Using the fitted model from Equation 6-11, Table 6-1 provides the probability of a customer churning based on the customer's age and the number of churned contacts. The computed values of y are also provided in the table. Recalling the previous discussion of the logistic function, as the value of y increases, so does the probability of churning.

Table 6-1 *Estimated Churn Probabilities*

Customer	Age (Years)	Churned_Contacts	y	Prob. of Churning
1	50	1	–4.12	0.016
2	50	3	–3.36	0.034
3	50	6	–2.22	0.098
4	30	1	–0.92	0.285
5	30	3	–0.16	0.460
6	30	6	0.98	0.727
7	20	1	0.68	0.664
8	20	3	1.44	0.808
9	20	6	2.58	0.930

Based on the fitted model, there is a 93% chance that a 20-year-old customer who has had six contacts churn will also churn. (See the last row of Table 6-1.) Examining the sign and values of the estimated coefficients in Equation 6-11, it is observed that as the value of *Age* increases, the value of y decreases. Thus, the negative *Age* coefficient indicates that the probability of churning decreases for an older customer. On the other hand, based on the positive sign of the *Churned_Contacts* coefficient, the value of y and subsequently the probability of churning increases as the number of churned contacts increases.

6.2.3 Diagnostics

The churn example illustrates how to interpret a fitted logistic regression model. Using R, this section examines the steps to develop a logistic regression model and evaluate the model's effectiveness. For this example, the *churn_input* data frame is structured as follows:

```
head(churn_input)
```

```
  ID Churned Age Married Cust_years Churned_contacts
1  1       0  61       1          3                1
2  2       0  50       1          3                2
3  3       0  47       1          2                0
4  4       0  50       1          3                3
5  5       0  29       1          1                3
6  6       0  43       1          4                3
```

A *Churned* value of 1 indicates that the customer churned. A *Churned* value of 0 indicates that the customer remained as a subscriber. Out of the 8,000 customer records in this dataset, 1,743 customers (~22%) churned.

```
sum(churn_input$Churned)

[1] 1743
```

Using the Generalized Linear Model function, `glm()`, in R and the specified family/link, a logistic regression model can be applied to the variables in the dataset and examined as follows:

```
Churn_logistic1 <- glm (Churned~Age + Married + Cust_years +
                        Churned_contacts, data=churn_input,
                        family=binomial(link="logit"))
summary(Churn_logistic1)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       3.415201   0.163734  20.858   <2e-16 ***
Age              -0.156643   0.004088 -38.320   <2e-16 ***
Married           0.066432   0.068302   0.973    0.331
Cust_years        0.017857   0.030497   0.586    0.558
Churned_contacts  0.382324   0.027313  13.998   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As in the linear regression case, there are tests to determine if the coefficients are significantly different from zero. Such significant coefficients correspond to small values of `Pr(>|z|)`, which denote the p-value for the hypothesis test to determine if the estimated model parameter is significantly different from zero. Rerunning this analysis without the *Cust_years* variable, which had the largest corresponding p-value, yields the following:

```
Churn_logistic2 <- glm (Churned~Age + Married +  Churned_contacts,
                        data=churn_input, family=binomial(link="logit"))
summary(Churn_logistic2)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       3.472062   0.132107  26.282   <2e-16 ***
Age              -0.156635   0.004088 -38.318   <2e-16 ***
Married           0.066430   0.068299   0.973    0.331
Churned_contacts  0.381909   0.027302  13.988   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Because the p-value for the *Married* coefficient remains quite large, the *Married* variable is dropped from the model. The following R code provides the third and final model, which includes only the *Age* and *Churned_contacts* variables:

```
Churn_logistic3 <- glm (Churned~Age + Churned_contacts,
```

```
                    data=churn_input, family=binomial(link="logit"))
summary(Churn_logistic3)

Call:
glm(formula = Churned ~ Age + Churned_contacts,
               family = binomial(link = "logit"), data = churn_input)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4599  -0.5214  -0.1960  -0.0736   3.3671

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)       3.502716   0.128430   27.27   <2e-16 ***
Age              -0.156551   0.004085  -38.32   <2e-16 ***
Churned_contacts  0.381857   0.027297   13.99   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 8387.3  on 7999  degrees of freedom
Residual deviance: 5359.2  on 7997  degrees of freedom
AIC: 5365.2

Number of Fisher Scoring iterations: 6
```

For this final model, the entire summary output is provided. The output offers several values that can be used to evaluate the fitted model. It should be noted that the model parameter estimates correspond to the values provided in Equation 6-11 that were used to construct Table 6-1.

Deviance and the Pseudo-R^2

In logistic regression, deviance is defined to be $-2 * logL$, where L is the maximized value of the likelihood function that was used to obtain the parameter estimates. In the R output, two deviance values are provided. The ***null deviance*** is the value where the likelihood function is based only on the intercept term ($y = \beta_0$). The ***residual deviance*** is the value where the likelihood function is based on the parameters in the specified logistic model, shown in Equation 6-12.

$$y = \beta_0 + \beta_1 * Age + \beta_2 * Churned\_contacts \tag{6-12}$$

A metric analogous to R^2 in linear regression can be computed as shown in Equation 6-13.

$$\text{pseudo-}R^2 = 1 - \frac{residual\ dev.}{null\ dev.} = \frac{null\ dev. - res.\ dev.}{null\ dev.} \tag{6-13}$$

The pseudo-R^2 is a measure of how well the fitted model explains the data as compared to the default model of no predictor variables and only an intercept term. A $pseudo - R^2$ value near 1 indicates a good fit over the simple null model.

Deviance and the Log-Likelihood Ratio Test

In the *pseudo* $-R^2$ calculation, the –2 multipliers simply divide out. So, it may appear that including such a multiplier does not provide a benefit. However, the multiplier in the deviance definition is based on the log-likelihood test statistic shown in Equation 6-14:

$$T = -2 * log\left(\frac{L_{null}}{L_{alt.}}\right) = -2 * \log(L_{null}) - (-2) * \log(L_{alt.}) \quad (6\text{-}14)$$

where T is approximately Chi-squared distributed (χ^2_k) with

k *degrees of freedom* $(df) = df_{null} - df_{alternate}$

The previous description of the log-likelihood test statistic applies to any estimation using MLE. As can be seen in Equation 6-15, in the logistic regression case,

$$T = \text{null deviance} - \text{residual deviance} \sim \chi^2_{p-1} \quad (6\text{-}15)$$

where p is the number of parameters in the fitted model.

So, in a hypothesis test, a large value of T would indicate that the fitted model is significantly better than the null model that uses only the intercept term.

In the churn example, the log-likelihood ratio statistic would be this:

$T = 8387.3 - 5359.2 = 3028.1$ with 2 degrees of freedom and a corresponding p-value that is essentially zero.

So far, the log-likelihood ratio test discussion has focused on comparing a fitted model to the default model of using only the intercept. However, the log-likelihood ratio test can also compare one fitted model to another. For example, consider the logistic regression model when the categorical variable `Married` is included with `Age` and `Churned_contacts` in the list of input variables. The partial R output for such a model is provided here:

```
summary(Churn_logistic2)
Call:
glm(formula = Churned ~ Age + Married + Churned_contacts,
    family = binomial(link = "logit"),
    data = churn_input)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)      3.472062   0.132107  26.282   <2e-16 ***
Age             -0.156635   0.004088 -38.318   <2e-16 ***
Married          0.066430   0.068299   0.973    0.331
Churned_contacts 0.381909   0.027302  13.988   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 8387.3  on 7999  degrees of freedom
Residual deviance: 5358.3  on 7996  degrees of freedom
```

The residual deviances from each model can be used to perform a hypothesis test where $H_0 : \beta_{Married} = 0$ against $H_A : \beta_{Married} \neq 0$ using the base model that includes the `Age` and `Churned_contacts` variables. The test statistic follows:

$$T = 5359.2 - 5358.3 = 0.9 \text{ with } 7997 - 7996 = 1 \text{ degree of freedom}$$

Using R, the corresponding p-value is calculated as follows:

```
pchisq(.9 , 1, lower=FALSE)
```

```
[1] 0.3427817
```

Thus, at a 66% or higher confidence level, the null hypothesis, $H_0 : \beta_{Married} = 0$, would not be rejected. Thus, it seems reasonable to exclude the variable `Married` from the logistic regression model.

In general, this log-likelihood ratio test is particularly useful for forward and backward step-wise methods to add variables to or remove them from the proposed logistic regression model.

Receiver Operating Characteristic (ROC) Curve

Logistic regression is often used as a classifier to assign class labels to a person, item, or transaction based on the predicted probability provided by the model. In the Churn example, a customer can be classified with the label called `Churn` if the logistic model predicts a high probability that the customer will churn. Otherwise, a `Remain` label is assigned to the customer. Commonly, 0.5 is used as the default probability threshold to distinguish between any two class labels. However, any threshold value can be used depending on the preference to avoid false positives (for example, to predict `Churn` when actually the customer will `Remain`) or false negatives (for example, to predict `Remain` when the customer will actually `Churn`).

In general, for two class labels, C and ¬C, where "¬C" denotes "not C," some working definitions and formulas follow:

- **True Positive:** predict C, when actually C
- **True Negative:** predict ¬C, when actually ¬C
- **False Positive:** predict C, when actually ¬C
- **False Negative:** predict ¬C, when actually C

$$\text{False Positive Rate (FPR)} = \frac{\#\ of\ false\ positives}{\#\ of\ negatives} \tag{6-16}$$

$$\text{True Positive : Rate (TPR)} = \frac{\#\ of\ true\ positives}{\#\ of\ positives} \tag{6-17}$$

The plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) is known as the ***Receiver Operating Characteristic (ROC)*** curve. Using the `ROCR` package, the following R commands generate the ROC curve for the Churn example:

```
library(ROCR)

pred = predict(Churn_logistic3, type="response")
```

```
predObj = prediction(pred, churn_input$Churned )

rocObj = performance(predObj, measure="tpr", x.measure="fpr")
aucObj = performance(predObj, measure="auc")

plot(rocObj, main = paste("Area under the curve:",
                          round(aucObj@y.values[[1]] ,4)))
```

The usefulness of this plot in Figure 6-15 is that the preferred outcome of a classifier is to have a low FPR and a high TPR. So, when moving from left to right on the FPR axis, a good model/classifier has the TPR rapidly approach values near 1, with only a small change in FPR. The closer the ROC curve tracks along the vertical axis and approaches the upper-left hand of the plot, near the point (0,1), the better the model/classifier performs. Thus, a useful metric is to compute the area under the ROC curve (AUC). By examining the axes, it can be seen that the theoretical maximum for the area is 1.

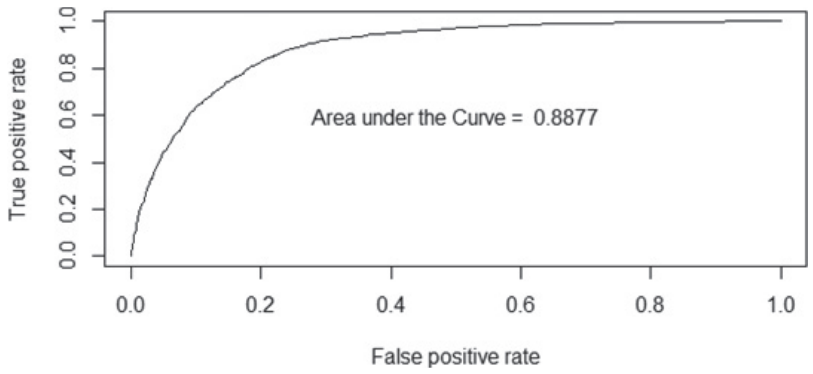

FIGURE 6-15 *ROC curve for the churn example*

To illustrate how the FPR and TPR values are dependent on the threshold value used for the classifier, the plot in Figure 6-16 was constructed using the following R code:

```
# extract the alpha(threshold), FPR, and TPR values from rocObj
alpha <- round(as.numeric(unlist(rocObj@alpha.values)),4)
fpr <- round(as.numeric(unlist(rocObj@x.values)),4)
tpr <- round(as.numeric(unlist(rocObj@y.values)),4)

# adjust margins and plot TPR and FPR
par(mar = c(5,5,2,5))
plot(alpha,tpr, xlab="Threshold", xlim=c(0,1),
                ylab="True positive rate", type="l")
par(new="True")
plot(alpha,fpr, xlab="", ylab="", axes=F, xlim=c(0,1), type="l" )
axis(side=4)
mtext(side=4, line=3, "False positive rate")
text(0.18,0.18,"FPR")
text(0.58,0.58,"TPR")
```

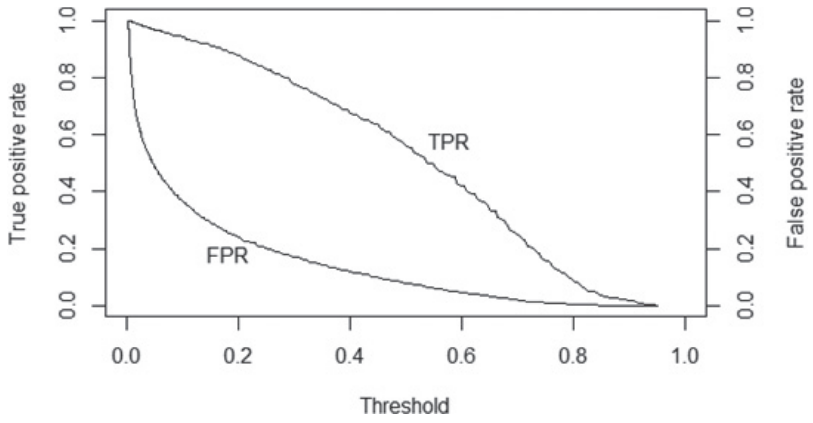

FIGURE 6-16 *The effect of the threshold value in the churn example*

For a threshold value of 0, every item is classified as a positive outcome. Thus, the TPR value is 1. However, all the negatives are also classified as a positive, and the FPR value is also 1. As the threshold value increases, more and more negative class labels are assigned. Thus, the FPR and TPR values decrease. When the threshold reaches 1, no positive labels are assigned, and the FPR and TPR values are both 0.

For the purposes of a classifier, a commonly used threshold value is 0.5. A positive label is assigned for any probability of 0.5 or greater. Otherwise, a negative label is assigned. As the following R code illustrates, in the analysis of the Churn dataset, the 0.5 threshold corresponds to a TPR value of 0.56 and a FPR value of 0.08.

```
i <- which(round(alpha,2) == .5)
paste("Threshold=" , (alpha[i]) , " TPR=" , tpr[i] , " FPR=" , fpr[i])

[1] "Threshold= 0.5004  TPR= 0.5571  FPR= 0.0793"
```

Thus, 56% of customers who will churn are properly classified with the *Churn* label, and 8% of the customers who will remain as customers are improperly labeled as *Churn*. If identifying only 56% of the churners is not acceptable, then the threshold could be lowered. For example, suppose it was decided to classify with a *Churn* label any customer with a probability of churning greater than 0.15. Then the following R code indicates that the corresponding TPR and FPR values are 0.91 and 0.29, respectively. Thus, 91% of the customers who will churn are properly identified, but at a cost of misclassifying 29% of the customers who will remain.

```
i <- which(round(alpha,2) == .15)
paste("Threshold=" , (alpha[i]) , " TPR=" , tpr[i] , " FPR=" , fpr[i])

[1] "Threshold= 0.1543  TPR= 0.9116  FPR= 0.2869"
[2] "Threshold= 0.1518  TPR= 0.9122  FPR= 0.2875"
[3] "Threshold= 0.1479  TPR= 0.9145  FPR= 0.2942"
[4] "Threshold= 0.1455  TPR= 0.9174  FPR= 0.2981"
```

The ROC curve is useful for evaluating other classifiers and will be utilized again in Chapter 7, "Advanced Analytical Theory and Methods: Classification."

Histogram of the Probabilities

It can be useful to visualize the observed responses against the estimated probabilities provided by the logistic regression. Figure 6-17 provides overlaying histograms for the customers who churned and for the customers who remained as customers. With a proper fitting logistic model, the customers who remained tend to have a low probability of churning. Conversely, the customers who churned have a high probability of churning again. This histogram plot helps visualize the number of items to be properly classified or misclassified. In the Churn example, an ideal histogram plot would have the remaining customers grouped at the left side of the plot, the customers who churned at the right side of the plot, and no overlap of these two groups.

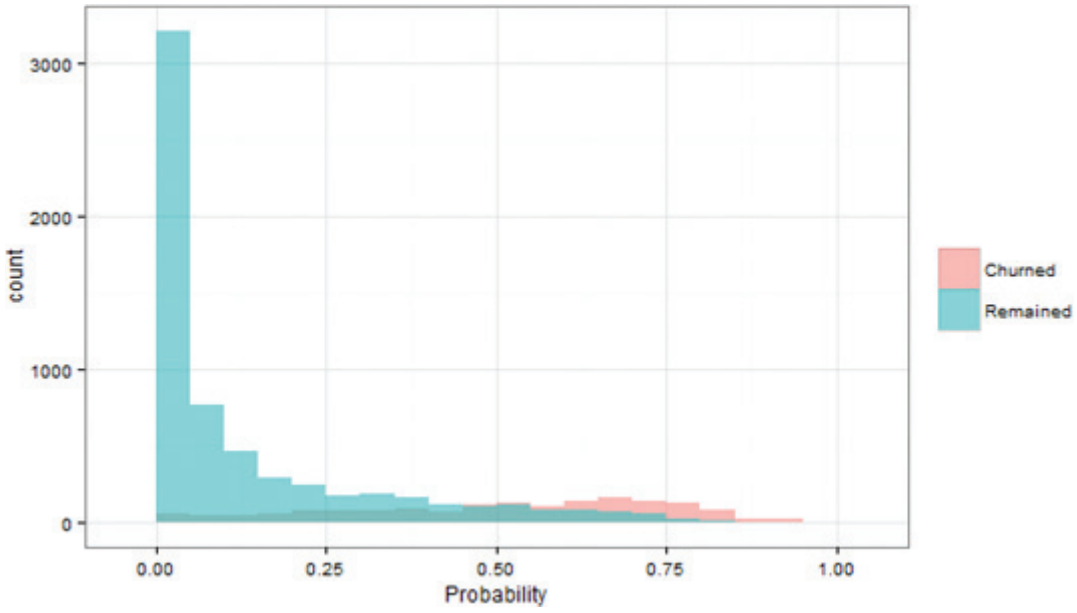

FIGURE 6-17 *Customer counts versus estimated churn probability*

6.3 Reasons to Choose and Cautions

Linear regression is suitable when the input variables are continuous or discrete, including categorical data types, but the outcome variable is continuous. If the outcome variable is categorical, logistic regression is a better choice.

Both models assume a linear additive function of the input variables. If such an assumption does not hold true, both regression techniques perform poorly. Furthermore, in linear regression, the assumption of normally distributed error terms with a constant variance is important for many of the statistical inferences that can be considered. If the various assumptions do not appear to hold, the appropriate transformations need to be applied to the data.

Although a collection of input variables may be a good predictor for the outcome variable, the analyst should not infer that the input variables directly cause an outcome. For example, it may be identified that those individuals who have regular dentist visits may have a reduced risk of heart attacks. However, simply sending someone to the dentist almost certainly has no effect on that person's chance of having a heart attack. It is possible that regular dentist visits may indicate a person's overall health and dietary choices, which may have a more direct impact on a person's health. This example illustrates the commonly known expression, "Correlation does not imply causation."

Use caution when applying an already fitted model to data that falls outside the dataset used to train the model. The linear relationship in a regression model may no longer hold at values outside the training dataset. For example, if income was an input variable and the values of income ranged from $35,000 to $90,000, applying the model to incomes well outside those incomes could result in inaccurate estimates and predictions.

The income regression example in Section 6.1.2 mentioned the possibility of using categorical variables to represent the 50 U.S. states. In a linear regression model, the state of residence would provide a simple additive term to the income model but no other impact on the coefficients of the other input variables, such as `Age` and `Education`. However, if state does influence the other variables' impact to the income model, an alternative approach would be to build 50 separate linear regression models: one model for each state. Such an approach is an example of the options and decisions that the data scientist must be willing to consider.

If several of the input variables are highly correlated to each other, the condition is known as ***multicollinearity***. Multicollinearity can often lead to coefficient estimates that are relatively large in absolute magnitude and may be of inappropriate direction (negative or positive sign). When possible, the majority of these correlated variables should be removed from the model or replaced by a new variable that is a function of the correlated variables. For example, in a medical application of regression, `height` and `weight` may be considered important input variables, but these variables tend to be correlated. In this case, it may be useful to use the Body Mass Index (BMI), which is a function of a person's height and weight.

$$BMI = \frac{weight}{height^2} \quad \text{where weight is in kilograms and height is in meters}$$

However, in some cases it may be necessary to use the correlated variables. The next section provides some techniques to address highly correlated variables.

6.4 Additional Regression Models

In the case of multicollinearity, it may make sense to place some restrictions on the magnitudes of the estimated coefficients. ***Ridge regression***, which applies a penalty based on the size of the coefficients, is one technique that can be applied. In fitting a linear regression model, the objective is to find the values of the coefficients that minimize the sum of the residuals squared. In ridge regression, a penalty term proportional to the sum of the squares of the coefficients is added to the sum of the residuals squared. ***Lasso regression*** is a related modeling technique in which the penalty is proportional to the sum of the absolute values of the coefficients.

Only binary outcome variables were examined in the use of logistic regression. If the outcome variable can assume more than two states, ***multinomial logistic regression*** can be used.

Summary

This chapter discussed the use of linear regression and logistic regression to model historical data and to predict future outcomes. Using R, examples of each regression technique were presented. Several diagnostics to evaluate the models and the underlying assumptions were covered.

Although regression analysis is relatively straightforward to perform using many existing software packages, considerable care must be taken in performing and interpreting a regression analysis. This chapter highlighted that in a regression analysis, the data scientist needs to do the following:

- Determine the best input variables and their relationship to the outcome variable .
- Understand the underlying assumptions and their impact on the modeling results.
- Transform the variables, as appropriate, to achieve adherence to the model assumptions.
- Decide whether building one comprehensive model is the best choice or consider building many models on partitions of the data.

Exercises

1. In the Income linear regression example, consider the distribution of the outcome variable `Income`. `Income` values tend to be highly skewed to the right (distribution of value has a large tail to the right). Does such a non-normally distributed outcome variable violate the general assumption of a linear regression model? Provide supporting arguments.
2. In the use of a categorical variable with n possible values, explain the following:
   a. Why only n – 1 binary variables are necessary
   b. Why using n variables would be problematic
3. In the example of using Wyoming as the reference case, discuss the effect on the estimated model parameters, including the intercept, if another state was selected as the reference case.
4. Describe how logistic regression can be used as a classifier.
5. Discuss how the ROC curve can be used to determine an appropriate threshold value for a classifier.
6. If the probability of an event occurring is 0.4, then
   a. What is the odds ratio?
   b. What is the log odds ratio?
7. If $b_3 = -.5$ is an estimated coefficient in a linear regression model, what is the effect on the odds ratio for every one unit increase in the value of x_3?

7

Advanced Analytical Theory and Methods: Classification

Key Concepts

Classification learning
Naïve Bayes
Decision tree
ROC curve
Confusion matrix

In addition to analytical methods such as clustering (Chapter 4, "Advanced Analytical Theory and Methods: Clustering"), association rule learning Chapter 5, "Advanced Analytical Theory and Methods: Association Rules", and modeling techniques like regression (Chapter 6, "Advanced Analytical Theory and Methods: Regression"), classification is another fundamental learning method that appears in applications related to data mining. In classification learning, a classifier is presented with a set of examples that are already classified and, from these examples, the classifier learns to assign unseen examples. In other words, the primary task performed by classifiers is to assign class labels to new observations. Logistic regression from the previous chapter is one of the popular classification methods. The set of labels for classifiers is predetermined, unlike in clustering, which discovers the structure without a training set and allows the data scientist optionally to create and assign labels to the clusters.

Most classification methods are supervised, in that they start with a training set of prelabeled observations to learn how likely the attributes of these observations may contribute to the classification of future unlabeled observations. For example, existing marketing, sales, and customer demographic data can be used to develop a classifier to assign a "purchase" or "no purchase" label to potential future customers.

Classification is widely used for prediction purposes. For example, by building a classifier on the transcripts of United States Congressional floor debates, it can be determined whether the speeches represent support or opposition to proposed legislation [1]. Classification can help health care professionals diagnose heart disease patients [2]. Based on an e-mail's content, e-mail providers also use classification to decide whether the incoming e-mail messages are spam [3].

This chapter mainly focuses on two fundamental classification methods: ***decision trees*** and ***naïve Bayes***.

7.1 Decision Trees

A ***decision tree*** (also called ***prediction tree***) uses a tree structure to specify sequences of decisions and consequences. Given input $X = \{x_1, x_2, \ldots x_n\}$, the goal is to predict a response or output variable Y. Each member of the set $\{x_1, x_2, \ldots x_n\}$ is called an ***input variable***. The prediction can be achieved by constructing a decision tree with test points and branches. At each test point, a decision is made to pick a specific branch and traverse down the tree. Eventually, a final point is reached, and a prediction can be made. Each test point in a decision tree involves testing a particular input variable (or attribute), and each branch represents the decision being made. Due to its flexibility and easy visualization, decision trees are commonly deployed in data mining applications for classification purposes.

The input values of a decision tree can be categorical or continuous. A decision tree employs a structure of test points (called ***nodes***) and branches, which represent the decision being made. A node without further branches is called a ***leaf node***. The leaf nodes return class labels and, in some implementations, they return the probability scores. A decision tree can be converted into a set of decision rules. In the following example rule, `income` and `mortgage_amount` are input variables, and the response is the output variable `default` with a probability score.

```
IF income < $50,000 AND mortgage_amount > $100K
THEN default = True WITH PROBABILITY 75%
```

Decision trees have two varieties: ***classification trees*** and ***regression trees***. Classification trees usually apply to output variables that are categorical—often binary—in nature, such as yes or no, purchase or not purchase, and so on. Regression trees, on the other hand, can apply to output variables that are numeric or continuous, such as the predicted price of a consumer good or the likelihood a subscription will be purchased.

Decision trees can be applied to a variety of situations. They can be easily represented in a visual way, and the corresponding decision rules are quite straightforward. Additionally, because the result is a series of logical `if-then` statements, there is no underlying assumption of a linear (or nonlinear) relationship between the input variables and the response variable.

7.1.1 Overview of a Decision Tree

Figure 7-1 shows an example of using a decision tree to predict whether customers will buy a product. The term ***branch*** refers to the outcome of a decision and is visualized as a line connecting two nodes. If a decision is numerical, the "greater than" branch is usually placed on the right, and the "less than" branch is placed on the left. Depending on the nature of the variable, one of the branches may need to include an "equal to" component.

Internal nodes are the decision or test points. Each internal node refers to an input variable or an attribute. The top internal node is called the ***root***. The decision tree in Figure 7-1 is a binary tree in that each internal node has no more than two branches. The branching of a node is referred to as a ***split***.

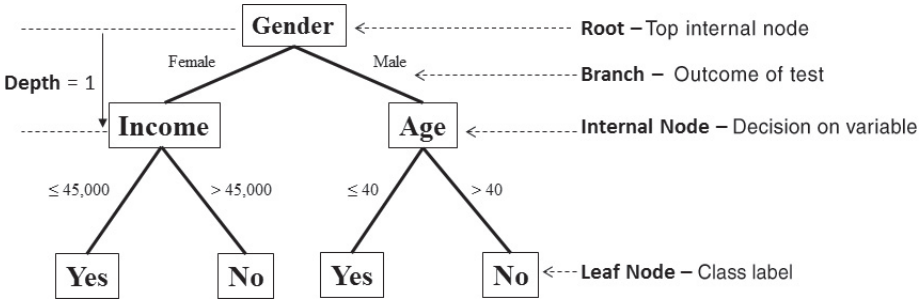

FIGURE 7-1 *Example of a decision tree*

Sometimes decision trees may have more than two branches stemming from a node. For example, if an input variable `Weather` is categorical and has three choices—`Sunny`, `Rainy`, and `Snowy`—the corresponding node `Weather` in the decision tree may have three branches labeled as `Sunny`, `Rainy`, and `Snowy`, respectively.

The ***depth*** of a node is the minimum number of steps required to reach the node from the root. In Figure 7-1 for example, nodes `Income` and `Age` have a depth of one, and the four nodes on the bottom of the tree have a depth of two.

Leaf nodes are at the end of the last branches on the tree. They represent class labels—the outcome of all the prior decisions. The path from the root to a leaf node contains a series of decisions made at various internal nodes.

