7.1 Decision Trees

A ***decision tree*** (also called ***prediction tree)*** uses a tree structure to specify sequences of decisions and consequences. Given input $X = \{x_1,x_2,...x_n\}$, the goal is to predict a response or output variable *Y*. Each member of the set $\{x_1,x_2,...x_n\}$ 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. 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.

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.*

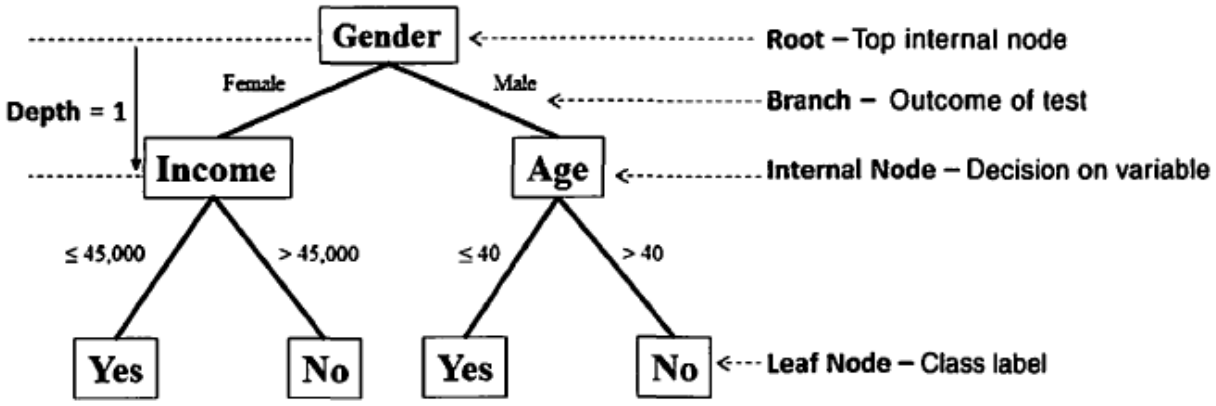

FIGURE 7-1 *Example of a decision tree*

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.

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.

Where decision tree is used?

- Decision trees are widely used in practice.
- To classify animals, questions (like cold-blooded or warm-blooded, mammal or not mammal) are answered to arrive at a certain classification.
- 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.

7.1.2-The General Algorithm of Decision Tree

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.

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... 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. 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.

$$H_x = -\sum_{\forall x \in X} P(x)\log_2 P(x) \quad (7\text{-}1)$$

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 tai ls 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).

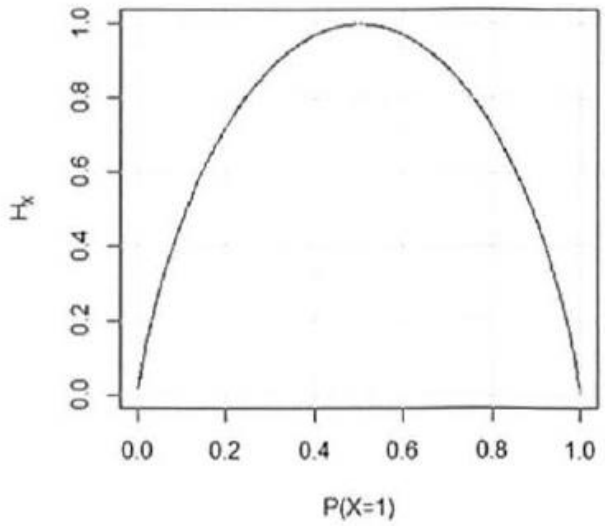

FIGURE 7-5 *Entropy of coin flips, where X=1 represents heads*

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.

$$H_{Y|X} = \sum_x P(x)H(Y|X = x)$$
$$= -\sum_{\forall x \in X} P(x) \sum_{\forall y \in Y} P(y|x)\log_2 P(y|x) \quad (7\text{-}2)$$

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.

$$InfoGain_A = H_S - H_{S|A} \quad (7\text{-}3)$$

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.

