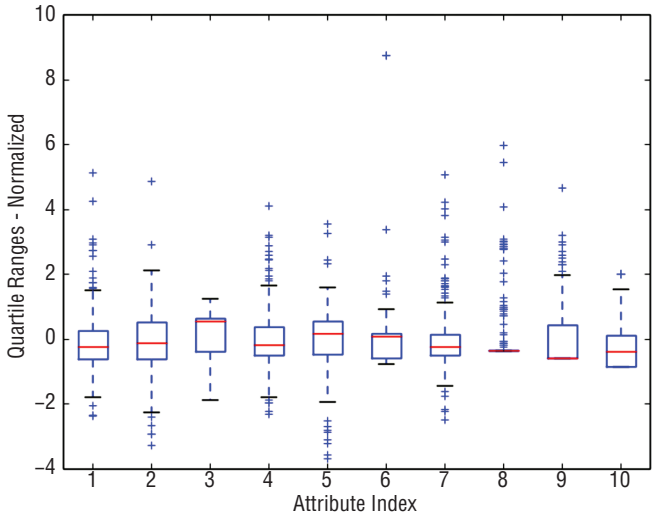

Figure 2-20: Box plot of the glass data

The box plot of the glass data attributes shows a remarkable number of outliers—remarkable at least by comparison to some of the other example problems. The glass data have a couple of elements that may drive the outlier behavior. One is that the problem is a classification problem. There's not necessarily any continuity in relationship between attribute values and class membership—no reason to expect proximity of attribute values across classes. Another unique feature of the glass data is that it is somewhat unbalanced. The number of examples of each class runs from 76 for the most populous class to 9 for the least populous. The average statistics can be dominated by the values for the most populous classes and there's no reason to expect members of other classes to have similar attribute values. The radical behavior can be a good thing for distinguishing classes from one another, but it also means that a method for making predictions has to be able to trace a fairly complicated boundary between the different classes. You'll learn in Chapter 3 that ensemble methods are producing more complicated decision boundaries than penalized linear regression if they are given enough data, and you'll see in Chapters 5 and 7 which family performs better on this data set.

The parallel coordinates plot might shed some more light on the behavior of these data. Figure 2-21 shows the parallel coordinates plot. The data is plotted using discrete colors for each possible output classification. Some of the variables in the plot show fairly distinct paths of color. For example, the dark blue lines group together fairly well and are well separated from the other classes along a number of the attributes. The dark blue lines are at the edges of the data for several attributes—in other words, outliers along those attributes. The light blue lines are less numerous than the dark blue ones and are at the edges for

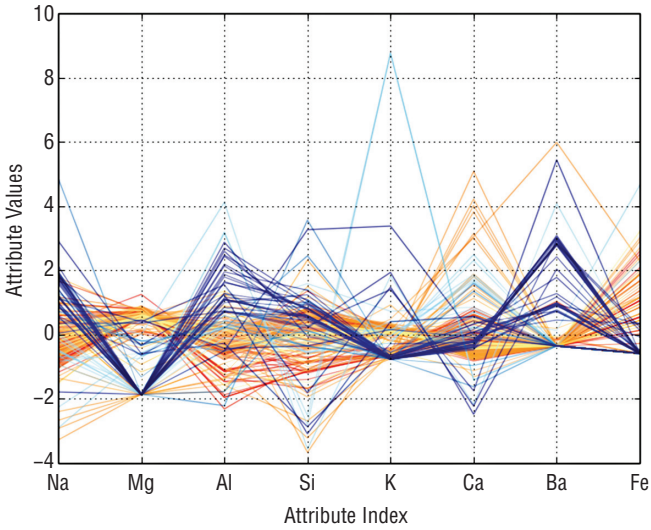

Figure 2-21: Parallel coordinate plot for the glass data

some of the same attributes as dark blue, but not for all of the same attributes. The middle brown lines also group together but toward the mid-range in value.