In Figure 7-1, the root node splits into two branches with a `Gender` test. The right branch contains all those records with the variable `Gender` equal to `Male`, and the left branch contains all those records with the variable `Gender` equal to `Female` to create the depth 1 internal nodes. Each internal node effectively acts as the root of a subtree, and a best test for each node is determined independently of the other internal nodes. The left-hand side (LHS) internal node splits on a question based on the `Income` variable to create leaf nodes at depth 2, whereas the right-hand side (RHS) splits on a question on the `Age` variable.

The decision tree in Figure 7-1 shows that females with income less than or equal to $45,000 and males 40 years old or younger are classified as people who would purchase the product. In traversing this tree, age does not matter for females, and income does not matter for males.

Decision trees are widely used in practice. For example, to classify animals, questions (like cold-blooded or warm-blooded, mammal or not mammal) are answered to arrive at a certain classification. Another example is a checklist of symptoms during a doctor's evaluation of a patient. The artificial intelligence engine of a video game commonly uses decision trees to control the autonomous actions of a character in response to various scenarios. Retailers can use decision trees to segment customers or predict response rates to marketing and promotions. Financial institutions can use decision trees to help decide if a loan application should be approved or denied. In the case of loan approval, computers can use the logical `if-then` statements to predict whether the customer will default on the loan. For customers with a clear (strong) outcome, no human interaction is required; for observations that may not generate a clear response, a human is needed for the decision.

By limiting the number of splits, a short tree can be created. Short trees are often used as ***components*** (also called ***weak learners*** or ***base learners***) in ***ensemble methods***. Ensemble methods use multiple predictive models to vote, and decisions can be made based on the combination of the votes. Some popular ensemble methods include random forest [4], bagging, and boosting [5]. Section 7.4 discusses these ensemble methods more.

The simplest short tree is called a ***decision stump***, which is a decision tree with the root immediately connected to the leaf nodes. A decision stump makes a prediction based on the value of just a single input variable. Figure 7-2 shows a decision stump to classify two species of an iris flower based on the petal width. The figure shows that, if the petal width is smaller than 1.75 centimeters, it's Iris versicolor; otherwise, it's Iris virginica.

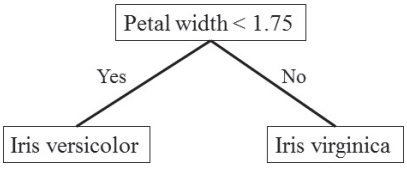

FIGURE 7-2 *Example of a decision stump*

To illustrate how a decision tree works, consider the case of a bank that wants to market its term deposit products (such as Certificates of Deposit) to the appropriate customers. Given the demographics of clients and their reactions to previous campaign phone calls, the bank's goal is to predict which clients would subscribe to a term deposit. The dataset used here is based on the original dataset collected from a Portuguese bank on directed marketing campaigns as stated in the work by Moro et al. [6]. Figure 7-3 shows a subset of the modified bank marketing dataset. This dataset includes 2,000 instances randomly drawn

from the original dataset, and each instance corresponds to a customer. To make the example simple, the subset only keeps the following categorical variables: (1) *job*, (2) *marital* status, (3) *education* level, (4) if the credit is in *default*, (5) if there is a *housing* loan, (6) if the customer currently has a personal *loan*, (7) *contact* type, (8) result of the previous marketing campaign contact (*poutcome*), and finally (9) if the client actually *subscribed* to the term deposit. Attributes (1) through (8) are input variables, and (9) is considered the outcome. The outcome *subscribed* is either `yes` (meaning the customer will subscribe to the term deposit) or `no` (meaning the customer won't subscribe). All the variables listed earlier are categorical.

	job	marital	education	default	housing	loan	contact	poutcome	subscribed
1	management	single	tertiary	no	yes	no	cellular	unknown	no
2	entrepreneur	married	tertiary	no	yes	yes	cellular	unknown	no
3	services	divorced	secondary	no	no	no	cellular	unknown	yes
4	management	married	tertiary	no	yes	no	cellular	unknown	no
5	management	married	secondary	no	yes	no	unknown	unknown	no
6	management	single	tertiary	no	yes	no	unknown	unknown	no
7	entrepreneur	married	tertiary	no	yes	no	cellular	failure	yes
8	admin.	married	secondary	no	no	no	cellular	unknown	no
9	blue-collar	married	secondary	no	yes	no	cellular	other	no
10	management	married	tertiary	yes	no	no	cellular	unknown	no
11	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
12	management	divorced	secondary	no	no	no	unknown	unknown	no
13	blue-collar	married	secondary	no	yes	no	cellular	unknown	no
14	retired	married	secondary	no	no	no	cellular	unknown	no
15	management	single	tertiary	no	yes	no	cellular	unknown	no
16	retired	married	secondary	yes	yes	no	cellular	unknown	no
17	unemployed	married	secondary	no	yes	no	telephone	unknown	no
18	management	divorced	tertiary	no	yes	no	cellular	unknown	no
19	management	married	tertiary	no	yes	no	cellular	unknown	no
20	blue-collar	married	secondary	no	yes	no	unknown	unknown	no
21	management	divorced	tertiary	no	yes	yes	cellular	failure	yes
22	blue-collar	divorced	secondary	no	yes	no	cellular	failure	no
23	blue-collar	single	secondary	no	yes	no	cellular	failure	no
24	admin.	single	secondary	no	no	no	unknown	unknown	no
25	blue-collar	married	secondary	no	yes	no	cellular	failure	no
26	blue-collar	single	secondary	no	yes	no	unknown	unknown	no
27	housemaid	married	secondary	no	no	no	cellular	unknown	no
28	technician	married	tertiary	no	no	no	cellular	unknown	no

FIGURE 7-3 *A subset of the bank marketing dataset*

A summary of the dataset shows the following statistics. For ease of display, the summary only includes the top six most frequently occurring values for each attribute. The rest are displayed as `(Other)`.

```
         job          marital         education    default
blue-collar:435   divorced: 228   primary  : 335   no :1961
management :423   married :1201   secondary:1010   yes:  39
technician :339   single  : 571   tertiary : 564
admin.     :235                   unknown  :  91
services   :168
retired    : 92
(Other)    :308
```

```
housing      loan           contact         month        poutcome
no : 916   no :1717   cellular :1287   may    :581   failure: 210
yes:1084   yes: 283   telephone: 136   jul    :340   other  :  79
                      unknown  : 577   aug    :278   success:  58
                                       jun    :232   unknown:1653
                                       nov    :183
                                       apr    :118
                                       (Other):268

subscribed
no :1789
yes: 211
```

Attribute *job* includes the following values.

```
     admin.    blue-collar   entrepreneur      housemaid
        235            435             70             63
 management        retired  self-employed       services
        423             92             69            168
    student     technician     unemployed        unknown
         36            339             60             10
```

Figure 7-4 shows a decision tree built over the bank marketing dataset. The root of the tree shows that the overall fraction of the clients who have not subscribed to the term deposit is 1,789 out of the total population of 2,000.

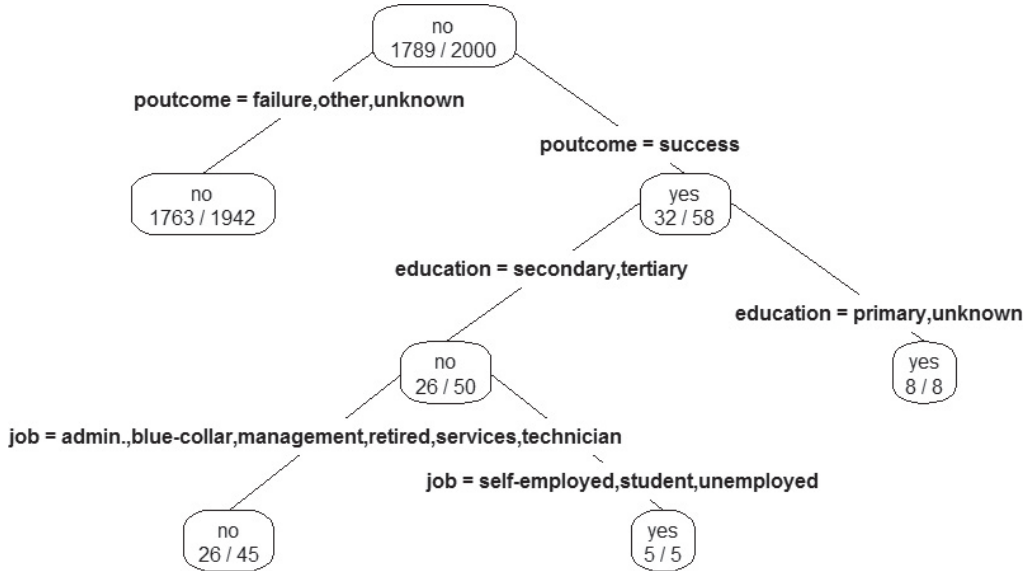

FIGURE 7-4 *Using a decision tree to predict if a client will subscribe to a term deposit*

At each split, the decision tree algorithm picks the most informative attribute out of the remaining attributes. The extent to which an attribute is informative is determined by measures such as entropy and information gain, as detailed in Section 7.1.2.

At the first split, the decision tree algorithm chooses the `poutcome` attribute. There are two nodes at depth=1. The left node is a leaf node representing a group for which the outcome of the previous marketing campaign contact is a `failure`, `other`, or `unknown`. For this group, 1,763 out of 1,942 clients have not subscribed to the term deposit.

The right node represents the *rest* of the population, for which the outcome of the previous marketing campaign contact is a `success`. For the population of this node, 32 out of 58 clients have subscribed to the term deposit.

This node further splits into two nodes based on the education level. If the `education` level is either `secondary` or `tertiary`, then 26 out of 50 of the clients have not subscribed to the term deposit. If the education level is `primary` or `unknown`, then 8 out of 8 times the clients have subscribed.

The left node at depth 2 further splits based on attribute `job`. If the occupation is `admin`, `blue collar`, `management`, `retired`, `services`, or `technician`, then 26 out of 45 clients have not subscribed. If the occupation is `self-employed`, `student`, or `unemployed`, then 5 out of 5 times the clients have subscribed.

7.1.2 The General Algorithm

In general, the objective of a decision tree algorithm is to construct a tree T from a training set S. If all the records in S belong to some class C (*subscribed* = yes, for example), or if S is sufficiently pure (greater than a preset threshold), then that node is considered a leaf node and assigned the label C. The ***purity*** of a node is defined as its probability of the corresponding class. For example, in Figure 7-4, the root $\mathrm{P}(\textit{subscribed} = \text{yes}) = 1 - \frac{1789}{2000} = 10.55\%$; therefore, the root is only 10.55% pure on the *subscribed* = yes class. Conversely, it is 89.45% pure on the *subscribed* = no class.

In contrast, if not all the records in S belong to class C or if S is not sufficiently pure, the algorithm selects the next most informative attribute A (duration, marital, and so on) and partitions S according to A's values. The algorithm constructs subtrees $T_1, T_2 \ldots$ for the subsets of S recursively until one of the following criteria is met:

- All the leaf nodes in the tree satisfy the minimum purity threshold.
- The tree cannot be further split with the preset minimum purity threshold.
- Any other stopping criterion is satisfied (such as the maximum depth of the tree).

The first step in constructing a decision tree is to choose the most informative attribute. A common way to identify the most informative attribute is to use entropy-based methods, which are used by decision tree learning algorithms such as ID3 (or Iterative Dichotomiser 3) [7] and C4.5 [8]. The entropy methods select the most informative attribute based on two basic measures:

- ***Entropy,*** which measures the *impurity* of an attribute
- ***Information gain,*** which measures the *purity* of an attribute

Given a class X and its label $x \in X$, let $P(x)$ be the probability of x. H_x, the entropy of X, is defined as shown in Equation 7-1.

$$H_X = -\sum_{\forall x \in X} P(x) \log_2 P(x) \tag{7-1}$$

Equation 7-1 shows that entropy H_X becomes 0 when all $P(x)$ is 0 or 1. For a binary classification (`true` or `false`), H_X is zero if $P(x)$ the probability of each label x is either zero or one. On the other hand, H_X achieves the maximum entropy when all the class labels are equally probable. For a binary classification, $H_X = 1$ if the probability of all class labels is 50/50. The maximum entropy increases as the number of possible outcomes increases.

As an example of a binary random variable, consider tossing a coin with known, not necessarily fair, probabilities of coming up heads or tails. The corresponding entropy graph is shown in Figure 7-5. Let $x = 1$ represent heads and $x = 0$ represent tails. The entropy of the unknown result of the next toss is maximized when the coin is fair. That is, when heads and tails have equal probability $P(x = 1) = P(x = 0) = 0.5$, entropy $H_X = -(0.5 \times \log_2 0.5 + 0.5 \times \log_2 0.5) = 1$. On the other hand, if the coin is not fair, the probabilities of heads and tails would not be equal and there would be less uncertainty. As an extreme case, when the probability of tossing a head is equal to 0 or 1, the entropy is minimized to 0. Therefore, the entropy for a completely pure variable is 0 and is 1 for a set with equal occurrences for both the classes (head and tail, or yes and no).

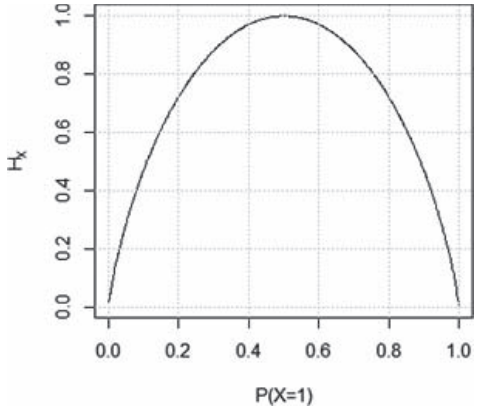

FIGURE 7-5 *Entropy of coin flips, where X=1 represents heads*

For the bank marketing scenario previously presented, the output variable is `subscribed`. The base entropy is defined as entropy of the output variable, that is $H_{subscribed}$. As seen previously, $P(subscribed = \text{yes}) = 0.1055$ and $P(subscribed = \text{no}) = 0.8945$. According to Equation 7-1, the base entropy $H_{subscribed} = -0.1055 \cdot \log_2 0.1055 - 0.8945 \cdot \log_2 0.8945 \approx 0.4862$.

The next step is to identify the conditional entropy for each attribute. Given an attribute X, its value x, its outcome Y, and its value y, conditional entropy $H_{Y|X}$ is the remaining entropy of Y given X, formally defined as shown in Equation 7-2.

$$\begin{aligned} H_{Y|X} &= \sum_{x} P(x) H(Y|X=x) \\ &= -\sum_{\forall x \in x} P(x) \sum_{\forall y \in Y} P(y|x) \log_2 P(y|x) \end{aligned} \tag{7-2}$$

Consider the banking marketing scenario, if the attribute `contact` is chosen, $X = \{$`cellular`, `telephone`, `unknown`$\}$. The conditional entropy of `contact` considers all three values.

Table 7-1 lists the probabilities related to the `contact` attribute. The top row of the table displays the probabilities of each value of the attribute. The next two rows contain the probabilities of the class labels conditioned on the `contact`.

TABLE 7-1 *Conditional Entropy Example*

	Cellular	Telephone	Unknown
P(contact)	0.6435	0.0680	0.2885
P(subscribed=yes | contact)	0.1399	0.0809	0.0347
P(subscribed=no | contact)	0.8601	0.9192	0.9653

The conditional entropy of the `contact` attribute is computed as shown here.

$$\begin{aligned} H_{subscribed|contact} = -\Big[&0.6435 \cdot \left(0.1399 \cdot \log_2 0.1399 + 0.8601 \cdot \log_2 0.8601\right) \\ &+ 0.0680 \cdot \left(0.0809 \cdot \log_2 0.0809 + 0.9192 \cdot \log_2 0.9192\right) \\ &+ 0.2885 \cdot \left(0.0347 \cdot \log_2 0.0347 + 0.9653 \cdot \log_2 0.9653\right)\Big] \\ &= 0.4661 \end{aligned}$$

Computation inside the parentheses is on the entropy of the class labels within a single `contact` value. Note that the conditional entropy is always less than or equal to the base entropy—that is, $H_{subscribed|marital} \leq H_{subscribed}$. The conditional entropy is smaller than the base entropy when the attribute and the outcome are correlated. In the worst case, when the attribute is uncorrelated with the outcome, the conditional entropy equals the base entropy.

The information gain of an attribute `A` is defined as the difference between the base entropy and the conditional entropy of the attribute, as shown in Equation 7-3.

$$InfoGain_A = H_S - H_{S|A} \tag{7-3}$$

In the bank marketing example, the information gain of the `contact` attribute is shown in Equation 7-4.

$$\begin{aligned} InfoGain_{contact} &= H_{subscribed} - H_{contact|subscribed} \\ &= 0.4862 - 0.4661 = 0.0201 \end{aligned} \tag{7-4}$$

Information gain compares the degree of purity of the parent node before a split with the degree of purity of the child node after a split. At each split, an attribute with the greatest information gain is considered the most informative attribute. Information gain indicates the purity of an attribute.

The result of information gain for all the input variables is shown in Table 7-2. Attribute `poutcome` has the most information gain and is the most informative variable. Therefore, `poutcome` is chosen for the first split of the decision tree, as shown in Figure 7-4. The values of information gain in Table 7-2 are small in magnitude, but the relative difference matters. The algorithm splits on the attribute with the largest information gain at each round.

TABLE 7-2 *Calculating Information Gain of Input Variables for the First Split*

Attribute	Information Gain
`poutcome`	0.0289
`contact`	0.0201
`housing`	0.0133
`job`	0.0101
`education`	0.0034
`marital`	0.0018
`loan`	0.0010
`default`	0.0005

Detecting Significant Splits

Quite often it is necessary to measure the significance of a split in a decision tree, especially when the information gain is small, like in Table 7-2.

Let N_A and N_B be the number of class A and class B in the parent node. Let N_{AL} represent the number of class A going to the left child node, N_{BL} represent the number of class B going to the left child node, N_{AR} represent the number of class B going to the right child node, and N_{BR} represent the number of class B going to the right child node.

Let p_L and p_R denote the proportion of data going to the left and right node, respectively.

$$p_L = \frac{N_{AL} + N_{BL}}{N_A + N_B}$$

$$p_R = \frac{N_{AR} + N_{BR}}{N_A + N_B}$$

The following measure computes the significance of a split. In other words, it measures how much the split deviates from what would be expected in the random data.

$$K = \frac{\left(N'_{AL} - N_{AL}\right)^2}{N'_{AL}} + \frac{\left(N'_{BL} - N_{BL}\right)^2}{N'_{BL}} + \frac{\left(N'_{AR} - N_{AR}\right)^2}{N'_{AR}} + \frac{\left(N'_{BR} - N_{BR}\right)^2}{N'_{BR}}$$

where

$$N'_{AL} = N_A \times p_L$$

$$N'_{BL} = N_B \times p_L$$

$$N'_{AR} = N_A \times p_R$$

$$N'_{BR} = N_B \times p_R$$

If K is small, the information gain from the split is not significant. If K is big, it would suggest the information gain from the split is significant.

Take the first split of the decision tree in Figure 7-4 on variable `poutcome` for example. $N_A = 1789, N_B = 211, N_{AL} = 1763, N_{BL} = 179, N_{AR} = 26, N_{BR} = 32$.

Following are the proportions of data going to the left and right node.

$$p_L = {}^{1942}\!/_{2000} = 0.971 \text{ and } p_R = {}^{58}\!/_{2000} = 0.029.$$

The N'_{AL}, N'_{BL}, N'_{AR}, and N'_{BR} represent the number of each class going to the left or right node if the data is random. Their values follow.

$$N'_{AL} = 1737.119, N'_{BL} = 204.881, N'_{AR} = 51.881 \text{ and } N'_{BR} = 6.119$$

Therefore, $K = 126.0324$, which suggests the split on `poutcome` is significant.

After each split, the algorithm looks at all the records at a leaf node, and the information gain of each candidate attribute is calculated again over these records. The next split is on the attribute with the highest information gain. A record can only belong to one leaf node after all the splits, but depending on the implementation, an attribute may appear in more than one split of the tree. This process of partitioning the records and finding the most informative attribute is repeated until the nodes are pure enough, or there is insufficient information gain by splitting on more attributes. Alternatively, one can stop the growth of the tree when all the nodes at a leaf node belong to a certain class (for example, `subscribed` = yes) or all the records have identical attribute values.

In the previous bank marketing example, to keep it simple, the dataset only includes categorical variables. Assume the dataset now includes a continuous variable called `duration`–representing the number of seconds the last phone conversation with the bank lasted as part of the previous marketing campaign. A continuous variable needs to be divided into a disjoint set of regions with the highest information gain.

A brute-force method is to consider every value of the continuous variable in the training data as a candidate split position. This brute-force method is computationally inefficient. To reduce the complexity, the training records can be sorted based on the duration, and the candidate splits can be identified by taking the midpoints between two adjacent sorted values. An examples is if the duration consists of sorted values {140, 160, 180, 200} and the candidate splits are 150, 170, and 190.

Figure 7-6 shows what the decision tree may look like when considering the *duration* attribute. The root splits into two partitions: those clients with duration $<$ 456 seconds, and those with duration $\geq$ 456 seconds. Note that for aesthetic purposes, labels for the *job* and *contact* attributes in the figure are abbreviated.

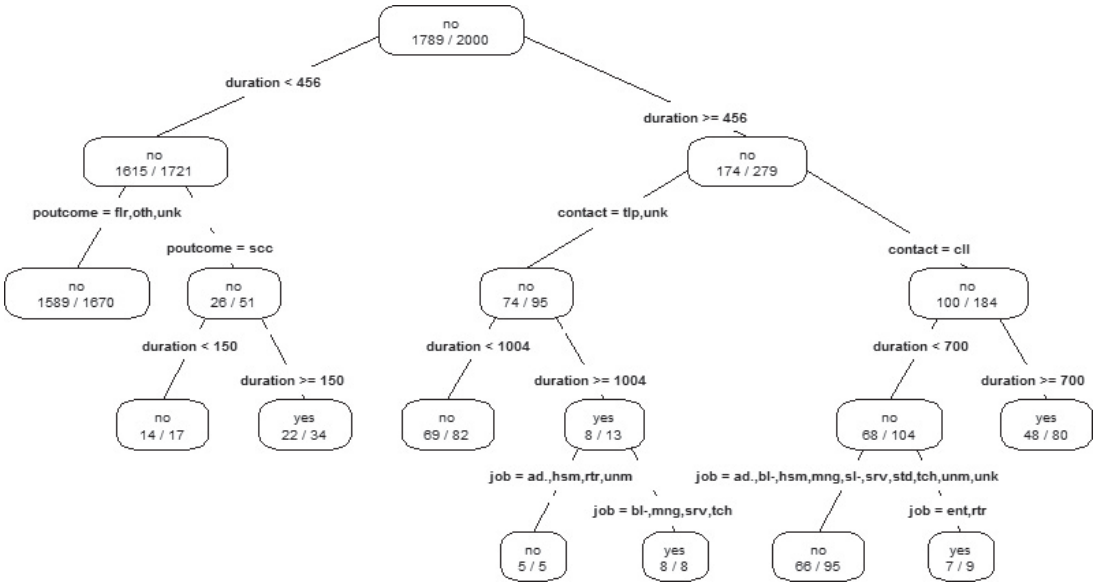

FIGURE 7-6 *Decision tree with attribute duration*

With the decision tree in Figure 7-6, it becomes trivial to predict if a new client is going to subscribe to the term deposit. For example, given the record of a new client shown in Table 7-3, the prediction is that this client will subscribe to the term deposit. The traversed paths in the decision tree are as follows.

- *duration* $\geq$ 456
- *contact* = cll (cellular)
- *duration* $<$ 700
- *job* = ent (entrepreneur), rtr (retired)

TABLE 7-3 *Record of a New Client*

Job	Marital	Education	Default	Housing	Loan	Contact	Duration	Poutcome
retired	married	secondary	no	yes	No	cellular	598	unknown

7.1.3 Decision Tree Algorithms

Multiple algorithms exist to implement decision trees, and the methods of tree construction vary with different algorithms. Some popular algorithms include ID3 [7], C4.5[8], and CART [9].

ID3 Algorithm

ID3 (or Iterative Dichotomiser 3) [7] is one of the first decision tree algorithms, and it was developed by John Ross Quinlan. Let A be a set of categorical input variables, P be the output variable (or the predicted class), and T be the training set. The ID3 algorithm is shown here.

```
1   ID3 (A, P, T)
2    if T∈φ
3      return φ
4    if all records in T have the same value for P
5      return a single node with that value
6    if A∈φ
7      return a single node with the most frequent value of P in T
8    Compute information gain for each attribute in A relative to T
9    Pick attribute D with the largest gain
10   Let { d1,d2…dm } be the values of attribute D
11   Partition T into { T1,T2…Tm } according to the values of D
12   return a tree with root D and branches labeled d1,d2…dm
           going respectively to trees ID3(A-{D}, P, T1),
           ID3(A-{D}, P, T2), . . . ID3(A-{D}, P, Tm)
```

C4.5

The C4.5 algorithm [8] introduces a number of improvements over the original ID3 algorithm. The C4.5 algorithm can handle missing data. If the training records contain unknown attribute values, the C4.5 evaluates the gain for an attribute by considering only the records where the attribute is defined.

Both categorical and continuous attributes are supported by C4.5. Values of a continuous variable are sorted and partitioned. For the corresponding records of each partition, the gain is calculated, and the partition that maximizes the gain is chosen for the next split.

The ID3 algorithm may construct a deep and complex tree, which would cause overfitting (Section 7.1.4). The C4.5 algorithm addresses the overfitting problem in ID3 by using a bottom-up technique called ***pruning*** to simplify the tree by removing the least visited nodes and branches.

CART

CART (or Classification And Regression Trees) [9] is often used as a generic acronym for the decision tree, although it is a specific implementation.

Similar to C4.5, CART can handle continuous attributes. Whereas C4.5 uses entropy-based criteria to rank tests, CART uses the Gini diversity index defined in Equation 7-5.

$$Gini_X = 1 - \sum_{\forall x \in X} P(x)^2 \tag{7-5}$$

Whereas C4.5 employs stopping rules, CART constructs a sequence of subtrees, uses cross-validation to estimate the misclassification cost of each subtree, and chooses the one with the lowest cost.

7.1.4 Evaluating a Decision Tree

Decision trees use ***greedy algorithms***, in that they always choose the option that seems the best available at that moment. At each step, the algorithm selects which attribute to use for splitting the remaining records. This selection may not be the best overall, but it is guaranteed to be the best at that step. This characteristic reinforces the efficiency of decision trees. However, once a bad split is taken, it is propagated through the rest of the tree. To address this problem, an ensemble technique (such as random forest) may randomize the splitting or even randomize data and come up with a multiple tree structure. These trees then vote for each class, and the class with the most votes is chosen as the predicted class.

There are a few ways to evaluate a decision tree. First, evaluate whether the splits of the tree make sense. Conduct sanity checks by validating the decision rules with domain experts, and determine if the decision rules are sound.

Next, look at the depth and nodes of the tree. Having too many layers and obtaining nodes with few members might be signs of overfitting. In overfitting, the model fits the training set well, but it performs poorly on the new samples in the testing set. Figure 7-7 illustrates the performance of an overfit model. The *x*-axis represents the amount of data, and the *y*-axis represents the errors. The blue curve is the training set, and the red curve is the testing set. The left side of the gray vertical line shows that the model predicts well on the testing set. But on the right side of the gray line, the model performs worse and worse on the testing set as more and more unseen data is introduced.

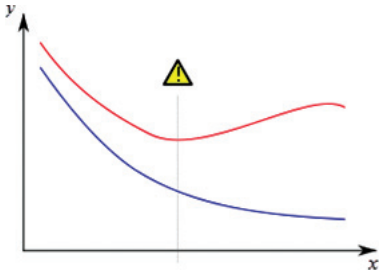

FIGURE 7-7 *An overfit model describes the training data well but predicts poorly on unseen data*

For decision tree learning, overfitting can be caused by either the lack of training data or the biased data in the training set. Two approaches [10] can help avoid overfitting in decision tree learning.

- Stop growing the tree early before it reaches the point where all the training data is perfectly classified.
- Grow the full tree, and then post-prune the tree with methods such as reduced-error pruning and rule-based post pruning.

Last, many standard diagnostics tools that apply to classifiers can help evaluate overfitting. These tools are further discussed in Section 7.3.

Decision trees are computationally inexpensive, and it is easy to classify the data. The outputs are easy to interpret as a fixed sequence of simple tests. Many decision tree algorithms are able to show the importance of each input variable. Basic measures, such as information gain, are provided by most statistical software packages.