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.

ID3 Algorithm

ID3 (or Iterative Dichotomiser 3) 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)
```

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.

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. 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 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.

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. 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.

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 or use ensemble methods to combine several weak learners for better classification.

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.

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.

7.2-Naive Bayes

Naive Bayes is a probabilistic classification method based on Bayes' theorem. Bayes' theorem gives the relationship between the probabilities of two events and their conditional probabilities.

A naive 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.

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.* 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.

Because naive Bayes classifiers are easy to implement and can execute efficientl. Spam filtering is a classic use case of naive Bayes text classification. Bayesian spam filtering has become a popular mechanism to distinguish spam e-mail from legitimate e-mail.

Naive Bayes classifiers can also be used for fraud detection. 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, naive Bayes can provide probability- based classification of whether a new claim is genuine.

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 $P(A|C)$.

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.

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

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.

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 $P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i)$.

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.

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

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, naive Bayes classifiers can handle missing values. Naive 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. Naive Bayes is computationally efficient and is able to handle high-dimensional data efficiently. .In some cases naive Bayes even outperforms other methods. Unlike logistic regression, the naive 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 naive Bayes classifier overall performs better than decision trees on categorical values with many levels. Compared to decision trees, naive Bayes is more resistant to overfitting, especially with the presence of a smoothing technique.