Listing 2-16: Parallel Coordinate Plot for the Glass Data—glassParallelPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("https://archive.ics.uci.edu/ml/machine-"
              "learning-databases/glass/glass.data")

glass = pd.read_csv(target_url,header=None, prefix="V")
glass.columns = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si',
                 'K', 'Ca', 'Ba', 'Fe', 'Type']

glassNormalized = glass
ncols = len(glassNormalized.columns)
nrows = len(glassNormalized.index)
summary = glassNormalized.describe()
nDataCol = ncols - 1

#normalize except for labels
for i in range(ncols - 1):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]

glassNormalized.iloc[:,i:(i + 1)] = \
```

continues

continued

```
        (glassNormalized.iloc[:,i:(i + 1)] - mean) / sd

#Plot Parallel Coordinate Graph with normalized values
for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = glassNormalized.iloc[i,1:nDataCol]
    labelColor = glassNormalized.iloc[i,nDataCol]/7.0
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

Listing 2-16 shows the code to produce a parallel coordinates plot of the glass data. With the rocks versus mines problem, the lines in the parallel coordinates plot were two-colored to account for the two different label values. In the regression problems (wine taste and abalone age), the labels could take any real value, and the lines in the plots were drawn in a spectrum of different colors. In this multiclass problem, each class gets its own color. There are six discrete colors. The labels run from 1 to 7; there are no 4s. The calculation of the color is similar to the calculation done in the regression problem—divide the numeric label by its maximum value. The resulting lines in the plots take six discrete colors. Figure 2-22 shows the correlation heat map for the glass data. The plot shows mostly low correlation between attributes. That means the attributes are mostly independent of one another, which is a good thing. The targets are not included in the correlation map because the problem has targets that take on one of several discrete values. This robs the correlation heat map of some explanatory power.

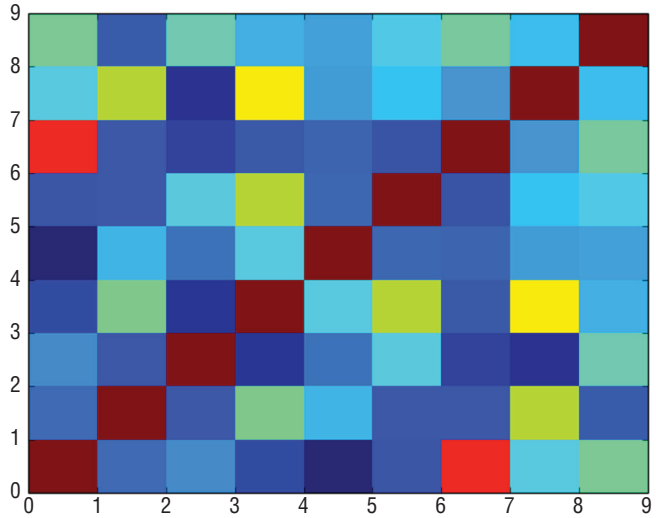

Figure 2-22: Correlation heat map for the glass problem

Exploratory studies for the glass data have revealed a very interesting problem. In particular, the box plots, coupled with the parallel coordinates plot, suggest that a good choice of algorithm might be an ensemble method if there's enough data to fit it. The sets of attributes corresponding to one class or another apparently have a complicated boundary between them. What algorithm will give the best predictive performance remains to be seen. The exploratory methods you have learned have done their job. They have given a good understanding of the tradeoffs for this problem, leading to some guesses about what algorithm will give the best performance.

Summary

This chapter introduced you to several tools for delving into new data sets and coming away with an understanding of how to proceed to building predictive models. The tools began with simply learning the size and shape of the data set and determining the types of attributes and labels. These facts about your data set will help you set your course through preprocesing the data and training predictive models. The chapter also covered several different statistical studies that can help you understand your data set. These included simple descriptive statistics (mean, variance, and quantiles) and second order statistics like correlations between attributes and correlations between attributes and labels. The correlation of attributes and binary labels required some techniques different from real number (regression labels). The chapter also introduced several visualization techniques. One was a Q-Q plot for visualizing outlier behavior in your data. Another was the parallel coordinates plot for visualizing the relationship between attributes and labels. All of these were applied to the problems that will be used in the rest of the book for demonstrating the algorithms covered and for comparing them.

Reference

1. Gorman, R. P., and Sejnowski, T. J. (1988). UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar, +Mines+vs.+Rocks%29. Irvine, CA: University of California, School of Information and Computer Science.