Decision trees are able to handle both numerical and categorical attributes and are robust with redundant or correlated variables. Decision trees can handle categorical attributes with many distinct values, such as country codes for telephone numbers. Decision trees can also handle variables that have a nonlinear effect on the outcome, so they work better than linear models (for example, linear regression and logistic regression) for highly nonlinear problems. Decision trees naturally handle variable interactions. Every node in the tree depends on the preceding nodes in the tree.

In a decision tree, the decision regions are rectangular surfaces. Figure 7-8 shows an example of five rectangular decision surfaces (A, B, C, D, and E) defined by four values— $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$ —of two attributes (x_1 and x_2). The corresponding decision tree is on the right side of the figure. A decision surface corresponds to a leaf node of the tree, and it can be reached by traversing from the root of the tree following by a series of decisions according to the value of an attribute. The decision surface can only be axis-aligned for the decision tree.

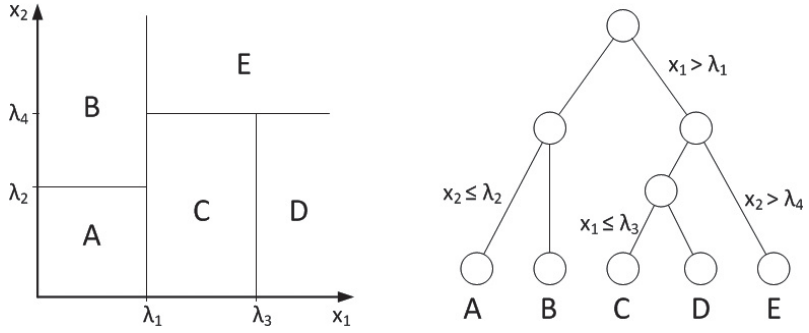

Figure 7-8 *Decision surfaces can only be axis-aligned*

The structure of a decision tree is sensitive to small variations in the training data. Although the dataset is the same, constructing two decision trees based on two different subsets may result in very different trees. If a tree is too deep, overfitting may occur, because each split reduces the training data for subsequent splits.

Decision trees are not a good choice if the dataset contains many irrelevant variables. This is different from the notion that they are robust with redundant variables and correlated variables. If the dataset

contains redundant variables, the resulting decision tree ignores all but one of these variables because the algorithm cannot detect information gain by including more redundant variables. On the other hand, if the dataset contains irrelevant variables and if these variables are accidentally chosen as splits in the tree, the tree may grow too large and may end up with less data at every split, where overfitting is likely to occur. To address this problem, feature selection can be introduced in the data preprocessing phase to eliminate the irrelevant variables.

Although decision trees are able to handle correlated variables, decision trees are not well suited when most of the variables in the training set are correlated, since overfitting is likely to occur. To overcome the issue of instability and potential overfitting of deep trees, one can combine the decisions of several randomized shallow decision trees—the basic idea of another classifier called random forest [4]—or use ensemble methods to combine several weak learners for better classification. These methods have been shown to improve predictive power compared to a single decision tree.

For binary decisions, a decision tree works better if the training dataset consists of records with an even probability of each result. In other words, the root of the tree has a 50% chance of either classification. This occurs by randomly selecting training records from each possible classification in equal numbers. It counteracts the likelihood that a tree will stump out early by passing purity tests because of bias in the training data.

When using methods such as logistic regression on a dataset with many variables, decision trees can help determine which variables are the most useful to select based on information gain. Then these variables can be selected for the logistic regression. Decision trees can also be used to prune redundant variables.

7.1.5 Decision Trees in R

In R, `rpart` is for modeling decision trees, and an optional package `rpart.plot` enables the plotting of a tree. The rest of this section shows an example of how to use decision trees in R with `rpart.plot` to predict whether to play golf given factors such as weather outlook, temperature, humidity, and wind.

In R, first set the working directory and initialize the packages.

```
setwd("c:/")
install.packages("rpart.plot")   # install package rpart.plot
library("rpart")   # load libraries
library("rpart.plot")
```

The working directory contains a comma-separated-value (CSV) file named `DTdata.csv`. The file has a header row, followed by 10 rows of training data.

```
Play,Outlook,Temperature,Humidity,Wind
yes,rainy,cool,normal,FALSE
no,rainy,cool,normal,TRUE
yes,overcast,hot,high,FALSE
no,sunny,mild,high,FALSE
yes,rainy,cool,normal,FALSE
yes,sunny,cool,normal,FALSE
yes,rainy,cool,normal,FALSE
yes,sunny,hot,normal,FALSE
yes,overcast,mild,high,TRUE
no,sunny,mild,high,TRUE
```

The CSV file contains five attributes: *Play*, *Outlook*, *Temperature*, *Humidity*, and *Wind*. *Play* would be the output variable (or the predicted class), and *Outlook*, *Temperature*, *Humidity*, and *Wind* would be the input variables. In R, read the data from the CSV file in the working directory and display the content.

```
play_decision <- read.table("DTdata.csv",header=TRUE,sep=",")
play_decision
```

```
   Play  Outlook Temperature Humidity  Wind
1   yes    rainy        cool   normal FALSE
2    no    rainy        cool   normal  TRUE
3   yes overcast         hot     high FALSE
4    no    sunny        mild     high FALSE
5   yes    rainy        cool   normal FALSE
6   yes    sunny        cool   normal FALSE
7   yes    rainy        cool   normal FALSE
8   yes    sunny         hot   normal FALSE
9   yes overcast        mild     high  TRUE
10   no    sunny        mild     high  TRUE
```

Display a summary of *play_decision*.

```
summary(play_decision)
```

```
  Play       Outlook  Temperature   Humidity    Wind
 no :3   overcast:2   cool:5      high  :4   Mode :logical
 yes:7   rainy   :4   hot :2      normal:6   FALSE:7
         sunny   :4   mild:3                 TRUE :3
                                            NA's :0
```

The `rpart` function builds a model of recursive partitioning and regression trees [9]. The following code snippet shows how to use the `rpart` function to construct a decision tree.

```
fit <- rpart(Play ~ Outlook + Temperature + Humidity + Wind,
             method="class",
             data=play_decision,
             control=rpart.control(minsplit=1),
             parms=list(split='information'))
```

The `rpart` function has four parameters. The first parameter, `Play ~ Outlook + Temperature + Humidity + Wind`, is the model indicating that attribute *Play* can be predicted based on attributes *Outlook*, *Temperature*, *Humidity*, and *Wind*. The second parameter, *method*, is set to "class," telling R it is building a classification tree. The third parameter, *data*, specifies the dataframe containing those attributes mentioned in the formula. The fourth parameter, *control*, is optional and controls the tree growth. In the preceding example, `control=rpart.control(minsplit=1)` requires that each node have at least one observation before attempting a split. The `minsplit=1` makes sense for the small dataset, but for larger datasets `minsplit` could be set to 10% of the dataset size to combat overfitting. Besides `minsplit`, other parameters are available to control the construction of the decision tree. For example, `rpart.control(maxdepth=10,cp=0.001)` limits the depth of the

tree to no more than 10, and a split must decrease the overall lack of fit by a factor of 0.001 before being attempted. The last parameter (`parms`) specifies the purity measure being used for the splits. The value of `split` can be either `information` (for using the information gain) or `gini` (for using the Gini index).

Enter **`summary(fit)`** to produce a summary of the model built from `rpart`.

The output includes a summary of every node in the constructed decision tree. If a node is a leaf, the output includes both the predicted class label (*yes* or *no* for *Play*) and the class probabilities—`P(`*Play*`)`. The leaf nodes include node numbers 4, 5, 6, and 7. If a node is internal, the output in addition displays the number of observations that lead to each child node and the improvement that each attribute may bring for the next split. These internal nodes include numbers 1, 2, and 3.

```
summary(fit)
Call:
rpart(formula = Play ~ Outlook + Temperature + Humidity + Wind,
  data = play_decision, method = "class",
  parms = list(split = "information"),
      control = rpart.control(minsplit = 1))
  n= 10
         CP nsplit rel error   xerror      xstd
1 0.3333333      0         1 1.000000 0.4830459
2 0.0100000      3         0 1.666667 0.5270463

Variable importance
       Wind     Outlook Temperature
         51          29          20

Node number 1: 10 observations,    complexity param=0.3333333
  predicted class=yes  expected loss=0.3  P(node) =1
    class counts:     3     7
   probabilities: 0.300 0.700
  left son=2 (3 obs) right son=3 (7 obs)
  Primary splits:
      Temperature splits as  RRL,     improve=1.3282860, (0 missing)
      Wind        < 0.5 to the right, improve=1.3282860, (0 missing)
      Outlook     splits as  RLL,     improve=0.8161371, (0 missing)
      Humidity    splits as  LR,      improve=0.6326870, (0 missing)
  Surrogate splits:
      Wind < 0.5 to the right, agree=0.8, adj=0.333, (0 split)

Node number 2: 3 observations,    complexity param=0.3333333
  predicted class=no   expected loss=0.3333333  P(node) =0.3
    class counts:     2     1
   probabilities: 0.667 0.333
```

```
  left son=4 (2 obs) right son=5 (1 obs)
  Primary splits:
      Outlook splits as  R-L,     improve=1.9095430, (0 missing)
      Wind    < 0.5 to the left,  improve=0.5232481, (0 missing)

Node number 3: 7 observations,    complexity param=0.3333333
  predicted class=yes  expected loss=0.1428571  P(node) =0.7
    class counts:     1     6
   probabilities: 0.143 0.857
  left son=6 (1 obs) right son=7 (6 obs)
  Primary splits:
      Wind        < 0.5 to the right, improve=2.8708140, (0 missing)
      Outlook     splits as  RLR,     improve=0.6214736, (0 missing)
      Temperature splits as  LR-,     improve=0.3688021, (0 missing)
      Humidity    splits as  RL,      improve=0.1674470, (0 missing)

Node number 4: 2 observations
  predicted class=no   expected loss=0  P(node) =0.2
    class counts:     2     0
   probabilities: 1.000 0.000

Node number 5: 1 observations
  predicted class=yes  expected loss=0  P(node) =0.1
    class counts:     0     1
   probabilities: 0.000 1.000

Node number 6: 1 observations
  predicted class=no   expected loss=0  P(node) =0.1
    class counts:     1     0
   probabilities: 1.000 0.000

Node number 7: 6 observations
  predicted class=yes  expected loss=0  P(node) =0.6
    class counts:     0     6
   probabilities: 0.000 1.000
```

The output produced by the summary is difficult to read and comprehend. The `rpart.plot()` function from the `rpart.plot` package can visually represent the output in a decision tree. Enter the following command to see the help file of `rpart.plot`:

```
?rpart.plot
```

Enter the following R code to plot the tree based on the model being built. The resulting tree is shown in Figure 7-9. Each node of the tree is labeled as either `yes` or `no` referring to the *`Play`* action of whether to play outside. Note that, by default, R has converted the values of *`Wind`* (`True/False`) into numbers.

```
rpart.plot(fit, type=4, extra=1)
```

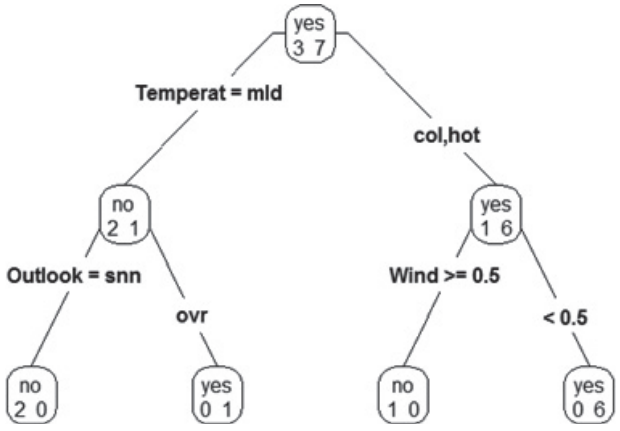

FIGURE 7-9 *A decision tree built from DTdata.csv*

The decisions in Figure 7-9 are abbreviated. Use the following command to spell out the full names and display the classification rate at each node.

```
rpart.plot(fit, type=4, extra=2, clip.right.labs=FALSE,
           varlen=0, faclen=0)
```

The decision tree can be used to predict outcomes for new datasets. Consider a testing set that contains the following record.

```
Outlook="rainy", Temperature="mild", Humidity="high", Wind=FALSE
```

The goal is to predict the play decision of this record. The following code loads the data into R as a data frame *`newdata`*. Note that the training set does not contain this case.

```
newdata <- data.frame(Outlook="rainy", Temperature="mild",
                      Humidity="high", Wind=FALSE)

newdata
  Outlook Temperature Humidity  Wind
1   rainy        mild     high FALSE
```

Next, use the *`predict`* function to generate predictions from a fitted *`rpart`* object. The format of the `predict` function follows.

```
predict(object, newdata = list(),
        type = c("vector", "prob", "class", "matrix"))
```

Parameter *type* is a character string denoting the type of the predicted value. Set it to either `prob` or `class` to predict using a decision tree model and receive the result as either the class probabilities or just the class. The output shows that one instance is classified as *Play*=`no`, and zero instances are classified as *Play*=`yes`. Therefore, in both cases, the decision tree predicts that the play decision of the testing set is not to play.

```
predict(fit,newdata=newdata,type="prob")

  no yes
1  1   0


predict(fit,newdata=newdata,type="class")

  1
  no
Levels: no yes
```

7.2 Naïve Bayes

Naïve Bayes is a probabilistic classification method based on Bayes' theorem (or Bayes' law) with a few tweaks. Bayes' theorem gives the relationship between the probabilities of two events and their conditional probabilities. Bayes' law is named after the English mathematician Thomas Bayes.

A naïve Bayes classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of other features. For example, an object can be classified based on its attributes such as shape, color, and weight. A reasonable classification for an object that is spherical, yellow, and less than 60 grams in weight may be a tennis ball. Even if these features depend on each other or upon the existence of the other features, a naïve Bayes classifier considers all these properties to contribute *independently* to the probability that the object is a tennis ball.

The input variables are generally categorical, but variations of the algorithm can accept continuous variables. There are also ways to convert continuous variables into categorical ones. This process is often referred to as the ***discretization of continuous variables***. In the tennis ball example, a continuous variable such as weight can be grouped into intervals to be converted into a categorical variable. For an attribute such as *income*, the attribute can be converted into categorical values as shown below.

- **Low Income:** income < $10,000
- **Working Class:** $10,000 ≤ income < $50,000
- **Middle Class:** $50,000 ≤ income < $1,000,000
- **Upper Class:** income ≥ $1,000,000

The output typically includes a class label and its corresponding probability score. The probability score is not the true probability of the class label, but it's proportional to the true probability. As shown later in the chapter, in most implementations, the output includes the log probability for the class, and class labels are assigned based on the highest values.

Because naïve Bayes classifiers are easy to implement and can execute efficiently even without prior knowledge of the data, they are among the most popular algorithms for classifying text documents. Spam filtering is a classic use case of naïve Bayes text classification. Bayesian spam filtering has become a popular mechanism to distinguish spam e-mail from legitimate e-mail. Many modern mail clients implement variants of Bayesian spam filtering.

Naïve Bayes classifiers can also be used for fraud detection [11]. In the domain of auto insurance, for example, based on a training set with attributes such as driver's rating, vehicle age, vehicle price, historical claims by the policy holder, police report status, and claim genuineness, naïve Bayes can provide probability-based classification of whether a new claim is genuine [12].

7.2.1 Bayes' Theorem

The ***conditional probability*** of event C occurring, given that event A has already occurred, is denoted as $P(C|A)$, which can be found using the formula in Equation 7-6.

$$P(C|A) = \frac{P(A \cap C)}{P(A)} \tag{7-6}$$

Equation 7-7 can be obtained with some minor algebra and substitution of the conditional probability:

$$P(C|A) = \frac{P(A|C) \cdot P(C)}{P(A)} \tag{7-7}$$

where C is the class label $C \in \{c_1, c_2, \ldots c_n\}$ and A is the observed attributes $A = \{a_1, a_2, \ldots a_m\}$. Equation 7-7 is the most common form of the ***Bayes' theorem***.

Mathematically, Bayes' theorem gives the relationship between the probabilities of C and A, $P(C)$ and $P(A)$, and the conditional probabilities of C given A and A given C, namely $P(C|A)$ and $P(A|C)$.

Bayes' theorem is significant because quite often $P(C|A)$ is much more difficult to compute than $P(A|C)$ and $P(C)$ from the training data. By using Bayes' theorem, this problem can be circumvented.

An example better illustrates the use of Bayes' theorem. John flies frequently and likes to upgrade his seat to first class. He has determined that if he checks in for his flight at least two hours early, the probability that he will get an upgrade is 0.75; otherwise, the probability that he will get an upgrade is 0.35. With his busy schedule, he checks in at least two hours before his flight only 40% of the time. Suppose John did not receive an upgrade on his most recent attempt. What is the probability that he did not arrive two hours early?

Let C = {John arrived at least two hours early}, and A = {John received an upgrade}, then $\neg C$ = {John did not arrive two hours early}, and $\neg A$ = {John did not receive an upgrade}.

John checked in at least two hours early only 40% of the time, or $P(C) = 0.4$. Therefore, $P(\neg C) = 1 - P(C) = 0.6$.

The probability that John received an upgrade given that he checked in early is 0.75, or $P(A|C) = 0.75$.

The probability that John received an upgrade given that he did not arrive two hours early is 0.35, or $P(A|\neg C) = 0.35$. Therefore, $P(\neg A|\neg C) = 0.65$.

The probability that John received an upgrade $P(A)$ can be computed as shown in Equation 7-8.

$$\begin{aligned} P(A) &= P(A \cap C) + P(A \cap \neg C) \\ &= P(C) \cdot P(A|C) + P(\neg C) \cdot P(A|\neg C) \\ &= 0.4 \times 0.75 + 0.6 \times 0.35 \\ &= 0.51 \end{aligned} \tag{7-8}$$

Thus, the probability that John did not receive an upgrade $P(\neg A) = 0.49$. Using Bayes' theorem, the probability that John did not arrive two hours early given that he did not receive his upgrade is shown in Equation 7-9.

$$\begin{aligned} P(\neg C|\neg A) &= \frac{P(\neg A|\neg C) \cdot P(\neg C)}{P(\neg A)} \\ &= \frac{0.65 \times 0.6}{0.49} \approx 0.796 \end{aligned} \tag{7-9}$$

Another example involves computing the probability that a patient carries a disease based on the result of a lab test. Assume that a patient named Mary took a lab test for a certain disease and the result came back positive. The test returns a positive result in 95% of the cases in which the disease is actually present, and it returns a positive result in 6% of the cases in which the disease is not present. Furthermore, 1% of the entire population has this disease. What is the probability that Mary actually has the disease, given that the test is positive?

Let C = {having the disease} and A = {testing positive}. The goal is to solve the probability of having the disease, given that Mary has a positive test result, $P(C|A)$. From the problem description, $P(C) = 0.01$, $P(\neg C) = 0.99$, $P(A|C) = 0.95$ and $P(A|\neg C) = 0.06$.

Bayes' theorem defines $P(C \mid A) = P(A \mid C)P(C) / P(A)$. The probability of testing positive, that is $P(A)$, needs to be computed first. That computation is shown in Equation 7-10.

$$\begin{aligned} P(A) &= P(A \cap C) + P(A \cap \neg C) \\ &= P(C) \cdot P(A|C) + P(\neg C) \cdot P(A|\neg C) \\ &= 0.01 \times 0.95 + 0.99 \times 0.06 = 0.0689 \end{aligned} \tag{7-10}$$

According to Bayes' theorem, the probability of having the disease, given that Mary has a positive test result, is shown in Equation 7-11.

$$P(C|A) = \frac{P(A|C)P(C)}{P(A)} = \frac{0.95 \times 0.01}{0.0689} \approx 0.1379 \tag{7-11}$$

That means that the probability of Mary actually having the disease given a positive test result is only 13.79%. This result indicates that the lab test may not be a good one. The likelihood of having the disease

was 1% when the patient walked in the door and only 13.79% when the patient walked out, which would suggest further tests.

A more general form of Bayes' theorem assigns a classified label to an object with multiple attributes $A = \{a_1, a_2, \ldots, a_m\}$ such that the label corresponds to the largest value of $P(c_i|A)$. The probability that a set of attribute values A (composed of m variables $a_1, a_2, \ldots, a_m$) should be labeled with a classification label c_i equals the probability that the set of variables $a_1, a_2, \ldots, a_m$ given c_i is true, times the probability of c_i divided by the probability of $a_1, a_2, \ldots, a_m$. Mathematically, this is shown in Equation 7-12.

$$P(c_i|A) = \frac{P(a_1, a_2, \ldots, a_m|c_i) \cdot P(c_i)}{P(a_1, a_2, \ldots, a_m)}, i = 1, 2, \ldots n \tag{7-12}$$

Consider the bank marketing example presented in Section 7.1 on predicting if a customer would subscribe to a term deposit. Let A be a list of attributes {`job`, `marital`, `education`, `default`, `housing`, `loan`, `contact`, `poutcome`}. According to Equation 7-12, the problem is essentially to calculate $P(c_i|A)$, where $c_i \in \{subscribed = yes, subscribed = no\}$.

7.2.2 Naïve Bayes Classifier

With two simplifications, Bayes' theorem can be extended to become a naïve Bayes classifier.

The first simplification is to use the conditional independence assumption. That is, each attribute is conditionally independent of every other attribute given a class label c_i. See Equation 7-13.

$$P(a_1, a_2, \ldots, a_m|c_i) = P(a_1|c_i)P(a_2|c_i)\cdots P(a_m|c_i) = \prod_{j=1}^{m} P(a_j|c_i) \tag{7-13}$$

Therefore, this naïve assumption simplifies the computation of $P(a_1, a_2, \ldots, a_m|c_i)$.

The second simplification is to ignore the denominator $P(a_1, a_2, \ldots, a_m)$. Because $P(a_1, a_2, \ldots, a_m)$ appears in the denominator of $P(c_i|A)$ for all values of `i`, removing the denominator will have no impact on the relative probability scores and will simplify calculations.

Naïve Bayes classification applies the two simplifications mentioned earlier and, as a result, $P(c_i|a_1, a_2, \ldots, a_m)$ is proportional to the product of $P(a_j|c_i)$ times $P(c_i)$. This is shown in Equation 7-14.

$$P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i) \qquad i = 1, 2, \ldots n \tag{7-14}$$

The mathematical symbol $\propto$ indicates that the LHS $P(c_i|A)$ is directly proportional to the RHS.

Section 7.1 has introduced a bank marketing dataset (Figure 7-3). This section shows how to use the naïve Bayes classifier on this dataset to predict if the clients would subscribe to a term deposit.

Building a naïve Bayes classifier requires knowing certain statistics, all calculated from the training set. The first requirement is to collect the probabilities of all class labels, $P(c_i)$. In the presented example, these would be the probability that a client will subscribe to the term deposit and the probability the client will not. From the data available in the training set, $\mathrm{P}(\textit{subscribed} = \text{yes}) \approx 0.11$ and $\mathrm{P}(\textit{subscribed} = \text{no}) \approx 0.89$.

The second thing the naïve Bayes classifier needs to know is the conditional probabilities of each attribute a_j given each class label c_i, namely $P(a_j|c_i)$. The training set contains several attributes: `job`, `marital`, `education`, `default`, `housing`, `loan`, `contact`, and `poutcome`. For each attribute and its possible values, computing the conditional probabilities given *subscribed* = yes or *subscribed* = no is required. For example, relative to the `marital` attribute, the following conditional probabilities are calculated.

$$P(\textit{single} \mid \textit{subscribed} = \textit{yes}) \approx 0.35$$

$$P(\textit{married} \mid \textit{subscribed} = \textit{yes}) \approx 0.53$$

$$P(\textit{divorced} \mid \textit{subscribed} = \textit{yes}) \approx 0.12$$

$$P(\textit{single} \mid \textit{subscribed} = \textit{no}) \approx 0.28$$

$$P(\textit{married} \mid \textit{subscribed} = \textit{no}) \approx 0.61$$

$$P(\textit{divorced} \mid \textit{subscribed} = \textit{no}) \approx 0.11$$

After training the classifier and computing all the required statistics, the naïve Bayes classifier can be tested over the testing set. For each record in the testing set, the naïve Bayes classifier assigns the classifier label c_i that maximizes $P(c_i) \cdot \prod_{j=1}^{m} P\left(a_j|c_i\right)$.

Table 7-4 contains a single record for a client who has a career in management, is married, holds a secondary degree, has credit not in default, has a housing loan but no personal loans, prefers to be contacted via cellular, and whose outcome of the previous marketing campaign contact was a success. Is this client likely to subscribe to the term deposit?

TABLE 7-4 *Record of an Additional Client*

Job	Marital	Education	Default	Housing	Loan	Contact	Poutcome
management	married	secondary	no	yes	no	cellular	Success

The conditional probabilities shown in Table 7-5 can be calculated after building the classifier with the training set.

TABLE 7-5 *Compute Conditional Probabilities for the New Record*

j	a_j	$P(a_j \mid \text{subscribed} = \text{yes})$	$P(a_j \mid \text{subscribed} = \text{no})$
1	job = management	0.22	0.21
2	marital = married	0.53	0.61
3	education = secondary	0.46	0.51
4	default = no	0.99	0.98
5	housing = yes	0.35	0.57
6	loan = no	0.90	0.85
7	contact = cellular	0.85	0.62
8	poutcome = success	0.15	0.01

Because $P(c_i|a_1,a_2,\ldots,a_m)$ is proportional to the product of $P(a_j|c_i)(j \in [1,m])$ times (c_i), the naïve Bayes classifier assigns the class label $c_{i,}$ which results in the greatest value over all i. Thus, $P(c_i|a_1,a_2,\ldots,a_m)$ is computed for each c_i with $P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P\left(a_j|c_i\right)$.

For A = {management, married, secondary, no, yes, no, cellular, success},

$$P(yes|A) \propto 0.11 \cdot (0.22 \cdot 0.53 \cdot 0.46 \cdot 0.99 \cdot 0.35 \cdot 0.90 \cdot 0.85 \cdot 0.15) \approx 0.00023$$

$$P(no|A) \propto 0.89 \cdot (0.21 \cdot 0.61 \cdot 0.51 \cdot 0.98 \cdot 0.57 \cdot 0.85 \cdot 0.62 \cdot 0.01) \approx 0.00017$$

Because $P(subscribed = yes|A) > P(subscribed = no|A)$, the client shown in Table 7-4 is assigned with the label $subscribed = yes$. That is, the client is classified as likely to subscribe to the term deposit.

Although the scores are small in magnitude, it is the ratio of $P(yes|A)$ and $P(no|A)$ that matters. In fact, the scores of $P(yes|A)$ and $P(no|A)$ are not the true probabilities but are only proportional to the true probabilities, as shown in Equation 7-14. After all, if the scores were indeed the true probabilities, the sum of $P(yes|A)$ and $P(no|A)$ would be equal to one. When looking at problems with a large number of attributes, or attributes with a high number of levels, these values can become very small in magnitude (close to zero), resulting in even smaller differences of the scores. This is the problem of ***numerical underflow***, caused by multiplying several probability values that are close to zero. A way to alleviate the problem is to compute

the logarithm of the products, which is equivalent to the summation of the logarithm of the probabilities. Thus, the naïve Bayes formula can be rewritten as shown in Equation 7-15.

$$P(c_i|A) \propto \log P(c_i) + \sum_{j=1}^{m} \log P(a_j|c_i) \qquad i = 1,2,\ldots n \tag{7-15}$$

Although the risk of underflow may increase as the number of attributes increases, the use of logarithms is usually applied regardless of the number of attribute dimensions.

7.2.3 Smoothing

If one of the attribute values does not appear with one of the class labels within the training set, the corresponding $P(a_j|c_i)$ will equal zero. When this happens, the resulting $P(c_i|A)$ from multiplying all the $P(a_j|c_i)(j \in [1,m])$ immediately becomes zero regardless of how large some of the conditional probabilities are. Therefore overfitting occurs. Smoothing techniques can be employed to adjust the probabilities of $P(a_j|c_i)$ and to ensure a nonzero value of $P(c_i|A)$. A smoothing technique assigns a small nonzero probability to rare events not included in the training dataset. Also, the smoothing addresses the possibility of taking the logarithm of zero that may occur in Equation 7-15.

There are various smoothing techniques. Among them is the ***Laplace smoothing*** (or add-one) technique that pretends to see every outcome once more than it actually appears. This technique is shown in Equation 7-16.

$$P^*(x) = \frac{count(x)+1}{\sum_x [count(x)+1]} \tag{7-16}$$

For example, say that 100 clients subscribe to the term deposit, with 20 of them single, 70 married, and 10 divorced. The "raw" probability is $P(single \,|\, subscribed = yes) = 20/100 = 0.2$. With Laplace smoothing adding one to the counts, the adjusted probability becomes $P'(single \,|\, subscribed = yes) = (20+1)/[(20+1)+(70+1)+(10+1)] \approx 0.2039$.