Despite the benefits of naive Bayes, it also comes with a few disadvantages. Naive 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, naive 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, naive Bayes classifiers in general are not very reliable for probability estimation and should be used only for assigning class labels. Naive 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.

7.3-Diagnostics of Classifiers

Classifiers 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 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.

		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

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.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \qquad (7\text{-}18)$$

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 ***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 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. *Precision* is the percentage of instances marked positive that really are positive, as shown in Equation 7-22.

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

ROC curve 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.

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).

7.4-Additional Classification Methods

Besides the two classifiers introduced in this chapter, several other methods are commonly used for classification, including bagging, boosting, random forest, and support vector machines (SVM).

Bagging (or bootstrap aggregating) 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) 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 [S] than a decision tree.

Random forest 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 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.

8-Time Series Analysis

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. In this example, the time series consists of an ordered sequence of 144 values.

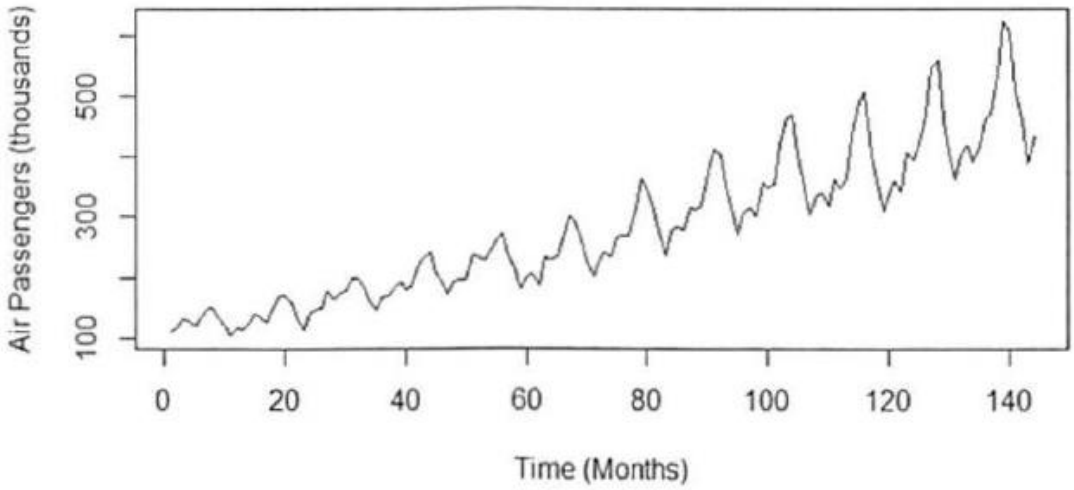

FIGURE 8-1 *Monthly international airline passengers*

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.
- **Spare parts planning:** Companies' service organizations have to forecast future spare part demands to ensure an adequate supply of parts to repair customer products. 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.
- **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.

8.1.1-Box-Jenkins Methodology

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 oran 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.

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.

The Box-Jenkins methodology for time series analysis involves the following three main steps:

1. Condition data and select a model.
    a. Identify and account for any trends or seasonality in the time series,
    b. 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.

8.2-ARIMA Model (Autoregressive Integrated Moving Average)

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 stationary time series is one whose properties do not depend on the time at which the series is observed. A time series, y_t, for t = 1,2,3,...,, 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, ...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.

$$cov(y_t , y_{t+h}) = E[(y_t - \mu_t)(y_{t+h} - \mu_{t+h})] \qquad (8.1)$$

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.

$$cov(h) = E[(y_t - \mu)(y_{t+h} - \mu)] \qquad (8.2)$$

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.

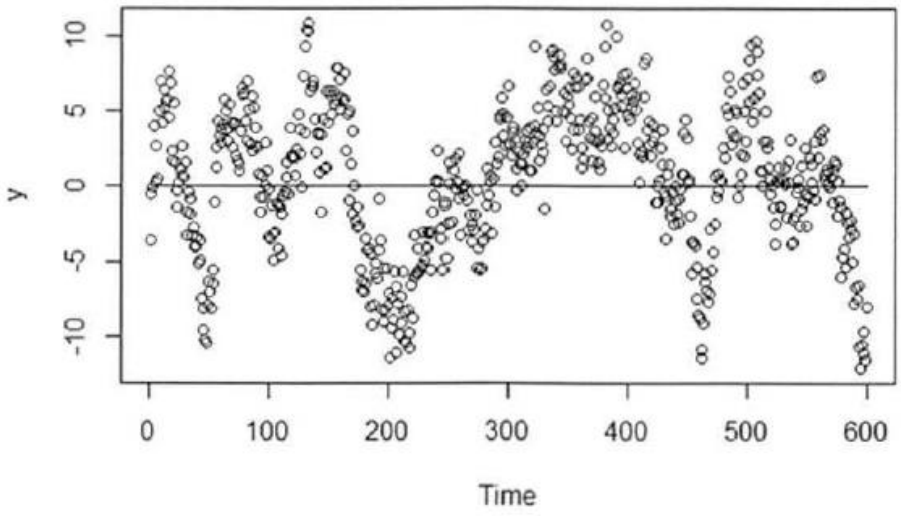

FIGURE 8-2 *A plot of a stationary series*

8.2.1-Autocorrelation Function (ACF)

The plot of *autocorrelation function (ACF)* provides insight into the covariance of the variables in the time series and its underlying structure. For a stationary time series, the ACF is defined as shown in Equation 8-4

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

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}. The plot of the ACF is provided in Figure 8-3 for stationary time series.

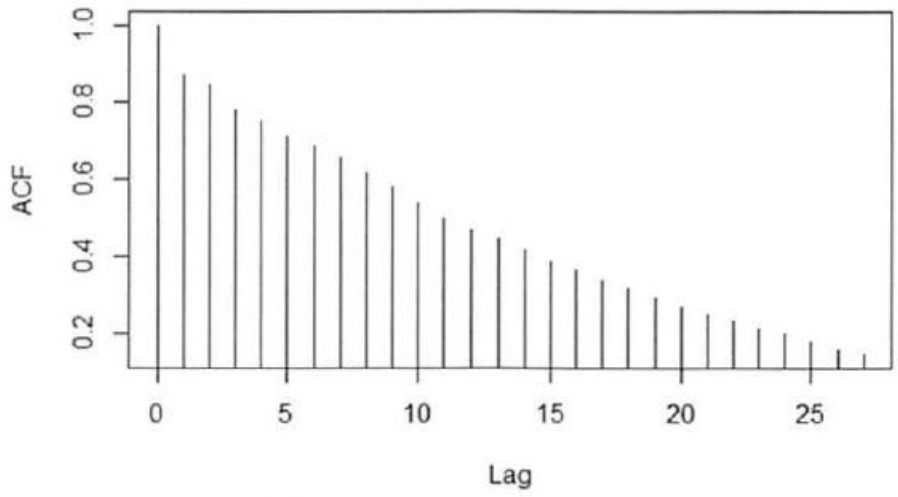

FIGURE 8-3 *Autocorrelation function (ACF)*

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, 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.

8.2.2-Autoregressive Models

An **autoregressive model** is when a value from a time series is regressed on previous values from that same time series.

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:

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

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.

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.

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

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

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} , 2, the partial correlation between y_t and y_{t+3} is relatively small.

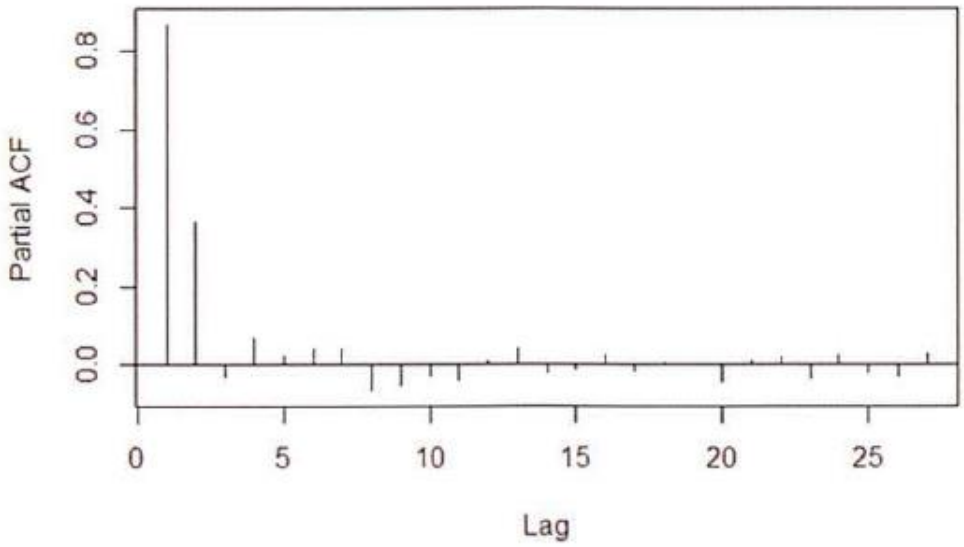

FIGURE 8-4 *Partial autocorrelation function (PACF) plot*

8.2.3-Moving Average Models

Fora 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.

$$y_t = \varepsilon_t + \theta_1\varepsilon_{t-1} + \ldots + \theta_q\varepsilon_{t-q} \quad (8.9)$$

Where θ_k is a constant for k = 1, 2…..,q
$\theta_q \neq 0$
