CHAPTER 5

SUPERVISED LEARNING

Organizing human knowledge into related areas is nearly as old as human knowledge itself, as is evident in writings from many ancient civilizations. In modern times, the task of organizing knowledge into systematic structures is studied by ontologists and library scientists, resulting in such well-known structures as the Dewey decimal system, the Library of Congress catalog, the AMS Mathematics Subject Classification, and the U.S. Patent Office subject classification [11, 68]. Subject-based organization routinely permeates our personal lives as we organize books, CDs, videos, and email.

The evolution of the Web has followed this familiar history. Around the same time as ad hoc keyword search engines like AltaVista became popular, the Yahoo! (*www.yahoo.com*) topic directory was launched. Since then, many other Web catalogs have appeared. The Open Directory Project (*dmoz.org*) and About.com are some of the best known at present.

Topic directories are some of the most popular sites on the Web. There is a wealth of information in the manner in which humans have assigned structure to an otherwise haphazard collection of Web pages. Earlier, directories were used mainly for browsing, assisted with some simple search capability on the few lines of description associated with resource links, not the contents of the external pages themselves. Of late, well-known search portals such as Google (*www.google.com/*) return with search responses the "topic path" of a response (such as `/Arts/Painting`), if the response URL has been associated with one or more topics in the Open Directory. Because the Open Directory is manually

maintained, it does not capture all URLs; therefore only a fraction of Google responses are tagged with topics.[1]

Topic tagging improves the search experience in many ways. They are a great warning for queries gone astray or ambiguous queries [48], and they are an indirect means for finding similar documents. Whereas most "find-similar" utilities look for pairwise syntactic similarities with the query document (see Section 3.3.1), a topic-based search first maps the query document to a class (or a few classes), thus greatly enhancing the vocabulary. Then it finds similar documents with respect to this enhanced vocabulary. Topic tagging can also be used to assist hierarchical visualization and browsing aids [101].

News tracking provides another example of the utility of detecting predefined topics in text. Most online news sites provide tools to customize "front pages" as per reader taste. URL- or keyword-based selection is often inadequate, for the same reasons that make keyword-based searching imperfect. Systems have been built to capture URL clicks and to use them to report similar pages. Further applications to topic tagging can be found in organizing email [1] and bookmarks by content [140, 141].

Assigning topic labels to documents is but one of many general uses of supervised learning for text. Text classification has been used to narrow down authors of anonymous documents by learning the writing style (stylometry), as with the *Federalist Papers* written by Alexander Hamilton, John Jay, and James Madison and published anonymously under the pen name Publius [156]. The Flesch-Kincaid index is a hand-tuned "classifier" of sorts, combining the number of syllables per word and number of words per sentence into an index of difficulty of reading technical documents [78]. Yet another application is to classify the purpose of hyperlinks. Documents are connected via hyperlinks and citations for a variety of reasons, including elaboration, citation of supporting material, critique, and so on. Machine learning can be used to guess the (main) purpose of creating a hyperlink.

5.1 The Supervised Learning Scenario

Library scientists undergo extensive training to be able to tag publications with correct and comprehensive topic labels. Can a computer program attain even a fraction of their learning capability? This is an important question in view of the

1. It is possible that Google also uses automatic classification to some extent.

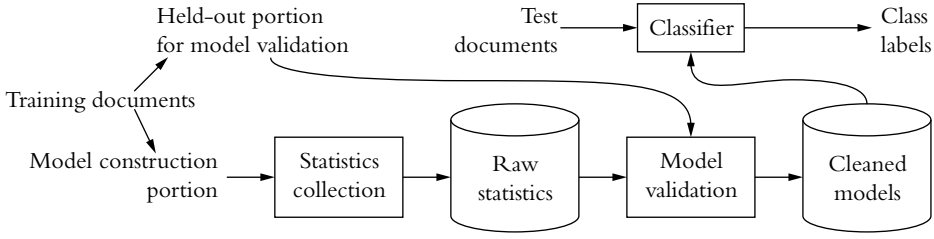

FIGURE 5.1 A typical supervised text-learning scenario.

growing volume of the Web, together with its vastly reduced editorial control and resulting diversity. Learning to assign objects to classes given examples is called *classification* or *supervised learning*, and is the subject of this chapter.

In supervised learning, the *learner* (also called the *classifier*) first receives training data in which each item (document or Web page, in our setting) is marked with a label or class from a discrete finite set. (Sometimes these labels may be related through a taxonomic structure, such as a hierarchical topic catalog. Except in Section 5.7, we will be largely concerned with "flat" sets of class labels.) The learning algorithm is trained using this data. It is common to "hold out" part of the labeled data to tune various parameters used in the classifier. Once the classifier is trained, it is given unlabeled "test" data and has to guess the label. Figure 5.1 illustrates the process at a high level.

Supervised learning has been intensively studied for several decades in AI, machine learning, and pattern recognition, and of late in data warehousing and mining. In those domains, the data is usually more "structured" than text or hypertext. Structured data usually comes in relational tables with well-defined data types for a moderate number of columns. For example, many data sets in the well-known U.C. Irvine repository [20] have between 5 and 50 features. Furthermore, the semantic connection between these columns and the class label is often well understood, at least qualitatively; for example, smoking and cancer risk, age and rash driving, or income and credit card fraud.

In this chapter, our goal is to study supervised learning specifically for text and hypertext documents. Text, as compared to structured data, has a very large number of potential features, of which many are irrelevant. If the vector-space model is used, each term is a potential feature. Furthermore, there are many features that show little information content individually, but in conjunction are vital inputs for learning.