One problem of the Laplace smoothing is that it may assign too much probability to unseen events. To address this problem, Laplace smoothing can be generalized to use ε instead of 1, where typically $\varepsilon \in [0,1]$. See Equation 7-17.

$$P^{**}(x) = \frac{count(x)+\varepsilon}{\sum_x [count(x)+\varepsilon]} \tag{7-17}$$

Smoothing techniques are available in most standard software packages for naïve Bayes classifiers. However, if for some reason (like performance concerns) the naïve Bayes classifier needs to be coded directly into an application, the smoothing and logarithm calculations should be incorporated into the implementation.

7.2.4 Diagnostics

Unlike logistic regression, naïve Bayes classifiers can handle missing values. Naïve Bayes is also robust to irrelevant variables—variables that are distributed among all the classes whose effects are not pronounced.

The model is simple to implement even without using libraries. The prediction is based on counting the occurrences of events, making the classifier efficient to run. Naïve Bayes is computationally efficient and is able to handle high-dimensional data efficiently. Related research [13] shows that the naive Bayes classifier in many cases is competitive with other learning algorithms, including decision trees and neural networks. In some cases naïve Bayes even outperforms other methods. Unlike logistic regression, the naïve Bayes classifier can handle categorical variables with many levels. Recall that decision trees can handle categorical variables as well, but too many levels may result in a deep tree. The naïve Bayes classifier overall performs better than decision trees on categorical values with many levels. Compared to decision trees, naïve Bayes is more resistant to overfitting, especially with the presence of a smoothing technique.

Despite the benefits of naïve Bayes, it also comes with a few disadvantages. Naïve Bayes assumes the variables in the data are conditionally independent. Therefore, it is sensitive to correlated variables because the algorithm may double count the effects. As an example, assume that people with low income and low credit tend to default. If the task is to score "default" based on both income and credit as two separate attributes, naïve Bayes would experience the double-counting effect on the default outcome, thus reducing the accuracy of the prediction.

Although probabilities are provided as part of the output for the prediction, naïve Bayes classifiers in general are not very reliable for probability estimation and should be used only for assigning class labels. Naïve Bayes in its simple form is used only with categorical variables. Any continuous variables should be converted into a categorical variable with the process known as discretization, as shown earlier. In common statistical software packages, however, naïve Bayes is implemented in a way that enables it to handle continuous variables as well.

7.2.5 Naïve Bayes in R

This section explores two methods of using the naïve Bayes classifier in R. The first method is to build from scratch by manually computing the probability scores, and the second method is to use the *naiveBayes* function from the *e1071* package. The examples show how to use naïve Bayes to predict whether employees would enroll in an onsite educational program.

In R, first set up the working directory and initialize the packages.

```
setwd("c:/")
install.packages("e1071")   # install package e1071
library(e1071)   # load the library
```

The working directory contains a CSV file (`sample1.csv`). The file has a header row, followed by 14 rows of training data. The attributes include *Age*, *Income*, *JobSatisfaction*, and *Desire*. The output variable is *Enrolls*, and its value is either `Yes` or `No`. Full content of the CSV file is shown next.

```
Age,Income,JobSatisfaction,Desire,Enrolls
<=30,High,No,Fair,No
<=30,High,No,Excellent,No
31 to 40,High,No,Fair,Yes
>40,Medium,No,Fair,Yes
>40,Low,Yes,Fair,Yes
>40,Low,Yes,Excellent,No
31 to 40,Low,Yes,Excellent,Yes
```

```
<=30,Medium,No,Fair,No
<=30,Low,Yes,Fair,Yes
>40,Medium,Yes,Fair,Yes
<=30,Medium,Yes,Excellent,Yes
31 to 40,Medium,No,Excellent,Yes
31 to 40,High,Yes,Fair,Yes
>40,Medium,No,Excellent,No
<=30,Medium,Yes,Fair,
```

The last record of the CSV is used later for illustrative purposes as a test case. Therefore, it does not include a value for the output variable *Enrolls*, which should be predicted using the naïve Bayes classifier built from the training set.

Execute the following R code to read data from the CSV file.

```
# read the data into a table from the file
sample <- read.table("sample1.csv",header=TRUE,sep=",")
# define the data frames for the NB classifier
traindata <- as.data.frame(sample[1:14,])
testdata <- as.data.frame(sample[15,])
```

Two data frame objects called *traindata* and *testdata* are created for the naïve Bayes classifier. Enter *traindata* and *testdata* to display the data frames.

The two data frames are printed on the screen as follows.

```
traindata
        Age Income JobSatisfaction    Desire Enrolls
1      <=30   High              No      Fair      No
2      <=30   High              No Excellent      No
3  31 to 40   High              No      Fair     Yes
4       >40 Medium              No      Fair     Yes
5       >40    Low             Yes      Fair     Yes
6       >40    Low             Yes Excellent      No
7  31 to 40    Low             Yes Excellent     Yes
8      <=30 Medium              No      Fair      No
9      <=30    Low             Yes      Fair     Yes
10      >40 Medium             Yes      Fair     Yes
11     <=30 Medium             Yes Excellent     Yes
12 31 to 40 Medium              No Excellent     Yes
13 31 to 40   High             Yes      Fair     Yes
14      >40 Medium              No Excellent      No

testdata
    Age Income JobSatisfaction Desire Enrolls
15 <=30 Medium             Yes   Fair
```

The first method shown here is to build a naïve Bayes classifier from scratch by manually computing the probability scores. The first step in building a classifier is to compute the prior probabilities of the attributes,

including `Age`, `Income`, `JobSatisfaction`, and `Desire`. According to the naïve Bayes classifier, these attributes are conditionally independent. The dependent variable (output variable) is `Enrolls`.

Compute the prior probabilities $P(c_i)$ for `Enrolls`, where $c_i \in C$ and $C = \{Yes, No\}$.

```
tprior <- table(traindata$Enrolls)
tprior
     No Yes
  0   5   9

tprior <- tprior/sum(tprior)
tprior
                 No       Yes
0.0000000 0.3571429 0.6428571
```

The next step is to compute conditional probabilities $P(A|C)$, where $A = \{$Age, Income, JobSatisfaction, Desire$\}$ and $C = \{Yes, No\}$. Count the number of "No" and "Yes" entries for each `Age` group, and normalize by the total number of "No" and "Yes" entries to get the conditional probabilities.

```
ageCounts <- table(traindata[,c("Enrolls", "Age")])
ageCounts
       Age
Enrolls <=30 >40 31 to 40
           0   0        0
    No     3   2        0
   Yes     2   3        4


ageCounts <- ageCounts/rowSums(ageCounts)
ageCounts
       Age
Enrolls      <=30       >40  31 to 40

    No  0.6000000 0.4000000 0.0000000
    Yes 0.2222222 0.3333333 0.4444444
```

Do the same for the other attributes including `Income`, `JobSatisfaction`, and `Desire`.

```
incomeCounts <- table(traindata[,c("Enrolls", "Income")])
incomeCounts <- incomeCounts/rowSums(incomeCounts)
incomeCounts
       Income
Enrolls      High       Low    Medium

    No  0.4000000 0.2000000 0.4000000
    Yes 0.2222222 0.3333333 0.4444444

jsCounts <- table(traindata[,c("Enrolls", "JobSatisfaction")])
jsCounts <- jsCounts/rowSums(jsCounts)
```

```
jsCounts
       Jobsatisfaction
Enrolls        No       Yes

    No  0.8000000 0.2000000
    Yes 0.3333333 0.6666667

desireCounts <- table(traindata[,c("Enrolls", "Desire")])
desireCounts <- desireCounts/rowSums(desireCounts)
desireCounts
       Desire
Enrolls Excellent      Fair

    No  0.6000000 0.4000000
    Yes 0.3333333 0.6666667
```

According to Equation 7-7, probability $P(c_i|A)$ is determined by the product of $P(a_j|c_i)$ times the (c_i) where $c_1 =$ Yes and $c_2 =$ No. The larger value of $P(\text{Yes}|A)$ and $P(\text{No}|A)$ determines the predicted result of the output variable. Given the test data, use the following code to predict the *Enrolls*.

```
prob_yes <-
         ageCounts["Yes",testdata[,c("Age")]]*
         incomeCounts["Yes",testdata[,c("Income")]]*
         jsCounts["Yes",testdata[,c("JobSatisfaction")]]*
         desireCounts["Yes",testdata[,c("Desire")]]*
         tprior["Yes"]

prob_no <-
         ageCounts["No",testdata[,c("Age")]]*
         incomeCounts["No",testdata[,c("Income")]]*
         jsCounts["No",testdata[,c("JobSatisfaction")]]*
         desireCounts["No",testdata[,c("Desire")]]*
         tprior["No"]

max(prob_yes,prob_no)
```

As shown below, the predicted result of the test set is *Enrolls*=Yes.

```
prob_yes
       Yes
0.02821869

prob_no
         No
0.006857143

max(prob_yes, prob_no)
[1] 0.02821869
```

The *e1071* package in R has a built-in *naiveBayes* function that can compute the conditional probabilities of a categorical class variable given independent categorical predictor variables using the Bayes rule. The function takes the form of `naiveBayes(formula, data,...)`, where the arguments are defined as follows.

- **formula:** A formula of the form `class ~ x1 + x2 + ...` assuming x1, x2… are conditionally independent
- **data:** A data frame of factors

Use the following code snippet to execute the model and display the results.

```
model <- naiveBayes(Enrolls ~ Age+Income+JobSatisfaction+Desire,
                    traindata)
# display model
model
```

The output that follows shows that the probabilities of *model* match the probabilities from the previous method. The default `laplace=laplace` setting enables the Laplace smoothing.

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y                 No       Yes
0.0000000 0.3571429 0.6428571

Conditional probabilities:
     Age
Y          <=30       >40  31 to 40

  No  0.6000000 0.4000000 0.0000000
  Yes 0.2222222 0.3333333 0.4444444

     Income
Y          High       Low    Medium

  No  0.4000000 0.2000000 0.4000000
  Yes 0.2222222 0.3333333 0.4444444

     JobSatisfaction
Y            No       Yes

  No  0.8000000 0.2000000
  Yes 0.3333333 0.6666667
```

```
     Desire
Y     Excellent      Fair

  No  0.6000000 0.4000000
  Yes 0.3333333 0.6666667
```

Next, predicting the outcome of *Enrolls* with the *testdata* shows the result is *Enrolls*=Yes.

```
# predict with testdata
results <- predict (model,testdata)
# display results
results
[1] Yes
Levels:  No Yes
```

The *naiveBayes* function accepts a Laplace parameter that allows the customization of the ε value of Equation 7-17 for the Laplace smoothing. The code that follows shows how to build a naïve Bayes classifier with Laplace smoothing $\varepsilon = 0.01$ for prediction.

```
# use the NB classifier with Laplace smoothing
model1 = naiveBayes(Enrolls ~., traindata, laplace=.01)


# display model
model1
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
                 No       Yes
0.0000000 0.3571429 0.6428571

Conditional probabilities:
     Age
Y            <=30         >40    31 to 40
      0.333333333 0.333333333 0.333333333
  No  0.598409543 0.399602386 0.001988072
  Yes 0.222591362 0.333333333 0.444075305

     Income
Y          High       Low    Medium
      0.3333333 0.3333333 0.3333333
  No  0.3996024 0.2007952 0.3996024
  Yes 0.2225914 0.3333333 0.4440753
```

```
     JobSatisfaction
Y            No       Yes
      0.5000000 0.5000000
  No  0.7988048 0.2011952
  Yes 0.3337029 0.6662971

     Desire
Y     Excellent      Fair
      0.5000000 0.5000000
  No  0.5996016 0.4003984
  Yes 0.3337029 0.6662971
```

The test case is again classified as *Enrolls*=Yes.

```
# predict with testdata
results1 <- predict (model1,testdata)


# display results
results1
[1] Yes
Levels:  No Yes
```

7.3 Diagnostics of Classifiers

So far, this book has talked about three classifiers: logistic regression, decision trees, and naïve Bayes. These three methods can be used to classify instances into distinct groups according to the similar characteristics they share. Each of these classifiers faces the same issue: how to evaluate if they perform well.

A few tools have been designed to evaluate the performance of a classifier. Such tools are not limited to the three classifiers in this book but rather serve the purpose of assessing classifiers in general.

A ***confusion matrix*** is a specific table layout that allows visualization of the performance of a classifier.

Table 7-6 shows the confusion matrix for a two-class classifier. ***True positives*** (TP) are the number of positive instances the classifier correctly identified as positive. ***False positives*** (FP) are the number of instances in which the classifier identified as positive but in reality are negative. ***True negatives*** (TN) are the number of negative instances the classifier correctly identified as negative. ***False negatives*** (FN) are the number of instances classified as negative but in reality are positive. In a two-class classification, a preset threshold may be used to separate positives from negatives. TP and TN are the correct guesses. A good classifier should have large TP and TN and small (ideally zero) numbers for FP and FN.

TABLE 7-6 *Confusion Matrix*

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

In the bank marketing example, the training set includes 2,000 instances. An additional 100 instances are included as the testing set. Table 7-7 shows the confusion matrix of a naïve Bayes classifier on 100 clients to predict whether they would subscribe to the term deposit. Of the 11 clients who subscribed to the term deposit, the model predicted 3 subscribed and 8 not subscribed. Similarly, of the 89 clients who did not subscribe to the term, the model predicted 2 subscribed and 87 not subscribed. All correct guesses are located from top left to bottom right of the table. It's easy to visually inspect the table for errors, because they will be represented by any nonzero values outside the diagonal.

Table 7-7 *Confusion Matrix of Naïve Bayes from the Bank Marketing Example*

		Predicted Class		
		Subscribe	Not Subscribed	Total
Actual Class	Subscribed	3	8	11
	Not Subscribed	2	87	89
Total		5	95	100

The ***accuracy*** (or the ***overall success rate***) is a metric defining the rate at which a model has classified the records correctly. It is defined as the sum of TP and TN divided by the total number of instances, as shown in Equation 7-18.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{7-18}$$

A good model should have a high accuracy score, but having a high accuracy score alone does not guarantee the model is well established. The following measures can be introduced to better evaluate the performance of a classifier.

As seen in Chapter 6, the ***true positive rate*** (TPR) shows what percent of positive instances the classifier correctly identified. It's also illustrated in Equation 7-19.

$$TPR = \frac{TP}{TP + FN} \tag{7-19}$$

The ***false positive rate*** (FPR) shows what percent of negatives the classifier marked as positive. The FPR is also called the ***false alarm rate*** or the ***type I error rate*** and is shown in Equation 7-20.

$$FPR = \frac{FP}{FP + TN} \tag{7-20}$$

The ***false negative rate*** (FNR) shows what percent of positives the classifier marked as negatives. It is also known as the ***miss rate*** or ***type II error rate*** and is shown in Equation 7-21. Note that the sum of TPR and FNR is 1.

$$FNR = \frac{FN}{TP + FN} \tag{7-21}$$

A well-performed model should have a high TPR that is ideally 1 and a low FPR and FNR that are ideally 0. In reality, it's rare to have TPR = 1, FPR = 0, and FNR = 0, but these measures are useful to compare the performance of multiple models that are designed for solving the same problem. Note that in general, the model that is more preferable may depend on the business situation. During the discovery phase of the data analytics lifecycle, the team should have learned from the business what kind of errors can be tolerated. Some business situations are more tolerant of type I errors, whereas others may be more tolerant of type II errors. In some cases, a model with a TPR of 0.95 and an FPR of 0.3 is more acceptable than a model with a TPR of 0.9 and an FPR of 0.1 even if the second model is more accurate overall. Consider the case of e-mail spam filtering. Some people (such as busy executives) only want important e-mail in their inbox and are tolerant of having some less important e-mail end up in their spam folder as long as no spam is in their inbox. Other people may not want any important or less important e-mail to be specified as spam and are willing to have some spam in their inboxes as long as no important e-mail makes it into the spam folder.

Precision and recall are accuracy metrics used by the information retrieval community, but they can be used to characterize classifiers in general. ***Precision*** is the percentage of instances marked positive that really are positive, as shown in Equation 7-22.

$$Precision = \frac{TP}{TP + FP} \tag{7-22}$$

Recall is the percentage of positive instances that were correctly identified. Recall is equivalent to the TPR. Chapter 9, "Advanced Analytical Theory and Methods: Text Analysis," discusses how to use precision and recall for evaluation of classifiers in the context of text analysis.

Given the confusion matrix from Table 7-7, the metrics can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% = \frac{3 + 87}{3 + 87 + 2 + 8} \times 100\% = 90\%$$

$$TPR\ (or\ Recall) = \frac{TP}{TP + FN} = \frac{3}{3 + 8} \approx 0.273$$

$$FPR = \frac{FP}{FP + TN} = \frac{2}{2 + 87} \approx 0.022$$

$$FNR = \frac{FN}{TP + FN} = \frac{8}{3 + 8} \approx 0.727$$

$$Precision = \frac{TP}{TP + FP} = \frac{3}{3 + 2} = 0.6$$

These metrics show that for the bank marketing example, the naïve Bayes classifier performs well with accuracy and FPR measures and relatively well on precision. However, it performs poorly on TPR and FNR. To improve the performance, try to include more attributes in the datasets to better distinguish the characteristics of the records. There are other ways to evaluate the performance of a classifier in general, such as *N*-fold cross validation (Chapter 6) or bootstrap [14].

Chapter 6 has introduced the ***ROC curve,*** which is a common tool to evaluate classifiers. The abbreviation stands for ***receiver operating characteristic***, a term used in signal detection to characterize the trade-off between hit rate and false-alarm rate over a noisy channel. A ROC curve evaluates the performance of a classifier based on the TP and FP, regardless of other factors such as class distribution and error costs. The vertical axis is the True Positive Rate (TPR), and the horizontal axis is the False Positive Rate (FPR).

As seen in Chapter 6, any classifier can achieve the bottom left of the graph where TPR = FPR = 0 by classifying everything as negative. Similarly, any classifier can achieve the top right of the graph where TPR = FPR = 1 by classifying everything as positive. If a classifier performs "at chance" by random guessing the results, it can achieve any point on the diagonal line TPR=FPR by choosing an appropriate threshold of positive/negative. An ideal classifier should perfectly separate positives from negatives and thus achieve the top-left corner (TPR = 1, FPR = 0). The ROC curve of such classifiers goes straight up from TPR = FPR = 0 to the top-left corner and moves straight right to the top-right corner. In reality, it can be difficult to achieve the top-left corner. But a better classifier should be closer to the top left, separating it from other classifiers that are closer to the diagonal line.

Related to the ROC curve is the ***area under the curve*** (AUC). The AUC is calculated by measuring the area under the ROC curve. Higher AUC scores mean the classifier performs better. The score can range from 0.5 (for the diagonal line TPR=FPR) to 1.0 (with ROC passing through the top-left corner).

In the bank marketing example, the training set includes 2,000 instances. An additional 100 instances are included as the testing set. Figure 7-10 shows a ROC curve of the naïve Bayes classifier built on the training set of 2,000 instances and tested on the testing set of 100 instances. The figure is generated by the following R script. The `ROCR` package is required for plotting the ROC curve. The 2,000 instances are in a data frame called *`banktrain`*, and the additional 100 instances are in a data frame called *`banktest`*.

```
library(ROCR)

# training set
banktrain <- read.table("bank-sample.csv",header=TRUE,sep=",")
# drop a few columns
drops <- c("balance", "day", "campaign", "pdays", "previous", "month")
banktrain <- banktrain [,!(names(banktrain) %in% drops)]

# testing set
banktest <- read.table("bank-sample-test.csv",header=TRUE,sep=",")
banktest <- banktest [,!(names(banktest) %in% drops)]

# build the naïve Bayes classifier
nb_model <- naiveBayes(subscribed~.,
                       data=banktrain)
# perform on the testing set
nb_prediction <- predict(nb_model,
                         # remove column "subscribed"
                         banktest[,-ncol(banktest)],
                         type='raw')
score <- nb_prediction[, c("yes")]

actual_class <- banktest$subscribed == 'yes'

pred <- prediction(score, actual_class)
```

```
perf <- performance(pred, "tpr", "fpr")

plot(perf, lwd=2, xlab="False Positive Rate (FPR)",
     ylab="True Positive Rate (TPR)")
abline(a=0, b=1, col="gray50", lty=3)
```

The following R code shows that the corresponding AUC score of the ROC curve is about 0.915.