$\varepsilon_t \sim N(0,\sigma^2_\varepsilon)$ 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. Fora simulated MA(3) time series of the form $y_t = \varepsilon_t + 0.4\varepsilon_{t-1} + 1.1\varepsilon_{t-2} - 2.5\varepsilon_{t-3}$ $\varepsilon_t \sim N(0,1)$, Figure 8-5 provides the scatterplot of the simulated data overtime.

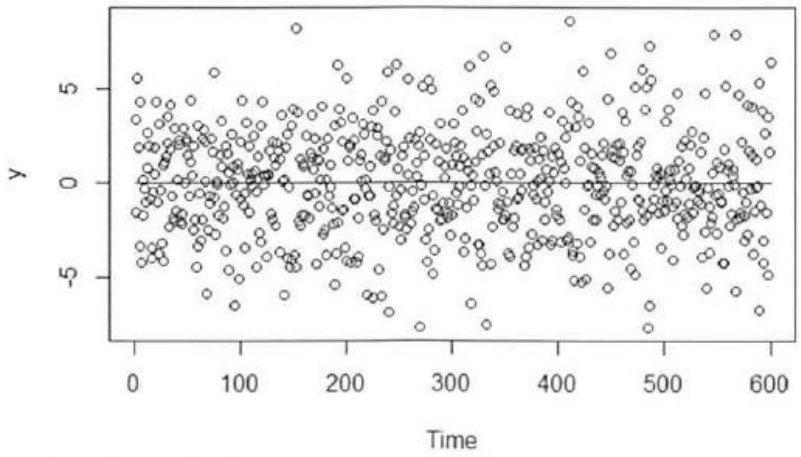

FIGURE 8-5 *Scatterplot of a simulated MA(3) time series*

Figure 8-6 provide; the ACF plot for the simulated data. Again, the ACF(O) 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.

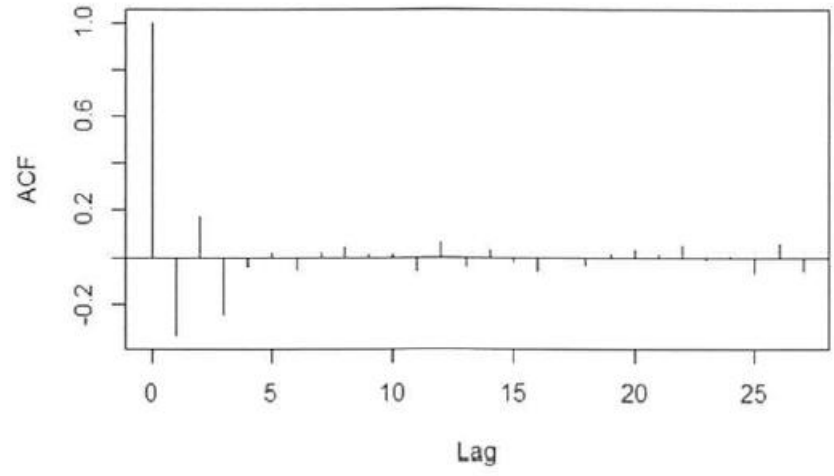

FIGURE 8-6 *ACF plot of a simulated MA(3) time series*

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 $p \neq 0$ and q=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. 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

$$d_t = y_t - y_{t-1} \text{ for } t=2,3,\ldots..,n \quad (8.16)$$

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.

8.2.5 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. 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.

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.

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.

8.3 Additional Methods

Additional time series methods include the following:

- **Autoregressive Moving Average with Exogenous inputs (ARM AX)** 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.
- **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

9-Text 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. 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 (N LP). Another major challenge with text analysis is that most of the time the text is not structured.

9.1-Text Analysis Steps

A text analysis problem usually consists of three important steps:

- Parsing
- Search and Retrieval
- Text Mining.

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 Hyper Text 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 such as clustering and classification, can be adapted to text mining. For example, the k-means can be modified to cluster text documents into groups, where each group represents a collection of documents with a similar topic. 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 naive Bayes. 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.

9.2-A Text Analysis Example

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 Consumer Reports. 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 of text analysis. This process is illustrated in Figure 9-1, and it includes following the modules.

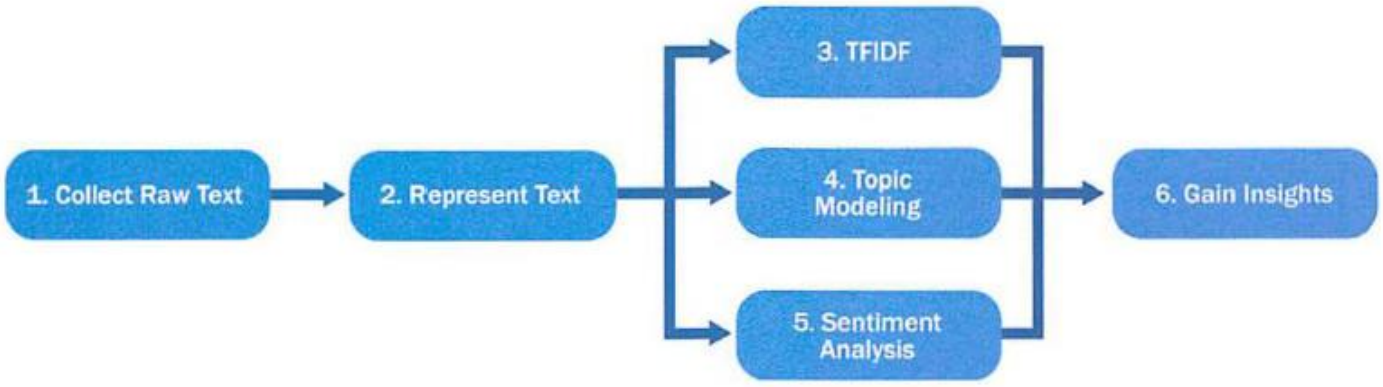

FIGURE 9-1 *ACME's Text Analysis Process*

1. **Collect raw text-** 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-** 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. This and the following two steps correspond to Phases 3 through 5 of the Data Analytic Lifecycle.
