of a variable using greater than and less than operations. For text, we know in advance that the values will either be binary or ternary.

Based on experience, we favor using local dictionaries of only a few hundred words. The most frequent words are adequate, and the stopwords should be removed. In many benchmarks for text categorization and decision rules, the results of learning with much larger dictionaries do not improve. Some additional speedups may be achieved in the pruning steps. Instead of considering all words in a phrase for pruning, only the last word in a phrase may be examined. The last word in a phrase is typically the most specialized, and the preceding words are often more predictive. The accompanying software includes an implementation of these concepts.

So far, we have seen the similarity measures of the nearest-neighbor methods that require the least effort in learning or assembling a sample, and we have seen decision rules that may take more time to learn but may still be preferred because they are more intuitive and often more accurate. Next, we look at the weighted-scoring methods that have an edge in learning speed or predictive accuracy.

### 3.4.4 Scoring by Probabilities

The most obvious method of classification is direct lookup of the probabilities of words in a document. Let $C$ be the class label we are interested in and $x$ be a feature vector that denotes the presence or absence of words from a dictionary. Mathematically, the objective is to estimate $Pr(C|x)$, the probability of a class, given the presence or absence of words from a dictionary. For singly labeled document collections, we can choose the category $C$ that has the largest probability score $Pr(C|x)$. For multiply labeled document collections, if our interest is to maximize the accuracy, then $C$ is selected whenever $Pr(C|x)$ is greater than 0.5. Another way to look at the multiply labeled case is that for each label we divide the document collection into two classes: one class with label $C$ and the other class with a label that is not $C$. Therefore, we have a binary classification problem for each label value $C$. This is reasonable because the multiple labels assigned to documents are usually independent of each other, and hence it is possible to view each label assignment as a separate classification problem with two classes (labeled and not labeled). It thus suffices to consider the binary class problem.

However, we know that, even for this problem, a complete computation of probability is impossible. Even a 100 word dictionary has $2^{100}$ possible combinations. Still, a simplified approach to probability estimation, called **Bayes with independence or naive Bayes**, has often been attempted. The mathematics is straightforward and the computation is efficient, which leads to wide application of this approach, especially in applications where a quick implementation takes priority over accuracy.

$$Pr(C|x) = Pr(x|C) * Pr(C)/Pr(x). \quad (3.2)$$

Bayes' rule is given in Equation (3.2), where $C$ is the class of interest and $x$ is a vector of ones and zeros corresponding to the presence or absence of dictionary words for a specific document.

Note that when applying Equation (3.2) for two or more classes, the common factor of $1/Pr(x)$ does not change the relative ranking of $Pr(C|x)$. Therefore, it does not need to be computed explicitly for ranking purposes. If it is dropped from the comparisons, one does not have an explicit probability estimate, only the numerator. However, it might be useful to compute $Pr(x)$ since it allows one to compute a probability estimate. Using probability estimates, one can adjust the kinds of errors made and also specify a **reject probability threshold** that must be exceeded to classify. When there are two classes, $C_1$ and $C_2$, $Pr(x)$ is readily computed as in Equation (3.3).

$$Pr(x) = Pr(x|C_1)Pr(C_1) + Pr(x|C_2)Pr(C_2). \quad (3.3)$$

The key to using these equations is to compute $Pr(x|C)$. If we assume that the words are independent then instead of looking up the probability of the complete vector of $x$, we can look up the probability of the presence or absence of each word, $Pr(x_j|C)$, and multiply them all together. We use $x_j$ to denote the $j$-th component of $x$. Equation (3.4) states this mathematically.

$$Pr(x|C) = \prod_j Pr(x_j|C), \quad Pr(x) = \sum_C Pr(C) \prod_j Pr(x_j|C). \quad (3.4)$$

The conditional probabilities in Equation (3.4) are readily estimated if one uses the simple binary presence or absence of a word as a feature value that would give only two possible values for each feature.

The probability estimates are easy to obtain from our spreadsheet. $Pr(C)$ is determined from the frequency of ones in the last column divided by $n$, the number of examples, $freq(C)/n$. Each $x_j$ is either a 1 or a 0 (presence or absence of the word $w_j$). The quantity $Pr(x_j = 1|C)$ is computed from the frequency of ones for $x_j$, where only the examples la-
beled $C$ are considered, $\text{freq}(x_j = 1, \text{label} = C)/\text{freq}(C)$. The probability of $w_j$ not occurring in $C$, $\text{Pr}(x_j = 0 | C)$, is $1 - \text{Pr}(x_j = 1 | C)$.

Figure 3.16 is an example for a dictionary of four words. The training sample consists of ten documents of which four are labeled as $\text{Class} = 1$ and the remaining six as $\text{Class} = 0$. We can easily compute estimates of the various conditional probabilities as shown in the figure. Now suppose we get a new document $D$ that has $w_2$, $w_3$, and $w_4$. Then, for the positive class, we could compute $D$'s probability as

$$\text{Pr}(\text{Class} = 1 | D) = (1 - .75) \times .25 \times .5 \times .5 \times .6 / \text{Pr}(D) = .00625 / \text{Pr}(D).$$

For the negative class, the probability would be computed as

$$\text{Pr}(\text{Class} = 0 | D) = (1 - .5) \times .67 \times .33 \times .5 \times .6 / \text{Pr}(D) = .03333 / \text{Pr}(D),$$

and as a result the document $D$ would be labeled as $\text{Class} = 0$ (if one computes $\text{Pr}(D)$, one gets a probability of 0.84 for the classification).

The performance on text benchmark applications for naive Bayes is usually weaker than for the other methods described in this chapter. Still, it requires almost no memory and little computation, so it does have its advocates. It usually works best with a relatively small dictionary representing the key words needed to make a decision for that class. An implementation is provided in the accompanying software.