```
auc <- performance(pred, "auc")
auc <- unlist(slot(auc, "y.values"))
auc
[1] 0.9152196
```

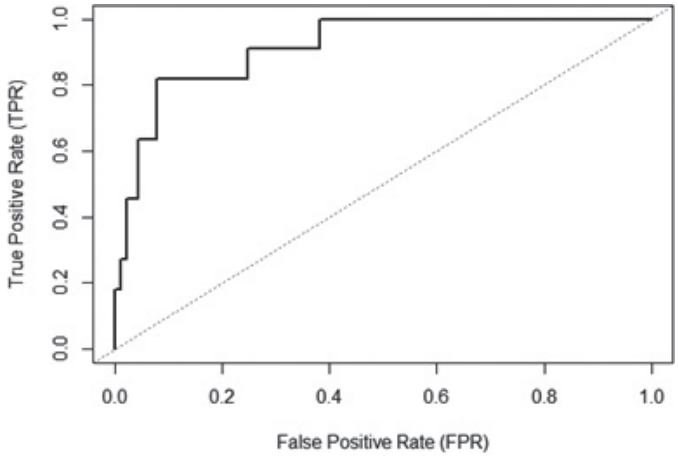

FIGURE 7-10 *ROC curve of the naïve Bayes classifier on the bank marketing dataset*

7.4 Additional Classification Methods

Besides the two classifiers introduced in this chapter, several other methods are commonly used for classification, including bagging [15], boosting [5], random forest [4], and support vector machines (SVM) [16]. Bagging, boosting, and random forest are all examples of ensemble methods that use multiple models to obtain better predictive performance than can be obtained from any of the constituent models.

Bagging (or bootstrap aggregating) [15] uses the bootstrap technique that repeatedly samples with replacement from a dataset according to a uniform probability distribution. "With replacement" means that when a sample is selected for a training or testing set, the sample is still kept in the dataset and may be selected again. Because the sampling is with replacement, some samples may appear several times in a training or testing set, whereas others may be absent. A model or base classifier is trained separately on each bootstrap sample, and a test sample is assigned to the class that received the highest number of votes.

Similar to bagging, boosting (or AdaBoost) [17] uses votes for classification to combine the output of individual models. In addition, it combines models of the same type. However, boosting is an iterative procedure

where a new model is influenced by the performances of those models built previously. Furthermore, boosting assigns a weight to each training sample that reflects its importance, and the weight may adaptively change at the end of each boosting round. Bagging and boosting have been shown to have better performances [5] than a decision tree.

Random forest [4] is a class of ensemble methods using decision tree classifiers. It is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. A special case of random forest uses bagging on decision trees, where samples are randomly chosen with replacement from the original training set.

SVM [16] is another common classification method that combines linear models with instance-based learning techniques. Support vector machines select a small number of critical boundary instances called support vectors from each class and build a linear decision function that separates them as widely as possible. SVM by default can efficiently perform linear classifications and can be configured to perform nonlinear classifications as well.

Summary

This chapter focused on two classification methods: decision trees and naïve Bayes. It discussed the theory behind these classifiers and used a bank marketing example to explain how the methods work in practice. These classifiers along with logistic regression (Chapter 6) are often used for the classification of data. As this book has discussed, each of these methods has its own advantages and disadvantages. How does one pick the most suitable method for a given classification problem? Table 7-8 offers a list of things to consider when selecting a classifier.

Table 7-8 *Choosing a Suitable Classifier*

Concerns	Recommended Method(s)
Output of the classification should include class probabilities in addition to the class labels.	Logistic regression, decision tree
Analysts want to gain an insight into how the variables affect the model.	Logistic regression, decision tree
The problem is high dimensional.	Naïve Bayes
Some of the input variables might be correlated.	Logistic regression, decision tree
Some of the input variables might be irrelevant.	Decision tree, naïve Bayes
The data contains categorical variables with a large number of levels.	Decision tree, naïve Bayes
The data contains mixed variable types.	Logistic regression, decision tree
There is nonlinear data or discontinuities in the input variables that would affect the output.	Decision tree

After the classification, one can use a few evaluation tools to measure how well a classifier has performed or compare the performances of multiple classifiers. These tools include confusion matrix, TPR, FPR, FNR, precision, recall, ROC curves, and AUC.

In addition to the decision trees and naïve Bayes, other methods are commonly used as classifiers. These methods include but are not limited to bagging, boosting, random forest, and SVM.

Exercises

1. For a binary classification, describe the possible values of entropy. On what conditions does entropy reach its minimum and maximum values?
2. In a decision tree, how does the algorithm pick the attributes for splitting?
3. John went to see the doctor about a severe headache. The doctor selected John at random to have a blood test for swine flu, which is suspected to affect 1 in 5,000 people in this country. The test is 99% accurate, in the sense that the probability of a false positive is 1%. The probability of a false negative is zero. John's test came back positive. What is the probability that John has swine flu?
4. Which classifier is considered computationally efficient for high-dimensional problems? Why?
5. A data science team is working on a classification problem in which the dataset contains many correlated variables, and most of them are categorical variables. Which classifier should the team consider using? Why?
6. A data science team is working on a classification problem in which the dataset contains many correlated variables, and most of them are continuous. The team wants the model to output the probabilities in addition to the class labels. Which classifier should the team consider using? Why?
7. Consider the following confusion matrix:

		Predicted Class		
		Good	Bad	Total
Actual Class	Good	671	29	300
	Bad	38	262	700
Total		709	291	1000

What are the true positive rate, false positive rate, and false negative rate?

Bibliography

[1] M. Thomas, B. Pang, and L. Lee, "Get Out the Vote: Determining Support or Opposition from Congressional Floor-Debate Transcripts," in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, 2006.

[2] M. Shouman, T. Turner, and R. Stocker, "Using Decision Tree for Diagnosing Heart Disease Patients," in *Australian Computer Society, Inc.,* Ballarat, Australia, in *Proceedings of the Ninth Australasian Data Mining Conference (AusDM '11).*

[3] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An Evaluation of Naïve Bayesian Anti-Spam Filtering," in *Proceedings of the Workshop on Machine Learning in the New Information Age,* Barcelona, Spain, 2000.

[4] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5–32, 2001.

[5] J. R. Quinlan, "Bagging, Boosting, and C4. 5," *AAAI/IAAI,* vol. 1, 1996.

[6] S. Moro, P. Cortez, and R. Laureano, "Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology," in *Proceedings of the European Simulation and Modelling Conference - ESM'2011,* Guimaraes, Portugal, 2011.

[7] J. R. Quinlan, "Induction of Decision Trees," *Machine Learning,* vol. 1, no. 1, pp. 81–106, 1986.

[8] J. R. Quinlan, *C4. 5: Programs for Machine Learning,* Morgan Kaufmann, 1993.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees,* Belmont, CA: Wadsworth International Group, 1984.

[10] T. M. Mitchell, "Decision Tree Learning," in *Machine Learning,* New York, NY, USA, McGraw-Hill, Inc., 1997, p. 68.

[11] C. Phua, V. C. S. Lee, S. Kate, and R. W. Gayler, "A Comprehensive Survey of Data Mining-Based Fraud Detection," *CoRR,* vol. abs/1009.6119, 2010.

[12] R. Bhowmik, "Detecting Auto Insurance Fraud by Data Mining Techniques," *Journal of Emerging Trends in Computing and Information Sciences,* vol. 2, no. 4, pp. 156–162, 2011.

[13] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification,* New York: Ellis Horwood, 1994.

[14] I. H. Witten, E. Frank, and M. A. Hall, "The Bootstrap," in *Data Mining,* Burlington, Massachusetts, Morgan Kaufmann, 2011, pp. 155–156.

[15] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, no. 2, pp. 123–140, 1996.

[16] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods,* Cambridge, United Kingdom: Cambridge university press, 2000.

[17] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences,* vol. 55, no. 1, pp. 119–139, 1997.

8

Advanced Analytical Theory and Methods: Time Series Analysis

Key Concepts

ACF
ARIMA
Autoregressive
Moving average
PACF
Stationary
Time series

This chapter examines the topic of time series analysis and its applications. Emphasis is placed on identifying the underlying structure of the time series and fitting an appropriate Autoregressive Integrated Moving Average (ARIMA) model.

8.1 Overview of Time Series Analysis

Time series analysis attempts to model the underlying structure of observations taken over time. A ***time series***, denoted $Y = a + bX$, is an ordered sequence of equally spaced values over time. For example, Figure 8-1 provides a plot of the monthly number of international airline passengers over a 12-year period.

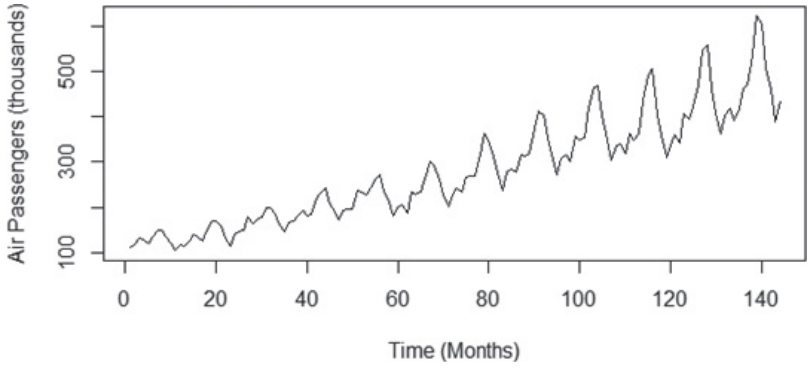

Figure 8-1 *Monthly international airline passengers*

In this example, the time series consists of an ordered sequence of 144 values. The analyses presented in this chapter are limited to equally spaced time series of one variable. Following are the goals of time series analysis:

- Identify and model the structure of the time series.
- Forecast future values in the time series.

Time series analysis has many applications in finance, economics, biology, engineering, retail, and manufacturing. Here are a few specific use cases:

- **Retail sales:** For various product lines, a clothing retailer is looking to forecast future monthly sales. These forecasts need to account for the seasonal aspects of the customer's purchasing decisions. For example, in the northern hemisphere, sweater sales are typically brisk in the fall season, and swimsuit sales are the highest during the late spring and early summer. Thus, an appropriate time series model needs to account for fluctuating demand over the calendar year.
- **Spare parts planning:** Companies' service organizations have to forecast future spare part demands to ensure an adequate supply of parts to repair customer products. Often the spares inventory consists of thousands of distinct part numbers. To forecast future demand, complex models for each part number can be built using input variables such as expected part failure rates, service diagnostic effectiveness, forecasted new product shipments, and forecasted trade-ins/decommissions.

However, time series analysis can provide accurate short-term forecasts based simply on prior spare part demand history.

- **Stock trading:** Some high-frequency stock traders utilize a technique called ***pairs trading***. In pairs trading, an identified strong positive correlation between the prices of two stocks is used to detect a market opportunity. Suppose the stock prices of Company A and Company B consistently move together. Time series analysis can be applied to the difference of these companies' stock prices over time. A statistically larger than expected price difference indicates that it is a good time to buy the stock of Company A and sell the stock of Company B, or vice versa. Of course, this trading approach depends on the ability to execute the trade quickly and be able to detect when the correlation in the stock prices is broken. Pairs trading is one of many techniques that falls into a trading strategy called ***statistical arbitrage***.

8.1.1 Box-Jenkins Methodology

In this chapter, a time series consists of an ordered sequence of equally spaced values over time. Examples of a time series are monthly unemployment rates, daily website visits, or stock prices every second. A time series can consist of the following components:

- Trend
- Seasonality
- Cyclic
- Random

The ***trend*** refers to the long-term movement in a time series. It indicates whether the observation values are increasing or decreasing over time. Examples of trends are a steady increase in sales month over month or an annual decline of fatalities due to car accidents.

The ***seasonality*** component describes the fixed, periodic fluctuation in the observations over time. As the name suggests, the seasonality component is often related to the calendar. For example, monthly retail sales can fluctuate over the year due to the weather and holidays.

A ***cyclic*** component also refers to a periodic fluctuation, but one that is not as fixed as in the case of a seasonality component. For example, retails sales are influenced by the general state of the economy. Thus, a retail sales time series can often follow the lengthy boom-bust cycles of the economy.

After accounting for the other three components, the ***random*** component is what remains. Although noise is certainly part of this random component, there is often some underlying structure to this random component that needs to be modeled to forecast future values of a given time series.

Developed by George Box and Gwilym Jenkins, the Box-Jenkins methodology for time series analysis involves the following three main steps:

1. Condition data and select a model.
   - Identify and account for any trends or seasonality in the time series.
   - Examine the remaining time series and determine a suitable model.
2. Estimate the model parameters.
3. Assess the model and return to Step 1, if necessary.

The primary focus of this chapter is to use the Box-Jenkins methodology to apply an ARIMA model to a given time series.

8.2 ARIMA Model

To fully explain an ARIMA (Autoregressive Integrated Moving Average) model, this section describes the model's various parts and how they are combined. As stated in the first step of the Box-Jenkins methodology, it is necessary to remove any trends or seasonality in the time series. This step is necessary to achieve a time series with certain properties to which autoregressive and moving average models can be applied. Such a time series is known as a stationary time series. A time series, y_t *for* $t = 1,2,3,\ldots$, is a ***stationary time series*** if the following three conditions are met:

- (a) The expected value (mean) of y_t is a constant for all values of t.
- (b) The variance of y_t is finite.
- (c) The covariance of y_t *and* y_{t+h} depends only on the value of $h = 0, 1, 2, \ldots$ for all t.

The covariance of y_t *and* y_{t+h} is a measure of how the two variables, y_t *and* y_{t+h}, vary together. It is expressed in Equation 8-1.

$$cov\left(y_t, y_{t+h}\right) = E[(y_t - \mu_t)(y_{t+h} - \mu_{t+h})] \tag{8-1}$$

If two variables are independent of each other, their covariance is zero. If the variables change together in the same direction, the variables have a positive covariance. Conversely, if the variables change together in the opposite direction, the variables have a negative covariance.

For a stationary time series, by condition (a), the mean is a constant, say μ. So, for a given stationary sequence, y_t, the covariance notation can be simplified to what's shown in Equation 8-2.

$$cov\left(h\right) = E\left[\left(y_t - \mu\right)\left(y_{t+h} - \mu\right)\right] \tag{8-2}$$

By part (c), the covariance between two points in the time series can be nonzero, as long as the value of the covariance is only a function of h. Equation 8-3 is an example for $h = 3$.

$$cov\,(3) = cov\,(y_1, y_4) = cov\,(y_2, y_5) = \ldots \tag{8-3}$$

It is important to note that for $h = 0$, the $cov\,(0) = cov(y_t, y_t) = var(y_t)$ for all t. Because the $var\,(y_t) < \infty$, by condition (b), the variance of y_t is a constant for all t. So the constant variance coupled with part (a), $E\,[y_t] = \mu$, for all t and some constant μ, suggests that a stationary time series can look like Figure 8-2. In this plot, the points appear to be centered about a fixed constant, zero, and the variance appears to be somewhat constant over time.

8.2.1 Autocorrelation Function (ACF)

Although there is not an overall trend in the time series plotted in Figure 8-2, it appears that each point is somewhat dependent on the past points. The difficulty is that the plot does not provide insight into the covariance of the variables in the time series and its underlying structure. The plot of ***autocorrelation function (ACF)*** provides this insight. For a stationary time series, the ACF is defined as shown in Equation 8-4.

$$ACF(h) = \frac{cov(y_t, y_{t+h})}{\sqrt{cov(y_t, y_t)\, cov(y_{t+h}, y_{t+h})}} = \frac{cov(h)}{cov(0)} \tag{8-4}$$

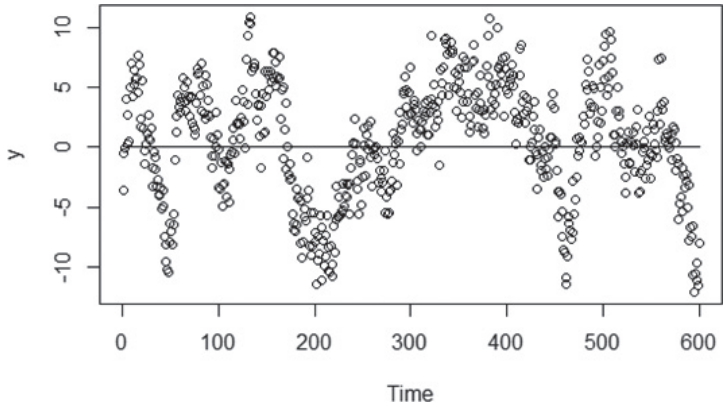

FIGURE 8-2 *A plot of a stationary series*

Because the cov(0) is the variance, the ACF is analogous to the correlation function of two variables, $corr(y_t, y_{t+h})$, and the value of the ACF falls between –1 and 1. Thus, the closer the absolute value of ACF(h) is to 1, the more useful y_t can be as a predictor of y_{t+h}.

Using the same dataset plotted in Figure 8-2, the plot of the ACF is provided in Figure 8-3.

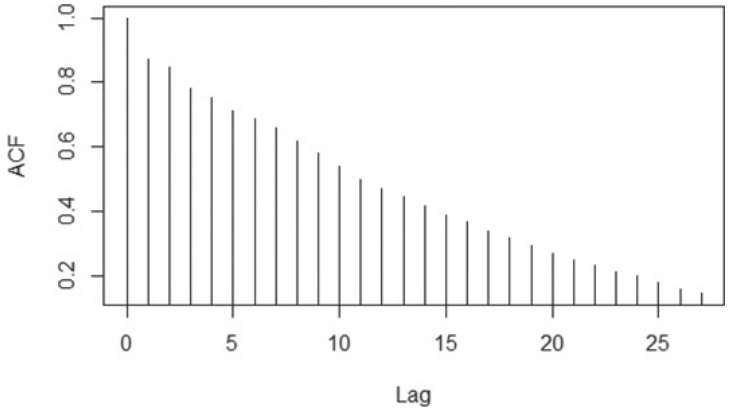

FIGURE 8-3 *Autocorrelation function (ACF)*

By convention, the quantity *h* in the ACF is referred to as the ***lag***, the difference between the time points *t* and *t* + *h*. At lag 0, the ACF provides the correlation of every point with itself. So ACF(0) always equals 1. According to the ACF plot, at lag 1 the correlation between y_t *and* y_{t-1} is approximately 0.9, which is very close to 1. So y_{t-1} appears to be a good predictor of the value of y_t. Because ACF(2) is around 0.8, y_{t-2} also appears to be a good predictor of the value of y_t. A similar argument could be made for lag 3 to lag 8. (All the autocorrelations are greater than 0.6.) In other words, a model can be considered that would express y_t as a linear sum of its previous 8 terms. Such a model is known as an autoregressive model of order 8.

8.2.2 Autoregressive Models

For a stationary time series, y_t $t = 1, 2, 3, \ldots$, an ***autoregressive model of order p***, denoted AR(p), is expressed as shown in Equation 8-5:

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t \tag{8-5}$$

where δ is a constant for a nonzero-centered time series:
ϕ_j is a constant for j = 1, 2, . . ., p
y_{t-j} is the value of the time series at time $t - j$
$\phi_p \neq 0$
$\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$ for all t

Thus, a particular point in the time series can be expressed as a linear combination of the prior p values, y_{t-j} for j = 1, 2, . . . p, of the time series plus a random error term, ε_t. In this definition, the ε_t time series is often called a ***white noise process*** and is used to represent random, independent fluctuations that are part of the time series.

From the earlier example in Figure 8-3, the autocorrelations are quite high for the first several lags. Although it appears that an AR(8) model might be a good candidate to consider for the given dataset, examining an AR(1) model provides further insight into the ACF and the appropriate value of p to choose. For an AR(1) model, centered around $\delta = 0$, Equation 8-5 simplifies to Equation 8-6.

$$y_t = \phi_1 y_{t-1} + \varepsilon_t \tag{8-6}$$

Based on Equation 8-6, it is evident that $y_{t-1} = \phi_1 y_{t-2} + \varepsilon_{t-1}$. Thus, substituting for y_{t-1} yields Equation 8-7.

$$\begin{aligned} y_t &= \phi_1(\phi y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t \\ &= \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \end{aligned} \tag{8-7}$$

Therefore, in a time series that follows an AR(1) model, considerable autocorrelation is expected at lag 2. As this substitution process is repeated, y_t can be expressed as a function of y_{t-h} for $h = 3, 4$ ...and a sum of the error terms. This observation means that even in the simple AR(1) model, there will be considerable autocorrelation with the larger lags even though those lags are not explicitly included in the model. What is needed is a measure of the autocorrelation between y_t and y_{t+h} for h = 1, 2, 3... with the effect of the y_{t+1} to y_{t+h-1} values excluded from the measure. The ***partial autocorrelation function (PACF)*** provides such a measure and is expressed as shown in Equation 8-8.

$$\begin{aligned} PACF(h) &= corr(y_t - y_t^*, y_{t+h} - y_{t+h}^*) \text{ for } h \geq 2 \\ &= corr(y_t, y_{t+1}) \qquad\qquad \text{for } h = 1 \end{aligned} \tag{8-8}$$

where $y_t^* = \beta_1 y_{t+1} + \beta_2 y_{t+2} \ldots + \beta_{h-1} y_{t+h-1}$,
$y_{t+h}^* = \beta_1 y_{t+h-1} + \beta_2 y_{t+h-2} \ldots + \beta_{h-1} y_{t+1}$, and
the h − 1 values of the βs are based on linear regression .

In other words, after linear regression is used to remove the effect of the variables between y_t and y_{t+h} on y_t and y_{t+h}, the PACF is the correlation of what remains. For $h = 1$, there are no variables between y_t and y_{t+1}. So the PACF(1) equals ACF(1). Although the computation of the PACF is somewhat complex, many software tools hide this complexity from the analyst.

For the earlier example, the PACF plot in Figure 8-4 illustrates that after lag 2, the value of the PACF is sharply reduced. Thus, after removing the effects of y_{t+1} and y_{t+2}, the partial correlation between y_t and y_{t+3} is relatively small. Similar observations can be made for h = 4, 5, …. Such a plot indicates that an AR(2) is a good candidate model for the time series plotted in Figure 8-2. In fact, the time series data for this example was randomly generated based on $y_t = 0.6y_{t-1} + 0.35y_{t-2} + \varepsilon_t$ where $\varepsilon_t \sim N(0, 4)$.

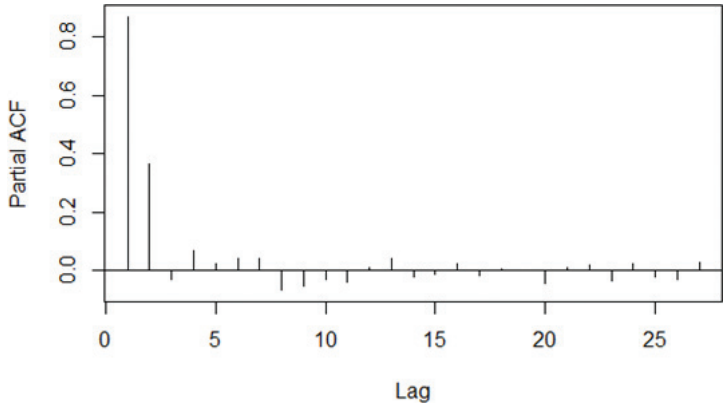

FIGURE 8-4 *Partial autocorrelation function (PACF) plot*

Because the ACF and PACF are based on correlations, negative and positive values are possible. Thus, the magnitudes of the functions at the various lags should be considered in terms of absolute values.

8.2.3 Moving Average Models

For a time series, y_t, centered at zero, a ***moving average model of order q***, denoted MA(q), is expressed as shown in Equation 8-9.

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{8-9}$$

where θ_k is a constant for k = 1, 2, ..., q
$\theta_q \neq 0$
$\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$ for all t

In an MA(q) model, the value of a time series is a linear combination of the current white noise term and the prior q white noise terms. So earlier random shocks directly affect the current value of the time series. For MA(q) models, the behavior of the ACF and PACF plots are somewhat swapped from the behavior of these plots for AR(p) models. For a simulated MA(3) time series of the form $y_t = \varepsilon_t - 0.4\,\varepsilon_{t-1} + 1.1\varepsilon_{t-2} - 2.5\,\varepsilon_{t-3}$ where $\varepsilon_t \sim N(0, 1)$, Figure 8-5 provides the scatterplot of the simulated data over time.

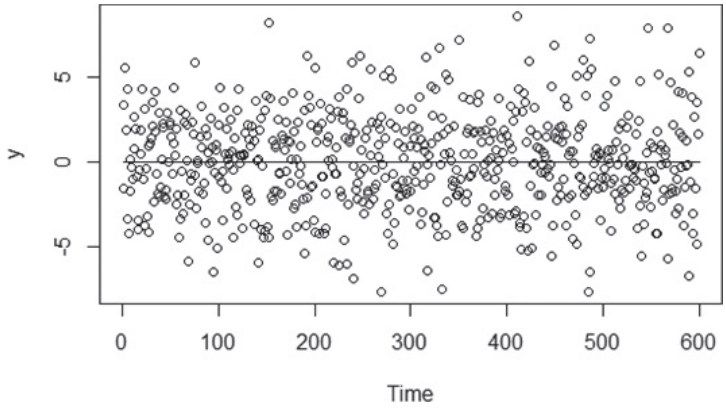

FIGURE 8-5 *Scatterplot of a simulated MA(3) time series*

Figure 8-6 provides the ACF plot for the simulated data. Again, the ACF(0) equals 1, because any variable is perfectly correlated with itself. At lags 1, 2, and 3, the value of the ACF is relatively large in absolute value compared to the subsequent terms. In an autoregressive model, the ACF slowly decays, but for an MA(3) model, the ACF somewhat abruptly cuts off after lag 3. In general, this pattern can be extended to any MA(q) model.

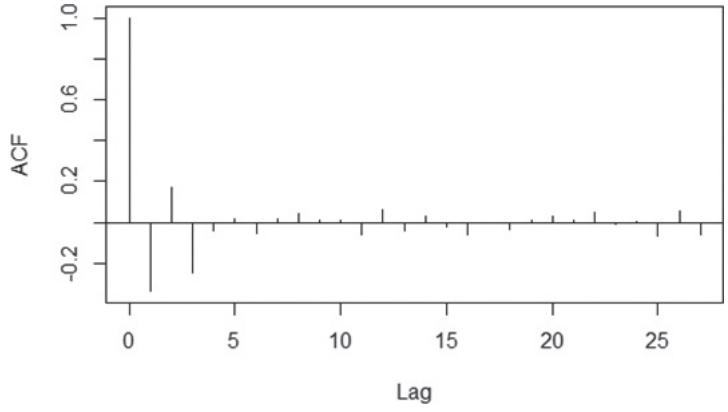

FIGURE 8-6 *ACF plot of a simulated MA(3) time series*

To understand why this phenomenon occurs, it is useful to examine Equations 8-10 through 8-14 for an MA(3) time series model:

$$y_t = \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \theta_3\varepsilon_{t-3} \quad (8\text{-}10)$$

$$y_{t-1} = \varepsilon_{t-1} + \theta_1\varepsilon_{t-2} + \theta_2\varepsilon_{t-3} + \theta_3\varepsilon_{t-4} \quad (8\text{-}11)$$

$$y_{t-2} = \varepsilon_{t-2} + \theta_1\varepsilon_{t-3} + \theta_2\varepsilon_{t-4} + \theta_3\varepsilon_t \quad (8\text{-}12)$$

$$y_{t-3} = \varepsilon_{t-3} + \theta_1\varepsilon_{t-4} + \theta_2\varepsilon_{t-5} + \theta_3\varepsilon \quad (8\text{-}13)$$

$$y_{t-4} = \varepsilon_{t-4} + \theta_1\varepsilon_{t-5} + \theta_2\varepsilon_{t-6} + \theta_3 \quad (8\text{-}14)$$

Because the expression of y_t shares specific white noise variables with the expressions for y_{t-1} through y_{t-3}, inclusive, those three variables are correlated to y_t. However, the expression of y_t in Equation 8-10 does not share white noise variables with y_{t-4} in Equation 8-14. So the theoretical correlation between y_t and y_{t-4} is zero. Of course, when dealing with a particular dataset, the theoretical autocorrelations are unknown, but the observed autocorrelations should be close to zero for lags greater than q when working with an MA(q) model.

8.2.4 ARMA and ARIMA Models

In general, the data scientist does not have to choose between an AR(p) and an MA(q) model to describe a time series. In fact, it is often useful to combine these two representations into one model. The combination of these two models for a stationary time series results in an ***Autoregressive Moving Average model, ARMA(p,q)***, which is expressed as shown in Equation 8-15.

$$y_t = \delta + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t + \theta_1\varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \quad (8\text{-}15)$$

where δ is a constant for a nonzero-centered time series

ϕ_j is a constant for j = 1, 2, ..., p

$\phi_p \neq 0$

θ_k is a constant for k = 1, 2, ..., q

$\theta_q \neq 0$

$\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$ for all t

If $p = 0$ and $q \neq 0$, then the ARMA(p,q) model is simply an AR(p) model. Similarly, if $p = 0$ and $q \neq 0$, then the ARMA(p,q) model is an MA(q) model.

To apply an ARMA model properly, the time series must be a stationary one. However, many time series exhibit some trend over time. Figure 8-7 illustrates a time series with an increasing linear trend over time. Since such a time series does not meet the requirement of a constant expected value (mean), the data needs to be adjusted to remove the trend. One transformation option is to perform a regression analysis on the time series and then to subtract the value of the fitted regression line from each observed y-value.

If detrending using a linear or higher order regression model does not provide a stationary series, a second option is to compute the difference between successive y-values. This is known as ***differencing***. In other words, for the n values in a given time series compute the differences as shown in Equation 8-16.

$$d_t = y_t - y_{t-1} \quad \text{for t} = 2,3,\ldots,\text{n} \quad (8\text{-}16)$$

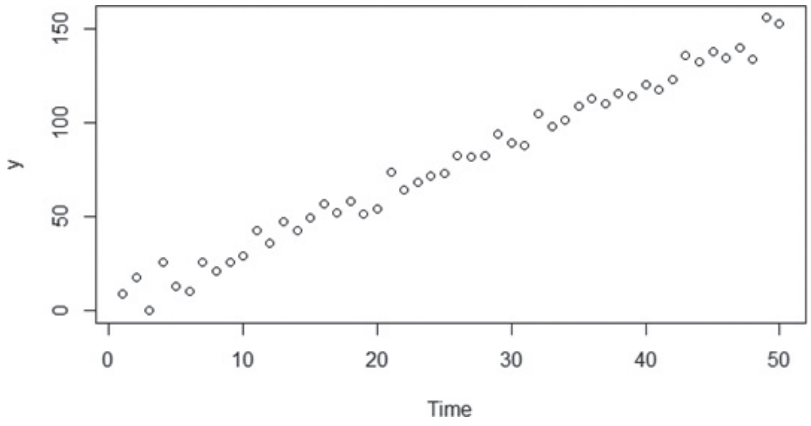

FIGURE 8-7 *A time series with a trend*

The mean of the time series plotted in Figure 8-8 is certainly not a constant. Applying differencing to the time series results in the plot in Figure 8-9. This plot illustrates a time series with a constant mean and a fairly constant variance over time.

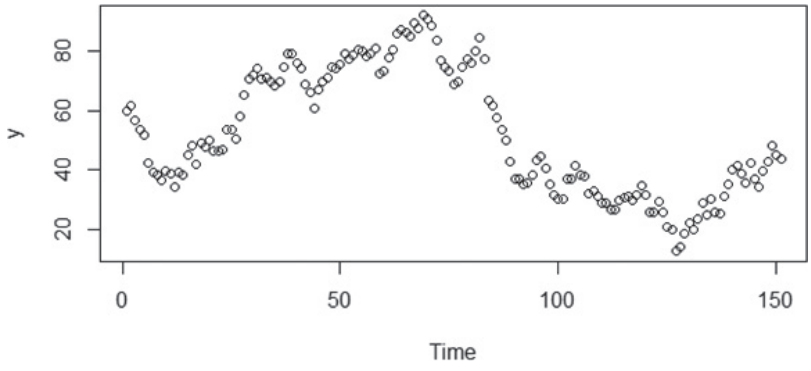

FIGURE 8-8 *Time series for differencing example*

If the differenced series is not reasonably stationary, applying differencing additional times may help. Equation 8-17 provides the twice differenced time series for $t = 3, 4, \ldots n$.

$$\begin{aligned} d_{t-1} - d_{t-2} &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned} \tag{8-17}$$

Successive differencing can be applied, but over-differencing should be avoided. One reason is that over-differencing may unnecessarily increase the variance. The increased variance can be detected by plotting the possibly over-differenced values and observing that the spread of the values is much larger, as seen in Figure 8-10 after differencing the values of y twice.

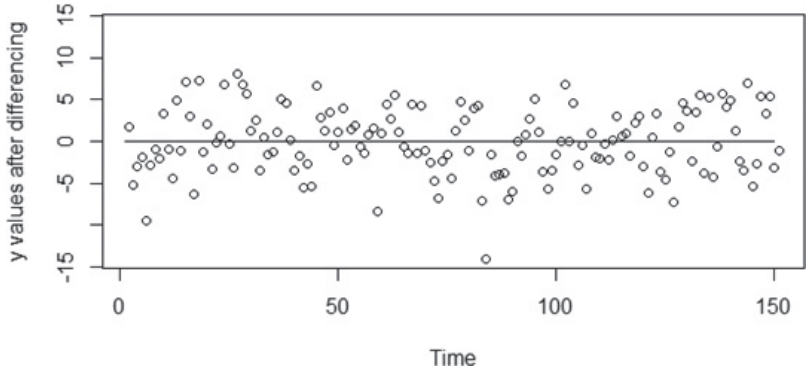

FIGURE 8-9 *Detrended time series using differencing*

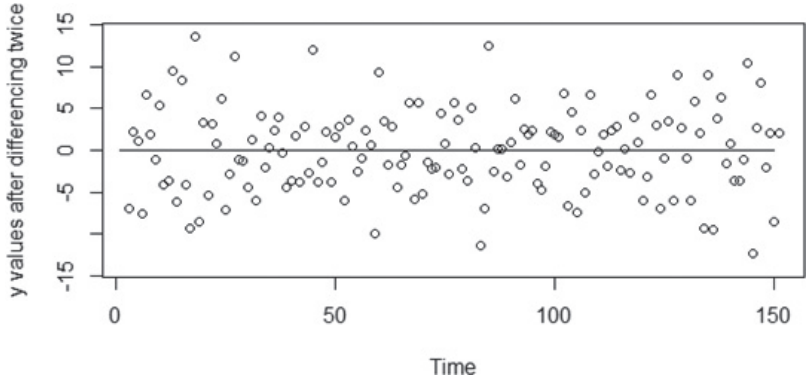

FIGURE 8-10 *Twice differenced series*

Because the need to make a time series stationary is common, the differencing can be included (integrated) into the ARMA model definition by defining the ***Autoregressive Integrated Moving Average*** model, denoted ARIMA(p,d,q). The structure of the ARIMA model is identical to the expression in Equation 8-15, but the ARMA(p,q) model is applied to the time series, y_t, after applying differencing d times.

Additionally, it is often necessary to account for seasonal patterns in time series. For example, in the retail sales use case example in Section 8.1, monthly clothing sales track closely with the calendar month. Similar to the earlier option of detrending a series by first applying linear regression, the seasonal pattern could be determined and the time series appropriately adjusted. An alternative is to use a ***seasonal autoregressive integrated moving average model***, denoted ARIMA(p,d,q) $\times$ $(P,D,Q)_s$ where:

- p, d, and q are the same as defined previously.
- s denotes the seasonal period.
- P is the number of terms in the AR model across the s periods.
- D is the number of differences applied across the s periods.
- Q is the number of terms in the MA model across the s periods.

For a time series with a seasonal pattern, following are typical values of s:

- 52 for weekly data
- 12 for monthly data
- 7 for daily data

The next section presents a seasonal ARIMA example and describes several techniques and approaches to identify the appropriate model and forecast the future.

8.2.5 Building and Evaluating an ARIMA Model

For a large country, the monthly gasoline production measured in millions of barrels has been obtained for the past 240 months (20 years). A market research firm requires some short-term gasoline production forecasts to assess the petroleum industry's ability to deliver future gasoline supplies and the effect on gasoline prices.