4. **Categorize documents by topics-** This can be achieved through topic models (such as latent Dirichlet allocation).
5. **Determine sentiments of the reviews-**. 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.
6. **Review the results and gain greater insights**- 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.

9.3-Collecting Raw Text

In Data Analytic 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 ibPhone 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 for third-party developers to access their data. For example, the Twitter API 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.

Many news portals and blogs provide data feeds that are in an open standard format, such as RSS or XML.

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.

The team can then construct the web scraper based on the identified patterns. The scraper can use the curl tool to fetch HTML source code given specific URLs, use XPath 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. 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.

9.4 Representing Text

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 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)

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.

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?

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. 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.

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.

Bag-of-words takes quite a naive 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.

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.

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.

It is important not only to create a representation of a document but also to create a representation of a corpus. 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.

9.5 Term Frequency-Inverse Document Frequency (TFIDF)

TFIDF is 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…t_n) containing *n* terms, the simplest form of term frequency of r in dean be defined as the number of times f 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

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

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 das shown in Equation 9-3.

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

Indeed, that is the intention of them verted *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 $D = [dyd_2,...d_N3$ 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

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

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.

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 dis defined as the term frequency of *t* in *d* multiplying the document frequency of *t* in the corpus as shown in Equation 9-7:

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

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.

9.6-Categorizing Documents by Topics

A topic consists of a cluster of words that frequently occur together and share the same theme. Document grouping can be achieved with clustering methods such as k-means clustering or classification methods such as support vector machines, or naive Bayes, 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 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, 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.

The simplest topic model is latent Dirichlet allocation (LDA), 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 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.

Many programming tools provide software packages that can perform LDA over datasets. R comes with an *Ida* package that has built-in functions and sample datasets.

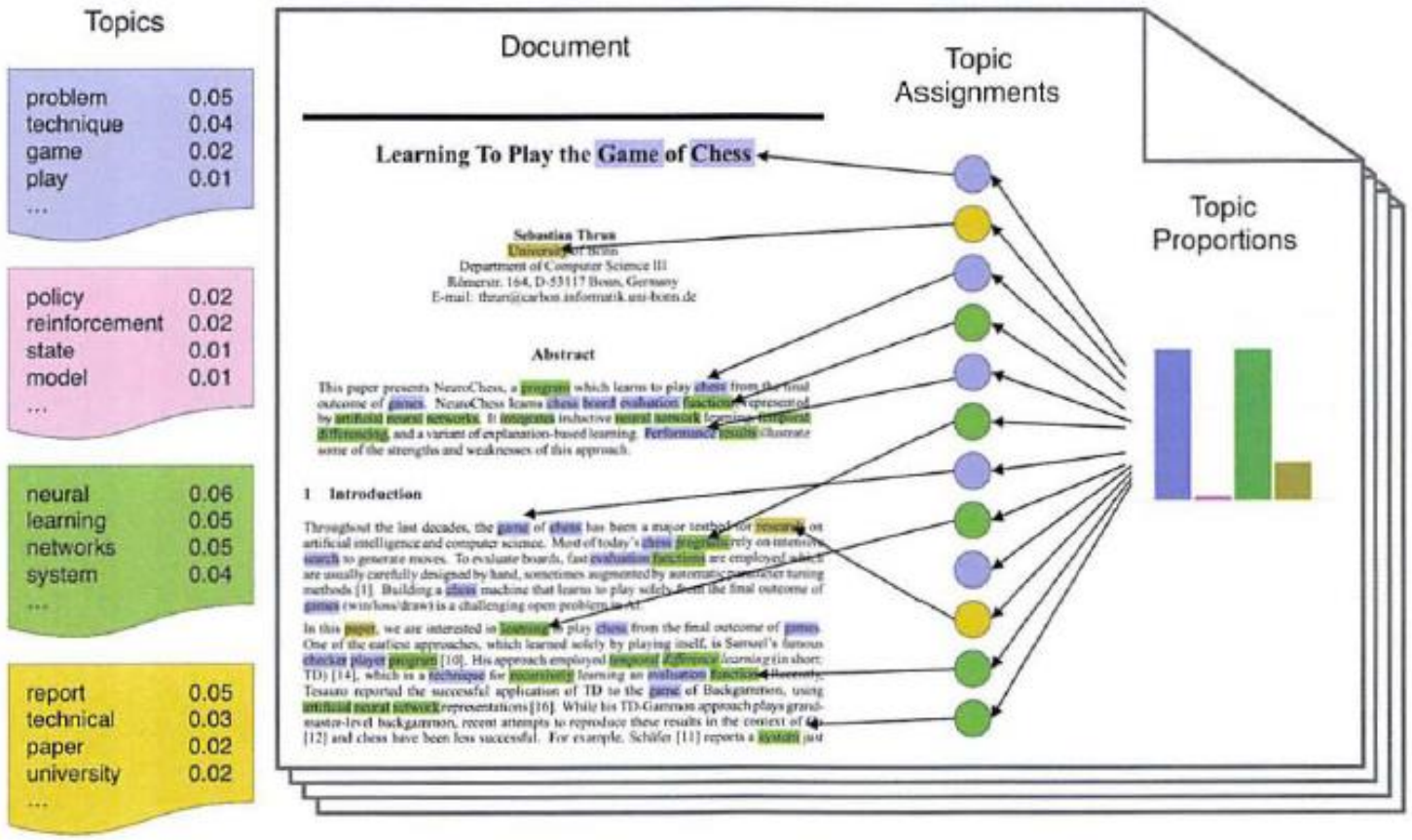

FIGURE 9-4 *The intuitions behind LDA*

9.7-Determining Sentiments

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.

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%, and it is likely to be outperformed by examination of corpus statistics.

Classification methods such as naive Bayes, 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 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.

Depending on the classifier, the data may need to be split into training and testing sets. One way 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.

Classifiers determine sentiments solely based on the datasets on which they are trained. The domain of the datasets and the characteristics of the features determine what the knowledge classifiers can learn. For example, lightweight is a positive feature for reviews on laptops but not necessarily for reviews on wheelbarrows or textbooks. In addition, the training and the testing sets should share similar traits for classifiers to perform well. For example, classifiers trained on movie reviews generally should not be tested on tweets or blog comments.

Note that an absolute sentiment level is not necessarily very informative. Instead, a baseline should be established and then compared against the latest observed values. For example, a ratio of 40% positive tweets on a topic versus 60% negative might not be considered a sign that a product is unsuccessful if other similar successful products have a similar ratio based on the psychology of when people tweet.