The naive Bayes method of estimating probabilities looks complex, but in fact it has a linear structure. This can be seen by noting that, given a binary feature vector $x$, the probability score of class $C$ is

$$\text{Pr}(C | x) = \frac{\text{Pr}(C)}{\text{Pr}(x)} \prod_{j} \left( \frac{\text{Pr}(x_j = 1 | C) \times x_j}{\text{Pr}(x_j = 0 | C)} \right).$$

which can be rewritten as

$$\text{Pr}(C | x) = \frac{1}{\text{Pr}(x)} \exp \left( \sum_{j} w_j x_j + b \right),$$

where

$$w_j = \ln \frac{\text{Pr}(x_j = 1 | C)}{\text{Pr}(x_j = 0 | C)}, \quad b = \ln \text{Pr}(C) + \sum_{j} \ln \text{Pr}(x_j = 0 | C).$$

This formulation, often called the multivariate Bernoulli model, allows the formulation of another naive Bayes model, referred to as the multinomial model, by replacing the linear weights formula in Equation (3.6) with the one in Equation (3.7). Here $n$ is the number of examples and $m$ is the number of features.

$$w_j = \ln \frac{\text{freq}(x_j = 1, \text{label} = C)}{\text{freq}(x_j = 0, \text{label} = C)}, \quad b = \ln \frac{\text{freq}(\text{label} = C)}{n}.$$

The multinomial model is frequently used in text categorization applications. It normalizes the length of a document, which often leads to slightly better performance. The parameter $\lambda > 0$ is a smoothing parameter, often set to 1 in the literature. However, we find that a smaller value such as 0.01 can sometimes be more effective.

In these forms, it is easier to see that one might also use other methods to directly train the linear weights. We will examine linear scoring methods in the next section.

### 3.4.5 Linear Scoring Methods

In order to achieve good prediction performance, it is often necessary to create a feature vector of very high dimension. Although many of the features are not useful, it can be difficult for a human to tell what feature is useful and what feature is not. Therefore, the prediction algorithm should have the ability to take a large set of features and then select only useful features from the full set. A very useful method to achieve this is by using linear scoring.

The naive Bayes method described above can be regarded as a special case of the linear scoring method. This can be seen clearly from Equation (3.5). However, the performance can be significantly improved using more sophisticated training methods to obtain the weight vector $w = [w_j]$ and bias $b$.

Consider the problem of distinguishing between two classes. The general scoring method is to assign a positive score to predict the posi-
Figure 3.17. Computing the Weighted Score of a Document

tive class and a negative score to predict the negative class. Figure 3.17 illustrates an example of using a set of weights to determine the score for a document. For all words that occur in a document, we find their corresponding weights. These weights are then summed to determine the document’s score.

Mathematically, this method is a linear scoring function. The general form is in Equation (3.8), where \( D \) is the document and \( w_j \) is the weight for the \( j \)-th word in the dictionary, \( b \) is a constant, and \( x_j \) is a one or zero, depending on the \( j \)-th word’s presence or absence in the document.

\[
Score(D) = \sum_j w_j x_j + b = w \cdot x + b. \tag{3.8}
\]

Linear scoring methods are classical approaches to solving a prediction problem. The weaknesses of this method are well-known. Geometrically, the method can be described as producing a line or hyperplane. Although a line cannot fit complex surfaces, and a curvy shape might be needed, it is often possible to create appropriate nonlinear features so that a curve in the original space lies in a hyperplane in the enlarged space with the additional nonlinear features. In this way, nonlinearity can be explicitly captured by constructing sophisticated nonlinear features. An advantage of this approach is that the modeling aspect becomes conceptually very simple since we can focus on creating useful features and let the learning algorithm determine how to assign a weight to each feature we create. Another advantage is that the linear scoring method can efficiently handle sparse data. This is important for text-mining applications since although feature vectors can have high dimensionality, they are usually very sparse.

We know from various benchmarks that the linear scoring approach does surprisingly well on text classification, to the point where it rivals the best results in some tasks. The modern approach to learning the weights is not the same as the classical statistical methods. The simple naive Bayes methods have severe problems with redundant attributes, which in text corresponds to words that behave like synonyms. Classical methods were developed to handle a small number of attributes, certainly not the tens of thousands of words in a global dictionary. The newer linear methods are oblivious to these limitations. A major advance in linear methods for text has been their ability to work with huge dictionaries and find weights for every word in a complete dictionary. If there are ten synonyms, it can weigh each one. This capability to work with so many words and weigh all of them both positively and negatively seems to capture the subtleties of language, where some words are precise and strong predictors and others are vague and weak predictors.

Surely, isn’t the computational time for learning in this high-dimensional space prohibitive? Not at all. The same problem that cannot be solved by a classical method can now be solved incredibly quickly, much faster than by nearest-neighbors similarity methods or the decision rules. Moreover, the natural extension for the linear model is not to find more complex mathematical functions. Instead, scoring might be extended by adding word phrases to the single-word dictionaries. Benchmarks using more complex scoring methods generally perform no better than the linear scores.

The key problem with these weighted-scoring methods is that of learning the weights, the second column in Figure 3.17. The words are those in the dictionary, and the weights for them will be learned from a collection of documents. The label is assigned by applying Equation (3.8). How do we learn the weights? Implementations can consist of less than 200 lines of code. However, the method is a mathematical process, an application of numerical analysis.

### 3.4.5.1 How to Find the Best Scoring Model

So how do we learn the weights? To determine the most efficient way, the treatment is necessarily mathematical. It may look complex, but the implementation in software is straightforward. The representation of the words can be binary, but the tf-idf transformation usually yields better results.

Let us first look at the mathematics behind the procedure. We consider a two-class prediction problem to be one that determines a label