```
library(forecast)

# read in gasoline production time series
# monthly gas production expressed in millions of barrels
gas_prod_input <- as.data.frame( read.csv("c:/data/gas_prod.csv") )

# create a time series object
gas_prod <- ts(gas_prod_input[,2])

#examine the time series
plot(gas_prod, xlab = "Time (months)",
     ylab = "Gasoline production (millions of barrels)")
```

Using R, the dataset is plotted in Figure 8-11.

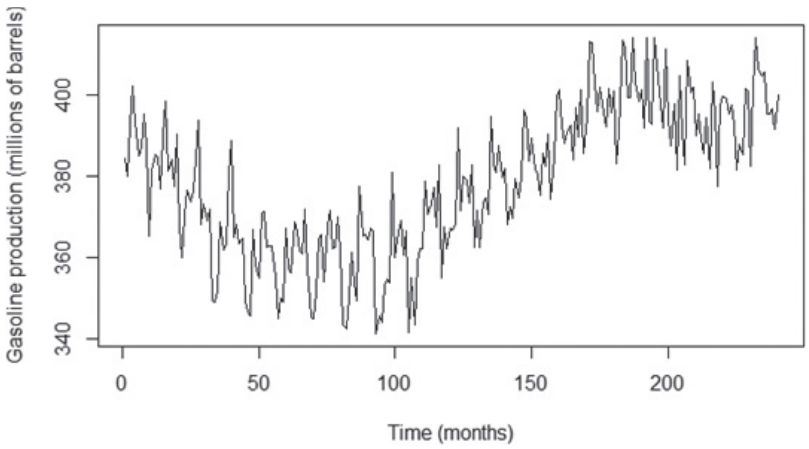

FIGURE 8-11 *Monthly gasoline production*

In R, the `ts()` function creates a time series object from a vector or a matrix. The use of time series objects in R simplifies the analysis by providing several methods that are tailored specifically for handling equally time spaced data series. For example, the `plot()` function does not require an explicitly specified variable for the x-axis.

To apply an ARMA model, the dataset needs to be a stationary time series. Using the `diff()` function, the gasoline production time series is differenced once and plotted in Figure 8-12.

```
plot(diff(gas_prod))
abline(a=0, b=0)
```

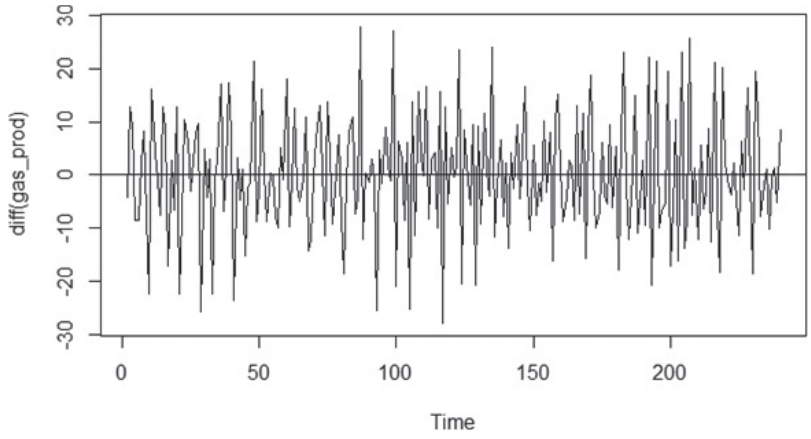

Figure 8-12 *Differenced gasoline production time series*

The differenced time series has a constant mean near zero with a fairly constant variance over time. Thus, a stationary time series has been obtained. Using the following R code, the ACF and PACF plots for the differenced series are provided in Figures 8-13 and 8-14, respectively.

```
# examine ACF and PACF of differenced series
acf(diff(gas_prod), xaxp = c(0, 48, 4), lag.max=48, main="")
pacf(diff(gas_prod), xaxp = c(0, 48, 4), lag.max=48, main="")
```

The dashed lines provide upper and lower bounds at a 95% significance level. Any value of the ACF or PACF outside of these bounds indicates that the value is significantly different from zero.

Figure 8-13 shows several significant ACF values. The slowly decaying ACF values at lags 12, 24, 36, and 48 are of particular interest. A similar behavior in the ACF was seen in Figure 8-3, but for lags 1, 2, 3,… Figure 8-13 indicates a seasonal autoregressive pattern every 12 months. Examining the PACF plot in Figure 8-14, the PACF value at lag 12 is quite large, but the PACF values are close to zero at lags 24, 36, and 48. Thus, a seasonal AR(1) model with period = 12 will be considered. It is often useful to address the seasonal portion of the overall ARMA model before addressing the nonseasonal portion of the model.

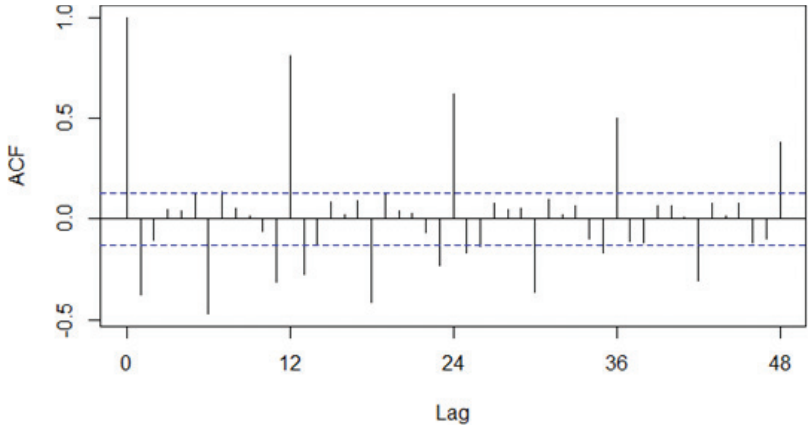

FIGURE 8-13 *ACF of the differenced gasoline time series*

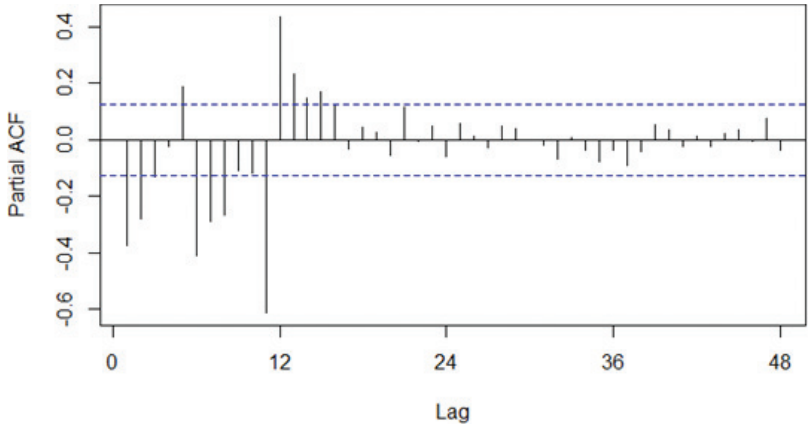

FIGURE 8-14 *PACF of the differenced gasoline time series*

The `arima()` function in R is used to fit a $(0,1,0) \times (1,0,0)_{12}$ model. The analysis is applied to the original time series variable, *`gas_prod`*. The differencing, d = 1, is specified by the order = c(0,1,0) term.

```
arima_1 <- arima (gas_prod,
                  order=c(0,1,0),
                  seasonal = list(order=c(1,0,0),period=12))
arima_1
```

```
Series: gas_prod
ARIMA(0,1,0)(1,0,0)[12]

Coefficients:
        sar1
      0.8335
```

```
s.e.  0.0324

sigma^2 estimated as 37.29:  log likelihood=-778.69
AIC=1561.38   AICc=1561.43   BIC=1568.33
```

The value of the coefficient for the seasonal AR(1) model is estimated to be 0.8335 with a standard error of 0.0324. Because the estimate is several standard errors away from zero, this coefficient is considered significant. The output from this first pass ARIMA analysis is stored in the variable `arima_1`, which contains several useful quantities including the residuals. The next step is to examine the residuals from fitting the $(0,1,0) \times (1,0,0)_{12}$ ARIMA model. The ACF and PACF plots of the residuals are provided in Figures 8-15 and 8-16, respectively.

```
# examine ACF and PACF of the (0,1,0)x(1,0,0)12 residuals
acf(arima_1$residuals, xaxp = c(0, 48, 4), lag.max=48, main="")
pacf(arima_1$residuals, xaxp = c(0, 48, 4), lag.max=48, main="")
```

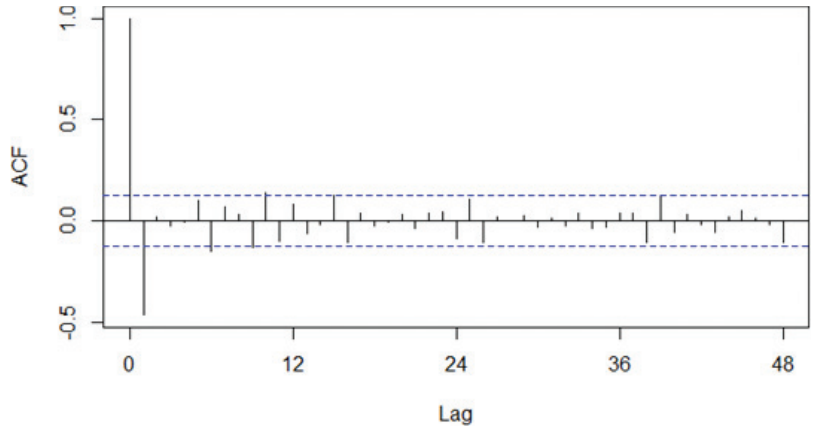

FIGURE 8-15 *ACF of residuals from seasonal AR(1) model*

The ACF plot of the residuals in Figure 8-15 indicates that the autoregressive behavior at lags 12, 24, 26, and 48 has been addressed by the seasonal AR(1) term. The only remaining ACF value of any significance occurs at lag 1. In Figure 8-16, there are several significant PACF values at lags 1, 2, 3, and 4.

Because the PACF plot in Figure 8-16 exhibits a slowly decaying PACF, and the ACF cuts off sharply at lag 1, an MA(1) model should be considered for the nonseasonal portion of the ARMA model on the differenced series. In other words, a $(0,1,1) \times (1,0,0)_{12}$ ARIMA model will be fitted to the original gasoline production time series.

```
arima_2 <- arima (gas_prod,
                  order=c(0,1,1),
                  seasonal = list(order=c(1,0,0),period=12))
arima_2

Series: gas_prod
ARIMA(0,1,1)(1,0,0)[12]

Coefficients:
          ma1     sar1
```

```
      -0.7065  0.8566
s.e.   0.0526  0.0298

sigma^2 estimated as 25.24:  log likelihood=-733.22
AIC=1472.43   AICc=1472.53   BIC=1482.86

acf(arima_2$residuals, xaxp = c(0, 48, 4), lag.max=48, main="")
pacf(arima_2$residuals, xaxp = c(0, 48,4), lag.max=48, main="")
```

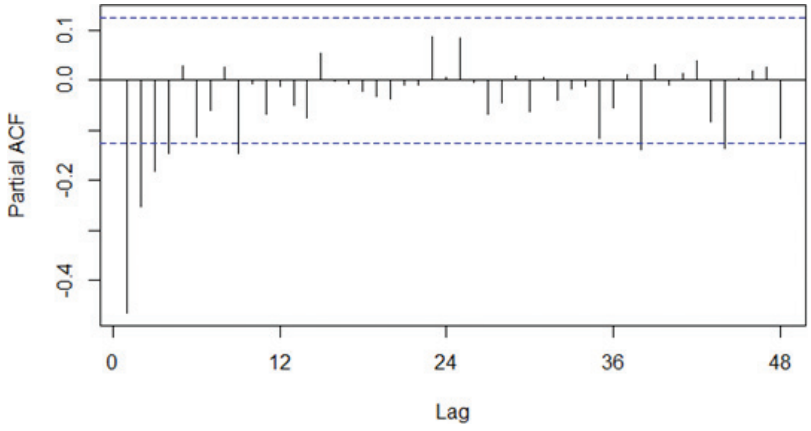

FIGURE 8-16 *PACF of residuals from seasonal AR(1) model*

Based on the standard errors associated with each coefficient estimate, the coefficients are significantly different from zero. In Figures 8-17 and 8-18, the respective ACF and PACF plots for the residuals from the second pass ARIMA model indicate that no further terms need to be considered in the ARIMA model.

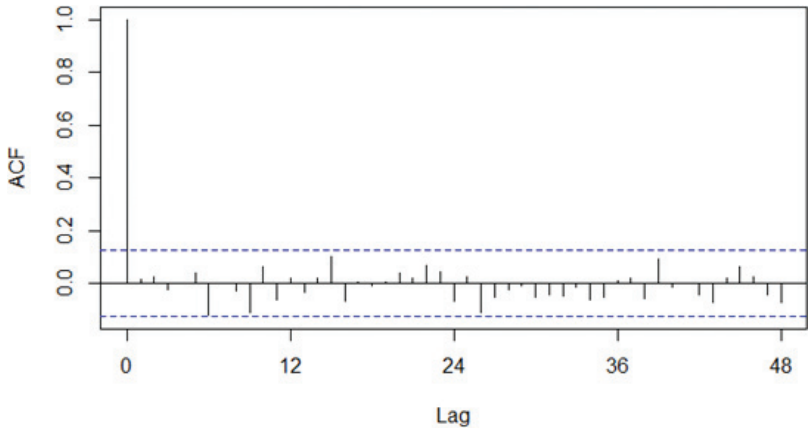

FIGURE 8-17 *ACF for the residuals from the $(0,1,1) \times (1,0,0)_{12}$ model*

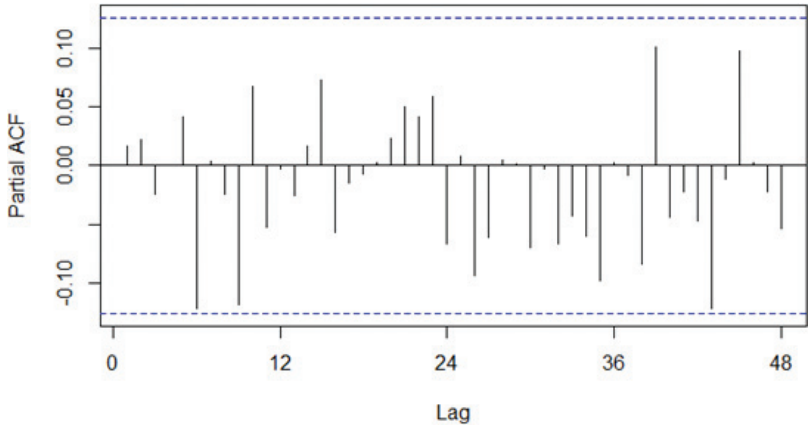

FIGURE 8-18 *PACF for the residuals from the* $(0,1,1)\times(1,0,0)_{12}$ *model*

It should be noted that the ACF and PACF plots each have several points that are close to the bounds at a 95% significance level. However, these points occur at relatively large lags. To avoid overfitting the model, these values are attributed to random chance. So no attempt is made to include these lags in the model. However, it is advisable to compare a reasonably fitting model to slight variations of that model.

Comparing Fitted Time Series Models

The `arima()` function in R uses Maximum Likelihood Estimation (MLE) to estimate the model coefficients. In the R output for an ARIMA model, the log-likelihood ($\log L$) value is provided. The values of the model coefficients are determined such that the value of the log likelihood function is maximized. Based on the $\log L$ value, the R output provides several measures that are useful for comparing the appropriateness of one fitted model against another fitted model. These measures follow:

- AIC (Akaike Information Criterion)
- AICc (Akaike Information Criterion, corrected)
- BIC (Bayesian Information Criterion)

Because these criteria impose a penalty based on the number of parameters included in the models, the preferred model is the fitted model with the smallest AIC, AICc, or BIC value. Table 8-1 provides the information criteria measures for the ARIMA models already fitted as well as a few additional fitted models. The highlighted row corresponds to the fitted ARIMA model obtained previously by examining the ACF and PACF plots.

TABLE 8-1 *Information Criteria to Measure Goodness of Fit*

ARIMA Model $(p,d,q) \times (P,Q,D)_s$	AIC	AICc	BIC
$(0,1,0) \times (1,0,0)_{12}$	1561.38	1561.43	1568.33
$(0,1,1) \times (1,0,0)_{12}$	1472.43	1472.53	1482.86
$(0,1,2) \times (1,0,0)_{12}$	1474.25	1474.42	1488.16
$(1,1,0) \times (1,0,0)_{12}$	1504.29	1504.39	1514.72
$(1,1,1) \times (1,0,0)_{12}$	1474.22	1474.39	1488.12

In this dataset, the $(0,1,1) \times (1,0,0)_{12}$ model does have the lowest AIC, AICc, and BIC values compared to the same criterion measures for the other ARIMA models.

Normality and Constant Variance

The last model validation step is to examine the normality assumption of the residuals in Equation 8-15. Figure 8-19 indicates residuals with a mean near zero and a constant variance over time. The histogram in Figure 8-20 and the Q-Q plot in Figure 8-21 support the assumption that the error terms are normally distributed. Q-Q plots were presented in Chapter 6, "Advanced Analytical Theory and Methods: Regression."

```
plot(arima_2$residuals, ylab = "Residuals")
abline(a=0, b=0)

hist(arima_2$residuals, xlab="Residuals", xlim=c(-20,20))

qqnorm(arima_2$residuals, main="")
qqline(arima_2$residuals)
```

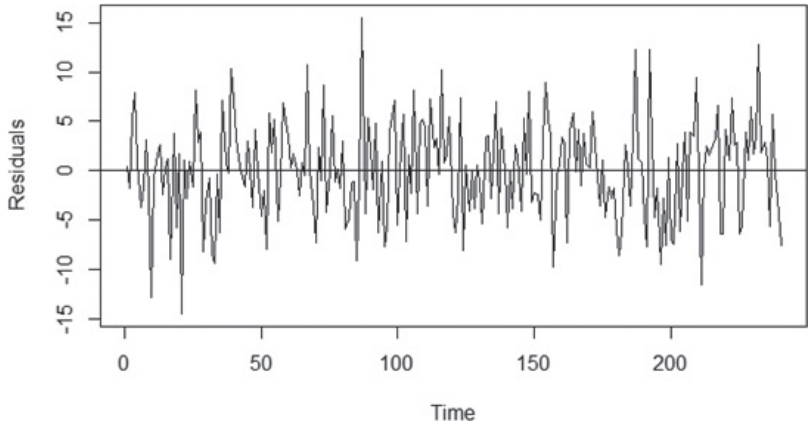

FIGURE 8-19 *Plot of residuals from the fitted $(0,1,1) \times (1,0,0)_{12}$ model*

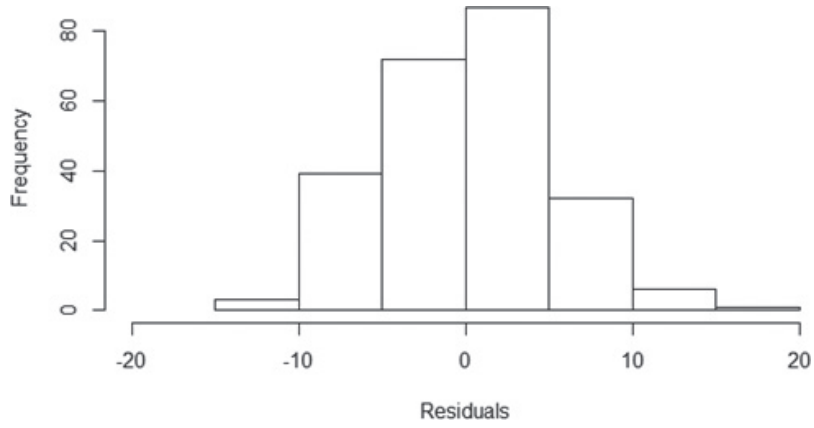

FIGURE 8-20 *Histogram of the residuals from the fitted* $(0,1,1)\times(1,0,0)_{12}$ *model*

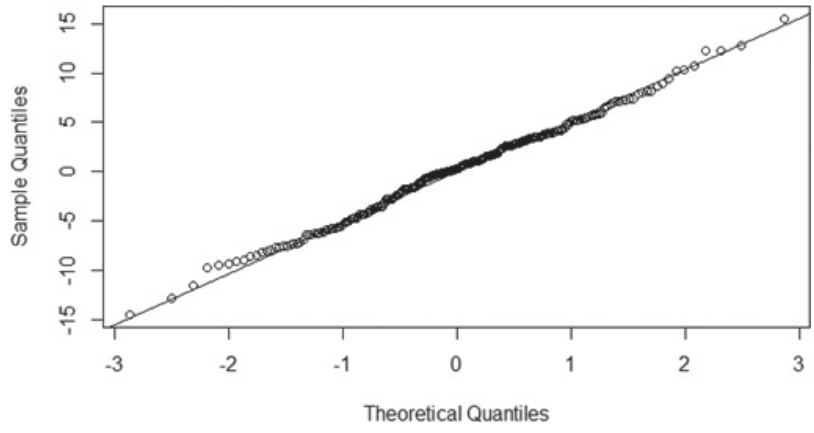

FIGURE 8-21 *Q-Q plot of the residuals from the fitted* $(0,1,1)\times(1,0,0)_{12}$ *model*

If the normality or the constant variance assumptions do not appear to be true, it may be necessary to transform the time series prior to fitting the ARIMA model. A common transformation is to apply a logarithm function.

Forecasting

The next step is to use the fitted $(0,1,1)\times(1,0,0)_{12}$ model to forecast the next 12 months of gasoline production. In R, the forecasts are easily obtained using the `predict()` function and the fitted model already stored in the variable *`arima_2`*. The predicted values along with the associated upper and lower bounds at a 95% confidence level are displayed in R and plotted in Figure 8-22.

```
#predict the next 12 months
arima_2.predict <- predict(arima_2,n.ahead=12)

matrix(c(arima_2.predict$pred-1.96*arima_2.predict$se,
```

```
          arima_2.predict$pred,
          arima_2.predict$pred+1.96*arima_2.predict$se), 12,3,
          dimnames=list( c(241:252) ,c("LB","Pred","UB"))  )

           LB     Pred       UB
241 394.9689 404.8167 414.6645
242 378.6142 388.8773 399.1404
243 394.9943 405.6566 416.3189
244 405.0188 416.0658 427.1128
245 397.9545 409.3733 420.7922
246 396.1202 407.8991 419.6780
247 396.6028 408.7311 420.8594
248 387.5241 399.9920 412.4598
249 387.1523 399.9507 412.7492
250 387.8486 400.9693 414.0900
251 383.1724 396.6076 410.0428
252 390.2075 403.9500 417.6926
plot(gas_prod, xlim=c(145,252),
     xlab = "Time (months)",
     ylab = "Gasoline production (millions of barrels)",
     ylim=c(360,440))
lines(arima_2.predict$pred)
lines(arima_2.predict$pred+1.96*arima_2.predict$se, col=4, lty=2)
lines(arima_2.predict$pred-1.96*arima_2.predict$se, col=4, lty=2)
```

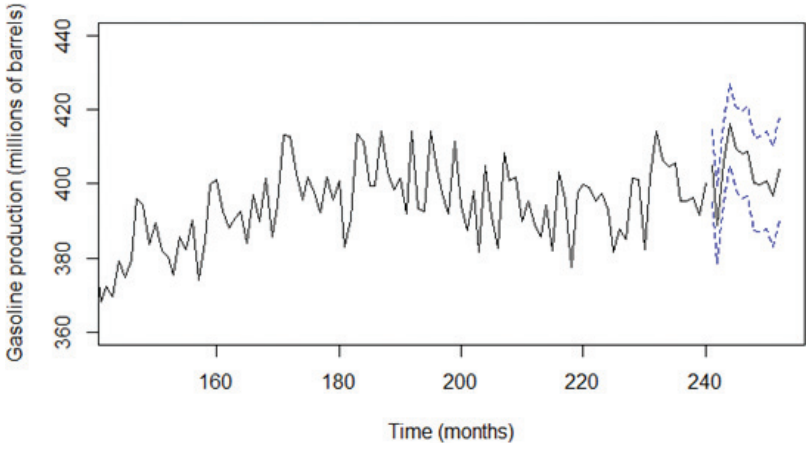

FIGURE 8-22 *Actual and forecasted gasoline production*

8.2.6 Reasons to Choose and Cautions

One advantage of ARIMA modeling is that the analysis can be based simply on historical time series data for the variable of interest. As observed in the chapter about regression (Chapter 6), various input variables

need to be considered and evaluated for inclusion in the regression model for the outcome variable. Because ARIMA modeling, in general, ignores any additional input variables, the forecasting process is simplified. If regression analysis was used to model gasoline production, input variables such as Gross Domestic Product (GDP), oil prices, and unemployment rate may be useful input variables. However, to forecast the gasoline production using regression, predictions are required for the GDP, oil price, and unemployment rate input variables.

The minimal data requirement also leads to a disadvantage of ARIMA modeling; the model does not provide an indication of what underlying variables affect the outcome. For example, if ARIMA modeling was used to forecast future retail sales, the fitted model would not provide an indication of what could be done to increase sales. In other words, causal inferences cannot be drawn from the fitted ARIMA model.

One caution in using time series analysis is the impact of severe shocks to the system. In the gas production example, shocks might include refinery fires, international incidents, or weather-related impacts such as hurricanes. Such events can lead to short-term drops in production, followed by persistently high increases in production to compensate for the lost production or to simply capitalize on any price increases.

Along similar lines of reasoning, time series analysis should only be used for short-term forecasts. Over time, gasoline production volumes may be affected by changing consumer demands as a result of more fuel-efficient gasoline-powered vehicles, electric vehicles, or the introduction of natural gas–powered vehicles. Changing market dynamics in addition to shocks will make any long-term forecasts, several years into the future, very questionable.

8.3 Additional Methods

Additional time series methods include the following:

- **Autoregressive Moving Average with Exogenous inputs (ARMAX)** is used to analyze a time series that is dependent on another time series. For example, retail demand for products can be modeled based on the previous demand combined with a weather-related time series such as temperature or rainfall.
- **Spectral analysis** is commonly used for signal processing and other engineering applications. Speech recognition software uses such techniques to separate the signal for the spoken words from the overall signal that may include some noise.
- **Generalized Autoregressive Conditionally Heteroscedastic (GARCH)** is a useful model for addressing time series with nonconstant variance or volatility. GARCH is used for modeling stock market activity and price fluctuations.
- **Kalman filtering** is useful for analyzing real-time inputs about a system that can exist in certain states. Typically, there is an underlying model of how the various components of the system interact and affect each other. A Kalman filter processes the various inputs, attempts to identify the errors in the input, and predicts the current state. For example, a Kalman filter in a vehicle navigation system can process various inputs, such as speed and direction, and update the estimate of the current location.
- **Multivariate time series analysis** examines multiple time series and their effect on each other. Vector ARIMA (VARIMA) extends ARIMA by considering a vector of several time series at a particular time, t. VARIMA can be used in marketing analyses that examine the time series related to a company's price and sales volume as well as related time series for the competitors.

Summary

This chapter presented time series analysis using ARIMA models. Time series analysis is different from other statistical techniques in the sense that most statistical analyses assume the observations are independent of each other. Time series analysis implicitly addresses the case in which any particular observation is somewhat dependent on prior observations.

Using differencing, ARIMA models allow nonstationary series to be transformed into stationary series to which seasonal and nonseasonal ARMA models can be applied. The importance of using the ACF and PACF plots to evaluate the autocorrelations was illustrated in determining ARIMA models to consider fitting. Akaike and Bayesian Information Criteria can be used to compare one fitted ARIMA model against another. Once an appropriate model has been determined, future values in the time series can be forecasted.

Exercises

1. Why use autocorrelation instead of autocovariance when examining stationary time series?
2. Provide an example that if the cov(X, Y) = 0, the two random variables, X and Y, are not necessarily independent.
3. Fit an appropriate ARIMA model on the following datasets included in R. Provide supporting evidence on why the fitted model was selected, and forecast the time series for 12 time periods ahead.
   a. **faithful:** Waiting times (in minutes) between Old Faithful geyser eruptions
   b. **JohnsonJohnson:** Quarterly earnings per J&J share
   c. **sunspot.month:** Monthly sunspot activity from 1749 to 1997
4. When should an ARIMA(p,d,q) model in which $d > 0$ be considered instead of an ARMA(p,q) model?

9

Advanced Analytical Theory and Methods: Text Analysis

Key Concepts

Term
Corpus
Text normalization
TFIDF
Topic modeling
Sentiment analysis

Text analysis, sometimes called text analytics, refers to the representation, processing, and modeling of textual data to derive useful insights. An important component of text analysis is text mining, the process of discovering relationships and interesting patterns in large text collections.

Text analysis suffers from the curse of high dimensionality. Take the popular children's book *Green Eggs and Ham* [1] as an example. Author Theodor Geisel (Dr. Seuss) was challenged to write an entire book with just 50 distinct words. He responded with the book *Green Eggs and Ham*, which contains 804 total words, only 50 of them distinct. These 50 words are:

> a, am, and, anywhere, are, be, boat, box, car, could, dark, do, eat, eggs, fox, goat, good, green, ham, here, house, I, if, in, let, like, may, me, mouse, not, on, or, rain, Sam, say, see, so, thank, that, the, them, there, they, train, tree, try, will, with, would, you

There's a substantial amount of repetition in the book. Yet, as repetitive as the book is, modeling it as a vector of counts, or features, for each distinct word still results in a 50-dimension problem.

Green Eggs and Ham is a simple book. Text analysis often deals with textual data that is far more complex. A ***corpus*** (plural: corpora) is a large collection of texts used for various purposes in Natural Language Processing (NLP). Table 9-1 lists a few example corpora that are commonly used in NLP research.

TABLE 9-1 *Example Corpora in Natural Language Processing*

Corpus	Word Count	Domain	Website
Shakespeare	0.88 million	Written	`http://shakespeare.mit.edu/`
Brown Corpus	1 million	Written	http://icame.uib.no/brown/bcm.html
Penn Treebank	1 million	Newswire	`http://www.cis.upenn.edu/~treebank/`
Switchboard Phone Conversations	3 million	Spoken	http://catalog.ldc.upenn.edu/LDC97S62
British National Corpus	100 million	Written and spoken	`http://www.natcorp.ox.ac.uk/`
NA News Corpus	350 million	Newswire	http://catalog.ldc.upenn.edu/LDC95T21
European Parliament Proceedings Parallel Corpus	600 million	Legal	`http://www.statmt.org/europarl/`
Google N-Grams Corpus	1 trillion	Written	http://catalog.ldc.upenn.edu/LDC2006T13

The smallest corpus in the list, the complete works of Shakespeare, contains about 0.88 million words. In contrast, the Google *n*-gram corpus contains one trillion words from publicly accessible web pages. Out of the one trillion words in the Google *n*-gram corpus, there might be one million distinct words, which would correspond to one million dimensions. The high dimensionality of text is an important issue, and it has a direct impact on the complexities of many text analysis tasks.

Another major challenge with text analysis is that most of the time the text is not structured. As introduced in Chapter 1, "Introduction to Big Data Analytics," this may include quasi-structured, semi-structured, or unstructured data. Table 9-2 shows some example data sources and data formats that text analysis may have to deal with. Note that this is not meant as an exhaustive list; rather, it highlights the challenge of text analysis.

TABLE 9-2 *Example Data Sources and Formats for Text Analysis*

Data Source	Data Format	Data Structure Type
News articles	TXT, HTML, or Scanned PDF	Unstructured
Literature	TXT, DOC, HTML, or PDF	Unstructured
E-mail	TXT, MSG, or EML	Unstructured
Web pages	HTML	Semi-structured
Server logs	LOG or TXT	Semi-structured or Quasi-structured
Social network API firehoses	XML, JSON, or RSS	Semi-structured
Call center transcripts	TXT	Unstructured

9.1 Text Analysis Steps

A text analysis problem usually consists of three important steps: parsing, search and retrieval, and text mining. Note that a text analysis problem may also consist of other subtasks (such as discourse and segmentation) that are outside the scope of this book.

Parsing is the process that takes unstructured text and imposes a structure for further analysis. The unstructured text could be a plain text file, a weblog, an Extensible Markup Language (XML) file, a HyperText Markup Language (HTML) file, or a Word document. Parsing deconstructs the provided text and renders it in a more structured way for the subsequent steps.

Search and retrieval is the identification of the documents in a corpus that contain search items such as specific words, phrases, topics, or entities like people or organizations. These search items are generally called ***key terms.*** Search and retrieval originated from the field of library science and is now used extensively by web search engines.

Text mining uses the terms and indexes produced by the prior two steps to discover meaningful insights pertaining to domains or problems of interest. With the proper representation of the text, many of the techniques mentioned in the previous chapters, such as clustering and classification, can be adapted to text mining. For example, the *k*-means from Chapter 4, "Advanced Analytical Theory and Methods: Clustering," can be modified to cluster text documents into groups, where each group represents a collection of documents with a similar topic [2]. The distance of a document to a centroid represents how closely the document talks about that topic. Classification tasks such as sentiment analysis and spam filtering are prominent use

cases for the naïve Bayes classifier (Chapter 7, "Advanced Analytical Theory and Methods: Classification"). Text mining may utilize methods and techniques from various fields of study, such as statistical analysis, information retrieval, data mining, and natural language processing.

Note that, in reality, all three steps do not have to be present in a text analysis project. If the goal is to construct a corpus or provide a catalog service, for example, the focus would be the parsing task using one or more text preprocessing techniques, such as part-of-speech (POS) tagging, named entity recognition, lemmatization, or stemming. Furthermore, the three tasks do not have to be sequential. Sometimes their orders might even look like a tree. For example, one could use parsing to build a data store and choose to either search and retrieve the related documents or use text mining on the entire data store to gain insights.

Part-of-Speech (POS) Tagging, Lemmatization, and Stemming

The goal of ***POS tagging*** is to build a model whose input is a sentence, such as:

he saw a fox

and whose output is a tag sequence. Each tag marks the POS for the corresponding word, such as:

PRP VBD DT NN

according to the Penn Treebank POS tags [3]. Therefore, the four words are mapped to pronoun (personal), verb (past tense), determiner, and noun (singular), respectively.

Both lemmatization and stemming are techniques to reduce the number of dimensions and reduce inflections or variant forms to the base form to more accurately measure the number of times each word appears.

With the use of a given dictionary, ***lemmatization*** finds the correct dictionary base form of a word. For example, given the sentence:

obesity causes many problems

the output of lemmatization would be:

obesity cause many problem

Different from lemmatization, ***stemming*** does not need a dictionary, and it usually refers to a crude process of stripping affixes based on a set of heuristics with the hope of correctly achieving the goal to reduce inflections or variant forms. After the process, words are stripped to become *stems*. A stem is not necessarily an actual word defined in the natural language, but it is sufficient to differentiate itself from the stems of other words. A well-known rule-based stemming algorithm is ***Porter's stemming algorithm***. It defines a set of production rules to iteratively transform words into their stems. For the sentence shown previously:

obesity causes many problems

the output of Porter's stemming algorithm is:

obes caus mani problem

9.2 A Text Analysis Example

To further describe the three text analysis steps, consider the fictitious company ACME, maker of two products: `bPhone` and `bEbook`. ACME is in strong competition with other companies that manufacture and sell similar products. To succeed, ACME needs to produce excellent phones and eBook readers and increase sales.

One of the ways the company does this is to monitor what is being said about ACME products in social media. In other words, what is the buzz on its products? ACME wants to search all that is said about ACME products in social media sites, such as Twitter and Facebook, and popular review sites, such as Amazon and ConsumerReports. It wants to answer questions such as these.

- Are people mentioning its products?
- What is being said? Are the products seen as good or bad? If people think an ACME product is bad, why? For example, are they complaining about the battery life of the `bPhone`, or the response time in their `bEbook`?

ACME can monitor the social media buzz using a simple process based on the three steps outlined in Section 9.1. This process is illustrated in Figure 9-1, and it includes the modules in the next list.

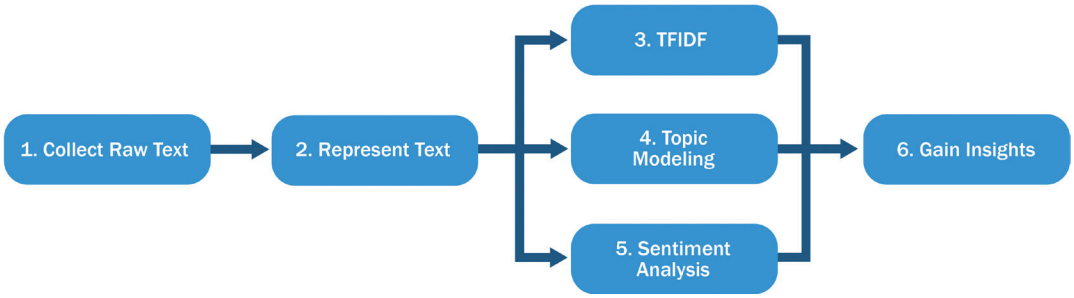

FIGURE 9-1 *ACME's Text Analysis Process*

1. Collect raw text (Section 9.3). This corresponds to Phase 1 and Phase 2 of the Data Analytic Lifecycle. In this step, the Data Science team at ACME monitors websites for references to specific products. The websites may include social media and review sites. The team could interact with social network application programming interfaces (APIs) process data feeds, or scrape pages and use product names as keywords to get the raw data. Regular expressions are commonly used in this case to identify text that matches certain patterns. Additional filters can be applied to the raw data for a more focused study. For example, only retrieving the reviews originating in New York instead of the entire United States would allow ACME to conduct regional studies on its products. Generally, it is a good practice to apply filters during the data collection phase. They can reduce I/O workloads and minimize the storage requirements.
2. Represent text (Section 9.4). Convert each review into a suitable document representation with proper indices, and build a corpus based on these indexed reviews. This step corresponds to Phases 2 and 3 of the Data Analytic Lifecycle.

3. Compute the usefulness of each word in the reviews using methods such as TFIDF (Section 9.5). This and the following two steps correspond to Phases 3 through 5 of the Data Analytic Lifecycle.
4. Categorize documents by topics (Section 9.6). This can be achieved through topic models (such as latent Dirichlet allocation).
5. Determine sentiments of the reviews (Section 9.7). Identify whether the reviews are positive or negative. Many product review sites provide ratings of a product with each review. If such information is not available, techniques like sentiment analysis can be used on the textual data to infer the underlying sentiments. People can express many emotions. To keep the process simple, ACME considers sentiments as positive, neutral, or negative.
6. Review the results and gain greater insights (Section 9.8). This step corresponds to Phase 5 and 6 of the Data Analytic Lifecycle. Marketing gathers the results from the previous steps. Find out what exactly makes people love or hate a product. Use one or more visualization techniques to report the findings. Test the soundness of the conclusions and operationalize the findings if applicable.

This process organizes the topics presented in the rest of the chapter and calls out some of the difficulties that are unique to text analysis.

9.3 Collecting Raw Text

Recall that in the Data Analytic Lifecycle seen in Chapter 2, "Data Analytics Lifecycle," discovery is the first phase. In it, the Data Science team investigates the problem, understands the necessary data sources, and formulates initial hypotheses. Correspondingly, for text analysis, data must be collected before anything can happen. The Data Science team starts by actively monitoring various websites for user-generated contents. The user-generated contents being collected could be related articles from news portals and blogs, comments on ACME's products from online shops or reviews sites, or social media posts that contain keywords `bPhone` or `bEbook`. Regardless of where the data comes from, it's likely that the team would deal with semi-structured data such as HTML web pages, Really Simple Syndication (RSS) feeds, XML, or JavaScript Object Notation (JSON) files. Enough structure needs to be imposed to find the part of the raw text that the team really cares about. In the brand management example, ACME is interested in what the reviews say about `bPhone` or `bEbook` and when the reviews are posted. Therefore, the team will actively collect such information.

Many websites and services offer public APIs [4, 5] for third-party developers to access their data. For example, the Twitter API [6] allows developers to choose from the Streaming API or the REST API to retrieve public Twitter posts that contain the keywords `bPhone` or `bEbook`. Developers can also read tweets in real time from a specific user or tweets posted near a specific venue. The fetched tweets are in the JSON format.

As an example, a sample tweet that contains the keyword `bPhone` fetched using the Twitter Streaming API version 1.1 is shown next.

```
01 {
02   "created_at": "Thu Aug 15 20:06:48 +0000 2013",
03   "coordinates": {
04       "type": "Point",
05       "coordinates": [
```

```
            -157.81538521787621,
            21.3002578885766
        ]
    },
        "favorite_count": 0,
        "id": 368101488276824010,
        "id_str": "368101488276824014",
        "lang": "en",
        "metadata": {
            "iso_language_code": "en",
            "result_type": "recent"
        },
        "retweet_count": 0,
        "retweeted": false,
        "source": "<a href=\"http://www.twitter.com\"
                    rel=\"nofollow\">Twitter for bPhone</a>",
        "text": "I once had a gf back in the day. Then the bPhone
                    came out lol",
        "truncated": false,
        "user": {
            "contributors_enabled": false,
            "created_at": „Mon Jun 24 09:15:54 +0000 2013",
            "default_profile": false,
            "default_profile_image": false,
            "description": "Love Life and Live Good",
            "favourites_count": 23,
            "follow_request_sent": false,
            "followers_count": 96,
            "following": false,
            "friends_count": 347,
            "geo_enabled": false,
            "id": 2542887414,
            "id_str": "2542887414",
            "is_translator": false,
            "lang": "en-gb",
            "listed_count": 0,
            "location": "Beautiful Hawaii",
            "name": "The Original DJ Ice",
            "notifications": false,
            "profile_background_color": "C0DEED",
            "profile_background_image_url":
"http://a0.twimg.com/profile_bg_imgs/378800000/b12e56725ee.jpeg",
            "profile_background_tile": true,
            "profile_image_url":
"http://a0.twimg.com/profile_imgs/378800010/2d55a4388bcffd5.jpeg",
            "profile_link_color": "0084B4",
            "profile_sidebar_border_color": "FFFFFF",
```

```
53            "profile_sidebar_fill_color": "DDEEF6",
54            "profile_text_color": "333333",
55            "profile_use_background_image": true,
56            "protected": false,
57            "screen_name": "DJ_Ice",
58            "statuses_count": 186,
59            "time_zone": "Hawaii",
60            "url": null,
61            "utc_offset": -36000,
62            "verified": false
63        }
64 }
```

Fields *created_at* at line 2 and *text* at line 22 in the previous tweet provide the information that interests ACME. The *created_at* entry stores the timestamp that the tweet was published, and the *text* field stores the main content of the Twitter post. Other fields could be useful, too. For example, utilizing fields such as *coordinates* (line 3 to 9), user's local language (*lang*, line 40), user's *location* (line 42), *time_zone* (line 59), and *utc_offset* (line 61) allows the analysis to focus on tweets from a specific region. Therefore, the team can research what people say about ACME's products at a more granular level.

Many news portals and blogs provide data feeds that are in an open standard format, such as RSS or XML. As an example, an RSS feed for a phone review blog is shown next.

```
01 <channel>
02     <title>All about Phones</title>
03     <description>My Phone Review Site</description>
04     <link>http://www.phones.com/link.htm</link>
05
06     <item>
07         <title>bPhone: The best!</title>
08         <description>I love LOVE my bPhone!</description>
09         <link>http://www.phones.com/link.htm</link>
10         <guid isPermaLink="false">1102345</guid>
11         <pubDate>Tue, 29 Aug 2011 09:00:00 -0400</pubDate>
12     <item>
13 </channel>
```

The content from the *title* (line 7), the *description* (line 8), and the published date (*pubDate*, line 11) is what ACME is interested in.

If the plan is to collect user comments on ACME's products from online shops and review sites where APIs or data feeds are not provided, the team may have to write web scrapers to parse web pages and automatically extract the interesting data from those HTML files. A ***web scraper*** is a software program (bot) that systematically browses the World Wide Web, downloads web pages, extracts useful information, and stores it somewhere for further study.

Unfortunately, it is nearly impossible to write a one-size-fits-all web scraper. This is because websites like online shops and review sites have different structures. It is common to customize a web scraper for a specific website. In addition, the website formats can change over time, which requires the web scraper to

be updated every now and then. To build a web scraper for a specific website, one must study the HTML source code of its web pages to find patterns before extracting any useful content. For example, the team may find out that each user comment in the HTML is enclosed by a DIV element inside another DIV with the ID *usrcommt*, or it might be enclosed by a DIV element with the CLASS *commtcls*.

The team can then construct the web scraper based on the identified patterns. The scraper can use the `curl` tool [7] to fetch HTML source code given specific URLs, use XPath [8] and regular expressions to select and extract the data that match the patterns, and write them into a data store.

Regular expressions can find words and strings that match particular patterns in the text effectively and efficiently. Table 9-3 shows some regular expressions. The general idea is that once text from the fields of interest is obtained, regular expressions can help identify if the text is really interesting for the project. In this case, do those fields mention *bPhone*, *bEbook*, or *ACME*? When matching the text, regular expressions can also take into account capitalizations, common misspellings, common abbreviations, and special formats for e-mail addresses, dates, and telephone numbers.

TABLE 9-3 *Example Regular Expressions*

Regular Expression	Matches	Note
`b(P|p)hone`	bPhone, bphone	Pipe "|" means "or"
`bEbo*k`	bEbk, bEbok, bEbook, bEboook, bEbooook, bEboooook, …	"*" matches zero or more occurrences of the preceding letter
`bEbo+k`	bEbok, bEbook, bEboook, bEbooook, bEboooook, …	"+" matches one or more occurrences of the preceding letter
`bEbo{2,4}k`	bEbook, bEboook, bEbooook	"{2,4}" matches from two to four repetitions of the preceding letter "o"
`^I love`	Text starting with "I love"	"^" matches the start of a string
`ACME$`	Text ending with "ACME"	"$" matches the end of a string

This section has discussed three different sources where raw data may come from: tweets that contain keywords *bPhone* or *bEbook*, related articles from news portals and blogs, and comments on ACME's products from online shops or reviews sites.

If one chooses not to build a data collector from scratch, many companies such as GNIP [9] and DataSift [10] can provide data collection or data reselling services.

Depending on how the fetched raw data will be used, the Data Science team needs to be careful not to violate the rights of the owner of the information and user agreements about use of websites during the data collection. Many websites place a file called `robots.txt` in the root directory—that is, `http://.../robots.txt` (for example, `http://www.amazon.com/robots.txt`). It lists the directories and files that are allowed or disallowed to be visited so that web scrapers or web crawlers know how to treat the website correctly.

9.4 Representing Text

After the previous step, the team now has some raw text to start with. In this data representation step, raw text is first transformed with text normalization techniques such as tokenization and case folding. Then it is represented in a more structured way for analysis.

Tokenization is the task of separating (also called tokenizing) words from the body of text. Raw text is converted into collections of tokens after the tokenization, where each token is generally a word.

A common approach is tokenizing on spaces. For example, with the tweet shown previously:

```
I once had a gf back in the day. Then the bPhone came out lol
```

tokenization based on spaces would output a list of tokens.

```
{I, once, had, a, gf, back, in, the, day.,
Then, the, bPhone, came, out, lol}
```

Note that token "`day.`" contains a period. This is the result of only using space as the separator. Therefore, tokens "`day.`" and "`day`" would be considered different terms in the downstream analysis unless an additional lookup table is provided. One way to fix the problem without the use of a lookup table is to remove the period if it appears at the end of a sentence. Another way is to tokenize the text based on punctuation marks and spaces. In this case, the previous tweet would become:

```
{I, once, had, a, gf, back, in, the, day, .,
Then, the, bPhone, came, out, lol}
```

However, tokenizing based on punctuation marks might not be well suited to certain scenarios. For example, if the text contains contractions such as `we'll`, tokenizing based on punctuation will split them into separated words `we` and `ll`. For words such as `can't`, the output would be `can` and `t`. It would be more preferable either not to tokenize them or to tokenize `we'll` into `we` and `'ll`, and `can't` into `can` and `'t`. The `'t` token is more recognizable as negative than the `t` token. If the team is dealing with certain tasks such as information extraction or sentiment analysis, tokenizing solely based on punctuation marks and spaces may obscure or even distort meanings in the text.

Tokenization is a much more difficult task than one may expect. For example, should words like `state-of-the-art`, `Wi-Fi`, and `San Francisco` be considered one token or more? Should words like `Résumé`, `résumé`, and `resume` all map to the same token? Tokenization is even more difficult beyond English. In German, for example, there are many unsegmented compound nouns. In Chinese, there are no spaces between words. Japanese has several alphabets intermingled. This list can go on.

It's safe to say that there is no single tokenizer that will work in every scenario. The team needs to decide what counts as a token depending on the domain of the task and select an appropriate tokenization technique that fits most situations well. In reality, it's common to pair a standard tokenization technique with a lookup table to address the contractions and terms that should not be tokenized. Sometimes it may not be a bad idea to develop one's own tokenization from scratch.

Another text normalization technique is called ***case folding***, which reduces all letters to lowercase (or the opposite if applicable). For the previous tweet, after case folding the text would become this:

```
i once had a gf back in the day. then the bphone came out lol
```

One needs to be cautious applying case folding to tasks such as information extraction, sentiment analysis, and machine translation. If implemented incorrectly, case folding may reduce or change the meaning of the text and create additional noise. For example, when `General Motors` becomes `general` and `motors`, the downstream analysis may very likely consider them as separated words rather than the name of a company. When the abbreviation of the World Health Organization `WHO` or the rock band `The Who` become `who`, they may both be interpreted as the pronoun `who`.

If case folding must be present, one way to reduce such problems is to create a lookup table of words not to be case folded. Alternatively, the team can come up with some heuristics or rules-based strategies for the case folding. For example, the program can be taught to ignore words that have uppercase in the middle of a sentence.

After normalizing the text by tokenization and case folding, it needs to be represented in a more structured way. A simple yet widely used approach to represent text is called ***bag-of-words***. Given a document, bag-of-words represents the document as a set of terms, ignoring information such as order, context, inferences, and discourse. Each word is considered a term or token (which is often the smallest unit for the analysis). In many cases, bag-of-words additionally assumes every term in the document is independent. The document then becomes a vector with one dimension for every distinct term in the space, and the terms are unordered. The permutation D* of a document D contains the same words exactly the same number of times but in a different order. Therefore, using the bag-of-words representation, document D and its permutation D* would share the same representation.

Bag-of-words takes quite a naïve approach, as order plays an important role in the semantics of text. With bag-of-words, many texts with different meanings are combined into one form. For example, the texts "a dog bites a man" and "a man bites a dog" have very different meanings, but they would share the same representation with bag-of-words.

Although the bag-of-words technique oversimplifies the problem, it is still considered a good approach to start with, and it is widely used for text analysis. A paper by Salton and Buckley [11] states the effectiveness of using single words as identifiers as opposed to multiple-term identifiers, which retain the order of the words:

> ***In reviewing the extensive literature accumulated during the past 25 years in the area of retrieval system evaluation, the overwhelming evidence is that the judicious use of single-term identifiers is preferable to the incorporation of more complex entities extracted from the texts themselves or obtained from available vocabulary schedules.***

Although the work by Salton and Buckley was published in 1988, there has been little, if any, substantial evidence to discredit the claim. Bag-of-words uses single-term identifiers, which are usually sufficient for the text analysis in place of multiple-term identifiers.

Using single words as identifiers with the bag-of-words representation, the ***term frequency*** (TF) of each word can be calculated. Term frequency represents the weight of each term in a document, and it is proportional to the number of occurrences of the term in that document. Figure 9-2 shows the 50 most frequent words and the numbers of occurrences from Shakespeare's *Hamlet*. The word frequency distribution roughly follows ***Zipf's Law*** [12, 13]—that is, the i-th most common word occurs approximately $1/i$ as

often as the most frequent term. In other words, the frequency of a word is inversely proportional to its rank in the frequency table. Term frequency is revisited later in this chapter.

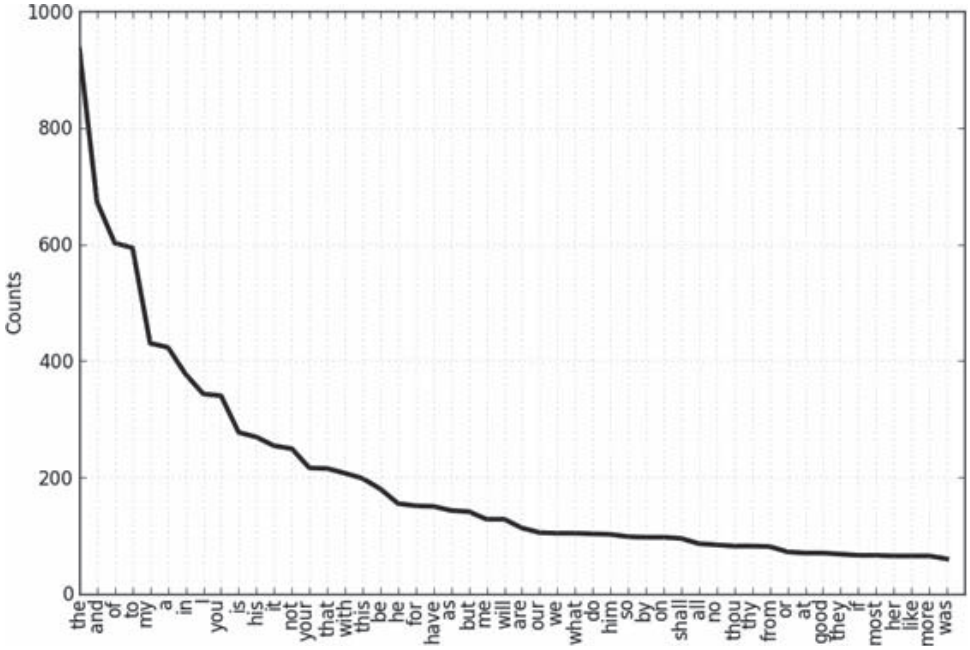

FIGURE 9-2 *The 50 most frequent words in Shakespeare's Hamlet*

What's Beyond Bag-of-Words?

Bag-of-words is a common technique to start with. But sometimes the Data Science team prefers other methods of text representation that are more sophisticated. These more advanced methods consider factors such as word order, context, inferences, and discourse. For example, one such method can keep track of the word order of every document and compare the normalized differences of the word orders [14]. These advanced techniques are outside the scope of this book.

Besides extracting the terms, their morphological ***features*** may need to be included. The morphological features specify additional information about the terms, which may include root words, affixes, part-of-speech tags, named entities, or intonation (variations of spoken pitch). The features from this step contribute to the downstream analysis in classification or sentiment analysis.

The set of features that need to be extracted and stored highly depends on the specific task to be performed. If the task is to label and distinguish the part of speech, for example, the features will include all the words in the text and their corresponding part-of-speech tags. If the task is to annotate the named entities

like names and organizations, the features highlight such information appearing in the text. Constructing the features is no trivial task; quite often this is done entirely manually, and sometimes it requires domain expertise.

Sometimes creating features is a text analysis task all to itself. One such example is ***topic modeling.*** Topic modeling provides a way to quickly analyze large volumes of raw text and identify the latent topics. Topic modeling may not require the documents to be labeled or annotated. It can discover topics directly from an analysis of the raw text. A topic consists of a cluster of words that frequently occur together and that share the same theme. Probabilistic topic modeling, discussed in greater detail later in Section 9.6, is a suite of algorithms that aim to parse large archives of documents and discover and annotate the topics.

It is important not only to create a representation of a document but also to create a representation of a corpus. As introduced earlier in the chapter, a corpus is a collection of documents. A corpus could be so large that it includes all the documents in one or more languages, or it could be smaller or limited to a specific domain, such as technology, medicine, or law. For a web search engine, the entire World Wide Web is the relevant corpus. Most corpora are much smaller. The Brown Corpus [15] was the first million-word electronic corpus of English, created in 1961 at Brown University. It includes text from around 500 sources, and the source has been categorized into 15 genres, such as news, editorial, fiction, and so on. Table 9-4 lists the genres of the Brown Corpus as an example of how to organize information in a corpus.

TABLE 9-4 *Categories of the Brown Corpus*

Category	Number of Sources	Example Source
A. Reportage	44	*Chicago Tribune*
B. Editorial	27	*Christian Science Monitor*
C. Reviews	17	*Life*
D. Religion	17	*William Pollard: Physicist and Christian*
E. Skills and Hobbies	36	*Joseph E. Choate: The American Boating Scene*
F. Popular Lore	48	*David Boroff: Jewish Teen-Age Culture*
G. Belles Lettres, Biography, Memoirs, and so on	75	*Selma J. Cohen: Avant-Garde Choreography*
H. Miscellaneous	30	*U. S. Dep't of Defense: Medicine in National Defense*
J. Learned	80	*J. F. Vedder: Micrometeorites*
K. General Fiction	29	*David Stacton: The Judges of the Secret Court*

(*continues*)

Table 9-4 *Categories of the Brown Corpus* (*Continued*)

Category	Number of Sources	Example Source
L. Mystery and Detective Fiction	24	*S. L. M. Barlow: Monologue of Murder*
M. Science Fiction	6	*Jim Harmon: The Planet with No Nightmare*
N. Adventure and Western Fiction	29	*Paul Brock: Toughest Lawman in the Old West*
P. Romance and Love Story	29	*Morley Callaghan: A Passion in Rome*
R. Humor	9	*Evan Esar: Humorous English*

Many corpora focus on specific domains. For example, the BioCreative corpora [16] are from biology, the Switchboard corpus [17] contains telephone conversations, and the European Parliament Proceedings Parallel Corpus [18] was extracted from the proceedings of the European Parliament in 21 European languages.

Most corpora come with metadata, such as the size of the corpus and the domains from which the text is extracted. Some corpora (such as the Brown Corpus) include the information content of every word appearing in the text. ***Information content*** (IC) is a metric to denote the importance of a term in a corpus. The conventional way [19] of measuring the IC of a term is to combine the knowledge of its hierarchical structure from an ontology with statistics on its actual usage in text derived from a corpus. Terms with higher IC values are considered more important than terms with lower IC values. For example, the word *necklace* generally has a higher IC value than the word *jewelry* in an English corpus because *jewelry* is more general and is likely to appear more often than *necklace*. Research shows that IC can help measure the semantic similarity of terms [20]. In addition, such measures do not require an annotated corpus, and they generally achieve strong correlations with human judgment [21, 20].

In the brand management example, the team has collected the ACME product reviews and turned them into the proper representation with the techniques discussed earlier. Next, the reviews and the representation need to be stored in a searchable archive for future reference and research. This archive could be a SQL database, XML or JSON files, or plain text files from one or more directories.

Corpus statistics such as IC can help identify the importance of a term from the documents being analyzed. However, IC values included in the metadata of a traditional corpus (such as Brown corpus) sitting externally as a knowledge base cannot satisfy the need to analyze the dynamically changed, unstructured data from the web. The problem is twofold. First, both traditional corpora and IC metadata do not change over time. Any term not existing in the corpus text and any newly invented words would automatically receive a zero IC value. Second, the corpus represents the entire knowledge base for the algorithm being used in the downstream analysis. The nature of the unstructured text determines that the data being analyzed can contain any topics, many of which may be absent in the given knowledge base. For example, if the task is to research people's attitudes on musicians, a traditional corpus constructed 50 years ago would not know that the term *U2* is a band; therefore, it would receive a zero on IC, which means it's not an

important term. A better approach would go through all the fetched documents and find out that most of them are related to music, with *U2* appearing too often to be an unimportant term. Therefore, it is necessary to come up with a metric that can easily adapt to the context and nature of the text instead of relying on a traditional corpus. The next section discusses such a metric. It's known as Term Frequency—Inverse Document Frequency (TFIDF), which is based entirely on all the fetched documents and which keeps track of the importance of terms occurring in each of the documents.

Note that the fetched documents may change constantly over time. Consider the case of a web search engine, in which each fetched document corresponds to a matching web page in a search result. The documents are added, modified, or removed and, as a result, the metrics and indices must be updated correspondingly. Additionally, word distributions can change over time, which reduces the effectiveness of classifiers and filters (such as spam filters) unless they are retrained.

9.5 Term Frequency—Inverse Document Frequency (TFIDF)

This section presents TFIDF, a measure widely used in information retrieval and text analysis. Instead of using a traditional corpus as a knowledge base, TFIDF directly works on top of the fetched documents and treats these documents as the "corpus." TFIDF is robust and efficient on dynamic content, because document changes require only the update of frequency counts.

Given a term t and a document $d = \{t_1, t_2, t_3, \ldots t_n\}$ containing n terms, the simplest form of term frequency of t in d can be defined as the number of times t appears in d, as shown in Equation 9-1.

$$TF_1(t,d) = \sum_{i=1}^{n} f(t,t_i) \qquad t_i \in d; |d| = n$$

where

$$f(t,t') = \begin{cases} 1, & \text{if } t = t' \\ 0, & \text{otherwise} \end{cases} \tag{9-1}$$

To understand how the term frequency is computed, consider a bag-of-words vector space of 10 words: `i`, `love`, `acme`, `my`, `bebook`, `bphone`, `fantastic`, `slow`, `terrible`, and `terrific`. Given the text `I love LOVE my bPhone` extracted from the RSS feed in Section 9.3, Table 9-5 shows its corresponding term frequency vector after case folding and tokenization.

TABLE 9-5 *A Sample Term Frequency Vector*

Term	Frequency
i	1
love	2
acme	0

(*continues*)

TABLE 9-5 *A Sample Term Frequency Vector* (*Continued*)

Term	Frequency
my	1
bebook	0
bphone	1
fantastic	0
slow	0
terrible	0
terrific	0

The term frequency function can be logarithmically scaled. Recall that in Figure 3-11 and Figure 3-12 of Chapter 3, "Review of Basic Data Analytic Methods Using R," it shows the logarithm can be applied to distribution with a long tail to enable more data detail. Similarly, the logarithm can be applied to word frequencies whose distribution also contains a long tail, as shown in Equation 9-2.

$$TF_2(t,d) = \log[TF_1(t,d) + 1] \tag{9-2}$$

Because longer documents contain more terms, they tend to have higher term frequency values. They also tend to contain more distinct terms. These factors can conspire to raise the term frequency values of longer documents and lead to undesirable bias favoring longer documents. To address this problem, the term frequency can be normalized. For example, the term frequency of term `t` in document `d` can be normalized based on the number of terms in `d` as shown in Equation 9-3.

$$TF_3(t,d) = \frac{TF_1(t,d)}{n} \qquad |d| = n \tag{9-3}$$

Besides the three common definitions mentioned earlier, there are other less common variations [22] of term frequency. In practice, one needs to choose the term frequency definition that is the most suitable to the data and the problem to be solved.

A term frequency vector (shown in Table 9-5) can become very high dimensional because the bag-of-words vector space can grow substantially to include all the words in English. The high dimensionality makes it difficult to store and parse the text and contribute to performance issues related to text analysis.

For the purpose of reducing dimensionality, not all the words from a given language need to be included in the term frequency vector. In English, for example, it is common to remove words such as `the`, `a`, `of`, `and`, `to`, and other articles that are not likely to contribute to semantic understanding. These common words are called ***stop words***. Lists of stop words are available in various languages for automating the identification of stop words. Among them is the *Snowball's stop words list* [23] that contains stop words in more than ten languages.

Another simple yet effective way to reduce dimensionality is to store a term and its frequency only if the term appears at least once in a document. Any term not existing in the term frequency vector by default will have a frequency of 0. Therefore, the previous term frequency vector would be simplified to what is shown in Table 9-6.

Table 9-6 *A Simpler Form of the Term Frequency Vector*

Term	Frequency
i	1
love	2
my	1
bphone	1

Some NLP techniques such as lemmatization and stemming can also reduce high dimensionality. Lemmatization and stemming are two different techniques that combine various forms of a word. With these techniques, words such as `play`, `plays`, `played`, and `playing` can be mapped to the same term.

It has been shown that the term frequency is based on the raw count of a term occurring in a stand-alone document. Term frequency by itself suffers a critical problem: It regards that stand-alone document as the entire world. The importance of a term is solely based on its presence in this particular document. Stop words such as `the`, `and`, and `a` could be inappropriately considered the most important because they have the highest frequencies in every document. For example, the top three most frequent words in Shakespeare's *Hamlet* are all stop words (`the`, `and`, and `of`, as shown in Figure 9-2). Besides stop words, words that are more general in meaning tend to appear more often, thus having higher term frequencies. In an article about consumer telecommunications, the word `phone` would be likely to receive a high term frequency. As a result, the important keywords such as `bPhone` and `bEbook` and their related words could appear to be less important. Consider a search engine that responds to a search query and fetches relevant documents. Using term frequency alone, the search engine would not properly assess how relevant each document is in relation to the search query.

A quick fix for the problem is to introduce an additional variable that has a broader view of the world—considering the importance of a term not only in a single document but in a collection of documents, or in a corpus. The additional variable should reduce the effect of the term frequency as the term appears in more documents.

Indeed, that is the intention of the ***inverted document frequency*** (IDF). The IDF inversely corresponds to the ***document frequency*** (DF), which is defined to be the number of documents in the corpus that contain a term. Let a corpus `D` contain `N` documents. The document frequency of a term `t` in corpus $D = \{d_1, d_2, \ldots d_N\}$ is defined as shown in Equation 9-4.

$$DF(t) = \sum_{i=1}^{N} f'(t, d_i) \qquad d_i \in D; |D| = N$$

where

$$f'(t, d') = \begin{cases} 1, & if \ \ t \in d' \\ 0, & otherwise \end{cases} \tag{9-4}$$

The Inverse document frequency of a term t is obtained by dividing N by the document frequency of the term and then taking the logarithm of that quotient, as shown in Equation 9-5.

$$IDF_1(t) = \log \frac{N}{DF(t)} \tag{9-5}$$

If the term is not in the corpus, it leads to a division-by-zero. A quick fix is to add 1 to the denominator, as demonstrated in Equation 9-6.

$$IDF_2(t) = \log \frac{N}{DF(t) + 1} \tag{9-6}$$

The precise base of the logarithm is not material to the ranking of a term. Mathematically, the base constitutes a constant multiplicative factor towards the overall result.

Figure 9-3 shows 50 words with (a) the highest corpus-wide term frequencies (TF), (b) the highest document frequencies (DF), and (c) the highest Inverse document frequencies (IDF) from the news category of the Brown Corpus. Stop words tend to have higher TF and DF because they are likely to appear more often in most documents.

Words with higher IDF tend to be more meaningful over the entire corpus. In other words, the IDF of a rare term would be high, and the IDF of a frequent term would be low. For example, if a corpus contains 1,000 documents, 1,000 of them might contain the word `the`, and 10 of them might contain the word `bPhone`. With Equation 9-5, the IDF of `the` would be 0, and the IDF of `bPhone` would be log100, which is greater than the IDF of `the`. If a corpus consists of mostly phone reviews, the word `phone` would probably have high TF and DF but low IDF.

Despite the fact that IDF encourages words that are more meaningful, it comes with a caveat. Because the total document count of a corpus (N) remains a constant, IDF solely depends on the DF. All words having the same DF value therefore receive the same IDF value. IDF scores words higher that occur less frequently across the documents. Those words that score the lowest DF receive the same highest IDF. In Figure 9-3 (c), for example, `sunbonnet` and `narcotic` appeared in an equal number of documents in the Brown corpus; therefore, they received the same IDF values. In many cases, it is useful to distinguish between two words that appear in an equal number of documents. Methods to further weight words should be considered to refine the IDF score.

The TFIDF (or TF-IDF) is a measure that considers both the prevalence of a term within a document (TF) and the scarcity of the term over the entire corpus (IDF). The TFIDF of a term t in a document d is defined as the term frequency of t in d multiplying the document frequency of t in the corpus as shown in Equation 9-7:

$$TFIDF(t, d) = TF(t, d) \times IDF(t) \tag{9-7}$$

TFIDF scores words higher that appear more often in a document but occur less often across all documents in the corpus. Note that TFIDF applies to a term in a specific document, so the same term is likely to receive different TFIDF scores in different documents (because the TF values may be different).

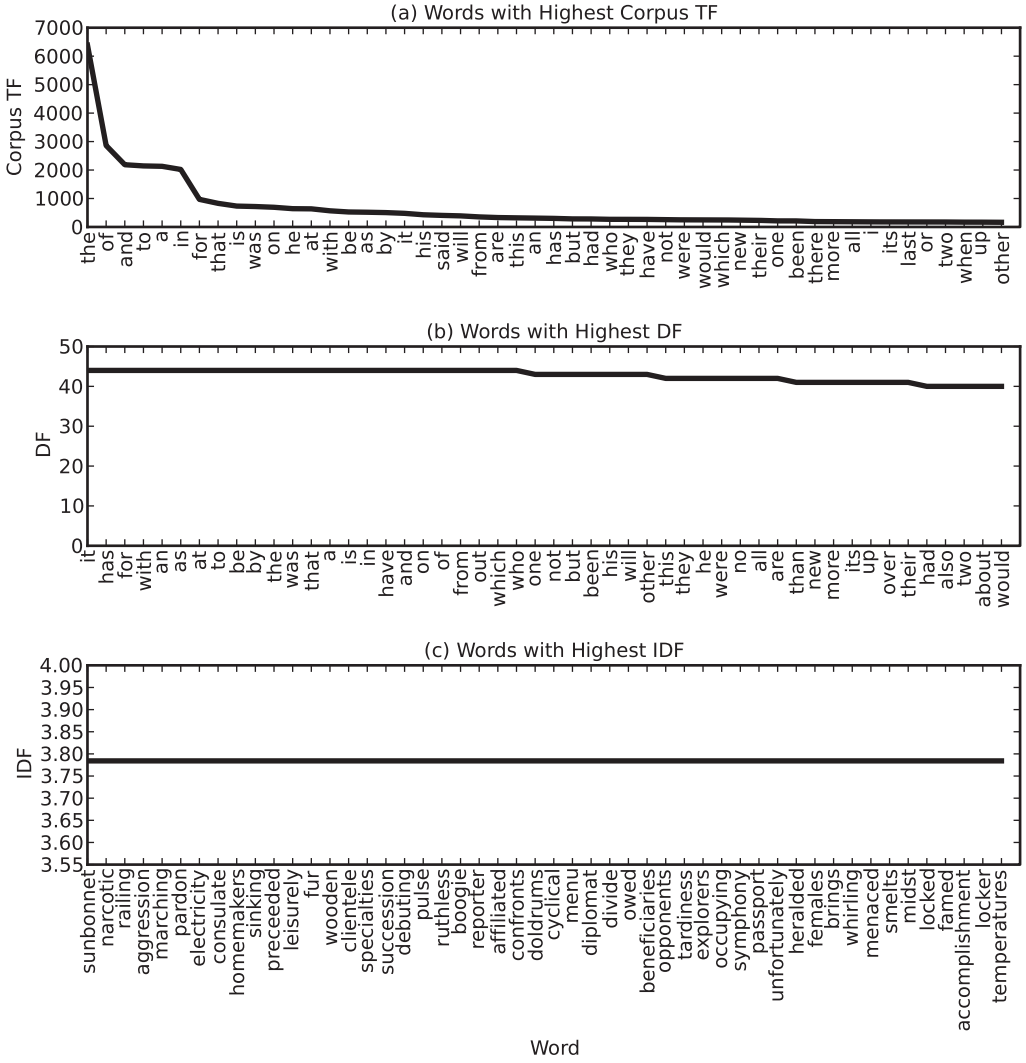

FIGURE 9-3 *Words from Brown corpus's news category with the highest corpus TF, DF, or IDF*

TFIDF is efficient in that the calculations are simple and straightforward, and it does not require knowledge of the underlying meanings of the text. But this approach also reveals little of the inter-document or intra-document statistical structure. The next section shows how topic models can address this shortcoming of TFIDF.

9.6 Categorizing Documents by Topics

With the reviews collected and represented, the data science team at ACME wants to categorize the reviews by topics. As discussed earlier in the chapter, a topic consists of a cluster of words that frequently occur together and share the same theme.

The topics of a document are not as straightforward as they might initially appear. Consider these two reviews:

1. The bPhone5x has coverage everywhere. It's much less flaky than my old bPhone4G.
2. While I love ACME's bPhone series, I've been quite disappointed by the bEbook. The text is illegible, and it makes even my old NBook look blazingly fast.

Is the first review about bPhone5x or bPhone4G? Is the second review about bPhone, bEbook, or NBook? For machines, these questions can be difficult to answer.

Intuitively, if a review is talking about bPhone5x, the term `bPhone5x` and related terms (such as `phone` and `ACME`) are likely to appear frequently. A document typically consists of multiple themes running through the text in different proportions—for example, 30% on a topic related to `phones`, 15% on a topic related to `appearance`, 10% on a topic related to `shipping`, 5% on a topic related to `service`, and so on.

Document grouping can be achieved with clustering methods such as *k*-means clustering [24] or classification methods such as support vector machines [25], *k*-nearest neighbors [26], or naïve Bayes [27]. However, a more feasible and prevalent approach is to use ***topic modeling***. Topic modeling provides tools to automatically organize, search, understand, and summarize from vast amounts of information. ***Topic models*** [28, 29] are statistical models that examine words from a set of documents, determine the themes over the text, and discover how the themes are associated or change over time. The process of topic modeling can be simplified to the following.

1. Uncover the hidden topical patterns within a corpus.
2. Annotate documents according to these topics.
3. Use annotations to organize, search, and summarize texts.

A ***topic*** is formally defined as a distribution over a fixed vocabulary of words [29]. Different topics would have different distributions over the same vocabulary. A topic can be viewed as a cluster of words with related meanings, and each word has a corresponding weight inside this topic. Note that a word from the vocabulary can reside in multiple topics with different weights. Topic models do not necessarily require prior knowledge of the texts. The topics can emerge solely based on analyzing the text.

The simplest topic model is ***latent Dirichlet allocation*** (LDA) [29], a generative probabilistic model of a corpus proposed by David M. Blei and two other researchers. In generative probabilistic modeling, data

is treated as the result of a generative process that includes hidden variables. LDA assumes that there is a fixed vocabulary of words, and the number of the latent topics is predefined and remains constant. LDA assumes that each latent topic follows a Dirichlet distribution [30] over the vocabulary, and each document is represented as a random mixture of latent topics.

Figure 9-4 illustrates the intuitions behind LDA. The left side of the figure shows four topics built from a corpus, where each topic contains a list of the most important words from the vocabulary. The four example topics are related to problem, policy, neural, and report. For each document, a distribution over the topics is chosen, as shown in the histogram on the right. Next, a topic assignment is picked for each word in the document, and the word from the corresponding topic (colored discs) is chosen. In reality, only the documents (as shown in the middle of the figure) are available. The goal of LDA is to infer the underlying topics, topic proportions, and topic assignments for every document.

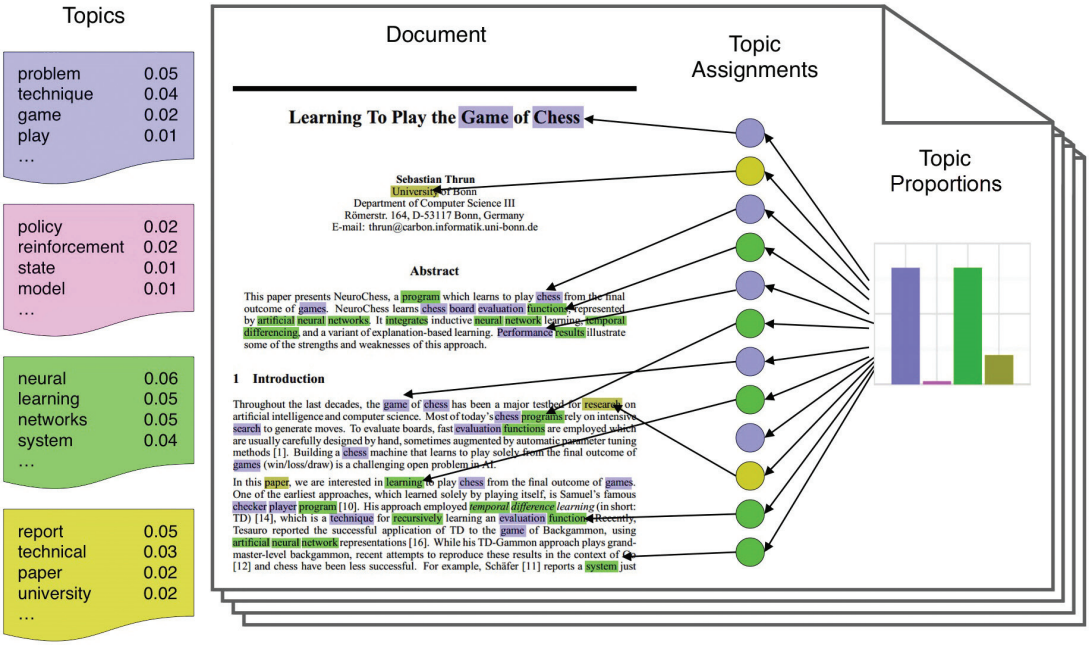

FIGURE 9-4 *The intuitions behind LDA*

The reader can refer to the original paper [29] for the mathematical detail of LDA. Basically, LDA can be viewed as a case of hierarchical Bayesian estimation with a posterior distribution to group data such as documents with similar topics.

Many programming tools provide software packages that can perform LDA over datasets. R comes with an `lda` package [31] that has built-in functions and sample datasets. The `lda` package was developed by David M. Blei's research group [32]. Figure 9-5 shows the distributions of ten topics on nine scientific documents randomly drawn from the `cora` dataset of the `lda` package. The `cora` dataset is a collection of 2,410 scientific documents extracted from the Cora search engine [33].

FIGURE 9-5 *Distributions of ten topics over nine scientific documents from the Cora dataset*

The code that follows shows how to generate a graph similar to Figure 9-5 using R and add-on packages such as `lda` and `ggplot`.

```
require("ggplot2")
require("reshape2")
require("lda")

# load documents and vocabulary
data(cora.documents)
data(cora.vocab)

theme_set(theme_bw())

# Number of topic clusters to display
K <- 10

# Number of documents to display
N <- 9
```

```
result <- lda.collapsed.gibbs.sampler(cora.documents,
                                      K,  ## Num clusters
                                      cora.vocab,
                                      25,  ## Num iterations
                                      0.1,
                                      0.1,
                                      compute.log.likelihood=TRUE)

# Get the top words in the cluster
top.words <- top.topic.words(result$topics, 5, by.score=TRUE)

# build topic proportions
topic.props <- t(result$document_sums) / colSums(result$document_sums)

document.samples <- sample(1:dim(topic.props)[1], N)
topic.props <- topic.props[document.samples,]

topic.props[is.na(topic.props)] <-  1 / K

colnames(topic.props) <- apply(top.words, 2, paste, collapse=" ")

topic.props.df <- melt(cbind(data.frame(topic.props),
                             document=factor(1:N)),
                       variable.name="topic",
                       id.vars = "document")

qplot(topic, value*100, fill=topic, stat="identity",
      ylab="proportion (%)", data=topic.props.df,
      geom="histogram") +
  theme(axis.text.x = element_text(angle=0, hjust=1, size=12)) +
  coord_flip() +
  facet_wrap(~ document, ncol=3)
```

Topic models can be used in document modeling, document classification, and collaborative filtering [29]. Topic models not only can be applied to textual data, they can also help annotate images. Just as a document can be considered a collection of topics, images can be considered a collection of image features.

9.7 Determining Sentiments

In addition to the TFIDF and topic models, the Data Science team may want to identify the sentiments in user comments and reviews of the ACME products. ***Sentiment analysis*** refers to a group of tasks that use statistics and natural language processing to mine opinions to identify and extract subjective information from texts.

Early work on sentiment analysis focused on detecting the polarity of product reviews from Epinions [34] and movie reviews from the Internet Movie Database (IMDb) [35] at the document level. Later work handles sentiment analysis at the sentence level [36]. More recently, the focus has shifted to phrase-level [37] and short-text forms in response to the popularity of micro-blogging services such as Twitter [38, 39, 40, 41, 42].

Intuitively, to conduct sentiment analysis, one can manually construct lists of words with positive sentiments (such as *brilliant*, *awesome*, and *spectacular*) and negative sentiments (such as *awful*, *stupid*, and *hideous*). Related work has pointed out that such an approach can be expected to achieve accuracy around 60% [35], and it is likely to be outperformed by examination of corpus statistics [43].

Classification methods such as naïve Bayes as introduced in Chapter 7, maximum entropy (MaxEnt), and support vector machines (SVM) are often used to extract corpus statistics for sentiment analysis. Related research has found out that these classifiers can score around 80% accuracy [35, 41, 42] on sentiment analysis over unstructured data. One or more of such classifiers can be applied to unstructured data, such as movie reviews or even tweets.

The movie review corpus by Pang et al. [35] includes 2,000 movie reviews collected from an IMDb archive of the rec.arts.movies.reviews newsgroup [43]. These movie reviews have been manually tagged into 1,000 positive reviews and 1,000 negative reviews.

Depending on the classifier, the data may need to be split into training and testing sets. As seen previously in Chapter 7, a useful rule of the thumb for splitting data is to produce a training set much bigger than the testing set. For example, an 80/20 split would produce 80% of the data as the training set and 20% as the testing set.

Next, one or more classifiers are trained over the training set to learn the characteristics or patterns residing in the data. The sentiment tags in the testing data are hidden away from the classifiers. After the training, classifiers are tested over the testing set to infer the sentiment tags. Finally, the result is compared against the original sentiment tags to evaluate the overall performance of the classifier.

The code that follows is written in Python using the Natural Language Processing Toolkit (NLTK) library (`http://nltk.org/`). It shows how to perform sentiment analysis using the naïve Bayes classifier over the movie review corpus.

The code splits the 2,000 reviews into 1,600 reviews as the training set and 400 reviews as the testing set. The naïve Bayes classifier learns from the training set. The sentiments in the testing set are hidden away from the classifier. For each review in the training set, the classifier learns how each feature impacts the outcome sentiment. Next, the classifier is given the testing set. For each review in the set, it predicts what the corresponding sentiment should be, given the features in the current review.

```
import nltk.classify.util
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import movie_reviews
from collections import defaultdict
import numpy as np

# define an 80/20 split for train/test
SPLIT = 0.8

def word_feats(words):
    feats = defaultdict(lambda: False)
    for word in words:
        feats[word] = True
    return feats

posids = movie_reviews.fileids('pos')
```

```
negids = movie_reviews.fileids('neg')

posfeats = [(word_feats(movie_reviews.words(fileids=[f])), 'pos')
            for f in posids]
negfeats = [(word_feats(movie_reviews.words(fileids=[f])), 'neg')
            for f in negids]

cutoff = int(len(posfeats) * SPLIT)

trainfeats = negfeats[:cutoff] + posfeats[:cutoff]
testfeats = negfeats[cutoff:] + posfeats[cutoff:]

print 'Train on %d instances\nTest on %d instances' % (len(trainfeats),
                                                       len(testfeats))

classifier = NaiveBayesClassifier.train(trainfeats)
print 'Accuracy:', nltk.classify.util.accuracy(classifier, testfeats)

classifier.show_most_informative_features()

# prepare confusion matrix

pos = [classifier.classify(fs) for (fs,l) in posfeats[cutoff:]]
pos = np.array(pos)
neg = [classifier.classify(fs) for (fs,l) in negfeats[cutoff:]]
neg = np.array(neg)

print 'Confusion matrix:'
print '\t'*2, 'Predicted class'
print '-'*40
print '|\t %d (TP) \t|\t %d (FN) \t| Actual class' % (
          (pos == 'pos').sum(), (pos == 'neg').sum()
print '-'*40
print '|\t %d (FP) \t|\t %d (TN) \t|' % (
          (neg == 'pos').sum(), (neg == 'neg').sum())
print '-'*40
```

The output that follows shows that the naïve Bayes classifier is trained on 1,600 instances and tested on 400 instances from the movie corpus. The classifier achieves an accuracy of 73.5%. Most information features for positive reviews from the corpus include words such as *outstanding*, *vulnerable*, and *astounding*; and words such as *insulting*, *ludicrous*, and *uninvolving* are the most informative features for negative reviews. At the end, the output also shows the confusion matrix corresponding to the classifier to further evaluate the performance.

```
Train on 1600 instances
Test on 400 instances
Accuracy: 0.735
Most Informative Features
```

```
            outstanding = True          pos : neg    =     13.9 : 1.0
              insulting = True          neg : pos    =     13.7 : 1.0
             vulnerable = True          pos : neg    =     13.0 : 1.0
              ludicrous = True          neg : pos    =     12.6 : 1.0
            uninvolving = True          neg : pos    =     12.3 : 1.0
             astounding = True          pos : neg    =     11.7 : 1.0
                 avoids = True          pos : neg    =     11.7 : 1.0
            fascination = True          pos : neg    =     11.0 : 1.0
              animators = True          pos : neg    =     10.3 : 1.0
                 symbol = True          pos : neg    =     10.3 : 1.0
Confusion matrix:
              Predicted class
---------------------------------------
|       195 (TP)     |        5 (FN)     | Actual class
---------------------------------------
|       101 (FP)       |     99 (TN)     |
---------------------------------------
```

As discussed earlier in Chapter 7, a ***confusion matrix*** is a specific table layout that allows visualization of the performance of a model over the testing set. Every row and column corresponds to a possible class in the dataset. Each cell in the matrix shows the number of test examples for which the actual class is the row and the predicted class is the column. Good results correspond to large numbers down the main diagonal (TP and TN) and small, ideally zero, off-diagonal elements (FP and FN). Table 9-7 shows the confusion matrix from the previous program output for the testing set of 400 reviews. Because a well-performed classifier should have a confusion matrix with large numbers for TP and TN and ideally near zero numbers for FP and FN, it can be concluded that the naïve Bayes classifier has many false negatives, and it does not perform very well on this testing set.

TABLE 9-7 *Confusion Matrix for the Example Testing Set*

		Predicted Class	
		Positive	Negative
Actual Class	Positive	195 (TP)	5 (FN)
	Negative	101 (FP)	99 (TN)

Chapter 7 has introduced a few measures to evaluate the performance of a classifier beyond the confusion matrix. Precision and recall are two measures commonly used to evaluate tasks related to text analysis. Definitions of precision and recall are given in Equations 9-8 and 9-9.

$$Precision = \frac{TP}{TP + FP} \tag{9-8}$$

$$Recall = \frac{TP}{TP + FN} \tag{9-9}$$