Unlike structured tables with a uniform number of columns[2] per instance, documents can have a diverse number of features. There is little hope of precisely characterizing the joint distribution of the relevant features, owing to sparsity of data as well as computational limitations. The number of distinct class labels is much larger than structured learning scenarios. Finally, the classes may be related by hierarchical relationships, commonly seen in topic directories on the Web.

The models that we shall study in this chapter, although mostly restricted to text alone, will form building blocks for more complex models that couple hyperlink structure with topics, discussed in Chapters 6 and 7. Hypertext classification is at the core of many resource discovery systems that start from pages related to a specified topic and locate additional relevant pages. We will study such systems in Chapters 7 and 8. Supervised learning and its variants are also used for extracting structured snippets of information from unstructured text, which we will discuss in Chapter 9.

5.2 Overview of Classification Strategies

I will present a number of techniques for text classification and comment on their features, strengths, and weaknesses.

Given a typical IR system based on vector-space similarity (see Chapter 3), it is very easy to build a *nearest neighbor* (NN) classifier. An NN classifier (Section 5.4) simply indexes all the training set documents, remembering their class labels. A test document is submitted as a query to the IR system, and the distribution of labels on the training documents most similar to it are used to make a decision.

The vector-space model assigns large weights to rare terms, without regard to the frequency with which terms occur across documents from different classes. The process of *feature selection* (Section 5.5) removes terms in the training documents that are statistically uncorrelated with the class labels, leaving behind a reduced subset of terms to be used for classification. Feature selection can improve both speed and accuracy.

Next we study *Bayesian classifiers* (Section 5.6), which fit a generative term distribution $\Pr(d|c)$ (see Section 4.4.1) to each class c of documents $\{d\}$. While testing, the distribution most likely to have generated a test document is used to label it. This is measured as $\Pr(c|d)$ and derived from $\Pr(d|c)$ using Bayes's rule.

---

2. To be sure, structured tabular data may have entries such as "null," "unknown," or "not applicable," but these are usually modeled as categorical attribute values.

Another approach is to estimate a *direct* distribution $\Pr(c|d)$ from term space to the probability of various classes. *Maximum entropy classifiers* (Section 5.8) are an example of this approach. We may even represent classes by numbers (for a two-class problem, -1 and $+1$, say) and construct a direct function from term space to the class variable. *Support vector machines* (Section 5.9.2) are an example of this approach.

For hypertext applications, it is sometimes necessary to assemble features of many different kinds into a document representation. We may wish to combine information from ordinary terms, terms in titles, headers and anchor text, the structure of the HTML tag-tree in which terms are embedded, terms in pages that are link neighbors of the test page, and citation to or from a page with a known class label, to name a few. We will discuss *rule induction* over such diverse features in Section 5.10.2.

As with ad hoc query processing in IR systems, care with tokenization and feature extraction may be important for classification tasks. For example, replacing monetary amounts, four-digit years, and time in the form "`hh:mm`" by a special token for each type of string has been known to improve accuracy. For words that can be associated with multiple senses or parts of speech, we have seen part-of-speech or sense disambiguation improve accuracy slightly. In another application, abbreviation of phrases was key: for example, some documents mentioned "mild steel" while others used "M.S." or "M/S." In a different context, "M.S." may be mistaken for an academic degree. Clearly, designing a suitable token representation for a specific classification task is a practiced art. Automating feature extraction and representation for specific tasks is an interesting area of research.

5.3 Evaluating Text Classifiers

There are several criteria to evaluate classification systems:

- Accuracy, the ability to predict the correct class labels most of the time. This is based on comparing the classifier-assigned labels with human-assigned labels.
- Speed and scalability for training and applying/testing in batch mode.
- Simplicity, speed, and scalability for document insertion, deletion, and modification, as well as moving large sets of documents from one class to another.
- Ease of diagnosis, interpretation of results, and adding human judgment and feedback to improve the classifier.

Ideally, I would like to compare classifiers with regard to all of these criteria, but simplicity and ease of use are subjective factors, and speed and scalability change with evolving hardware. Therefore, I will mainly focus on the issue of accuracy, with some comments on performance where appropriate.

5.3.1 Benchmarks

The research community has relied on a few labeled collections, some of which have by now become de facto benchmarks. I describe a few of them below.

Reuters: The Reuters corpus has roughly 10,700 labeled documents with 30,000 terms and 135 categories. The raw text takes about 21 MB. There is a predefined division of the labeled documents into roughly 7700 training and 3000 test documents. About 10% of the documents have multiple class labels. It appears that a document's membership in some of the classes is predicated simply on the occurrence of a small, well-defined set of keywords in the document.

OHSUMED: This corpus comprises 348,566 abstracts from medical journals, having in all around 230,000 terms and occupying 400 MB. It is mostly used to benchmark IR systems on ad hoc queries, but it can also be used for classification. Each document is tagged with one or more *Medical Subject Headings* (MeSH) terms from a set of over 19,000 MeSH terms, which may be regarded as labels.

20NG: This corpus has about 18,800 labeled Usenet postings organized in a directory structure with 20 topics. There are about 94,000 terms. The raw concatenated text takes up 25 MB. The labeled set is usually split randomly into training and test sets, with, say, 75% chosen as training documents. The class labels are in a shallow hierarchy with five classes at the first level and 20 leaf classes.

WebKB: The WebKB corpus has about 8300 documents in 7 categories. About 4300 pages on 7 categories (faculty, project, and the like) were collected from four universities, and about 4000 miscellaneous pages were collected from other universities. For each classification task, any one of the four university pages are selected as test documents and the rest as training documents. The raw text is about 26 MB.

Industry: This is a collection of about 10,000 home pages of companies from 105 industry sectors (e.g., `advertising`, `coal`, `railroad`, `semiconductors`, etc.). The industry sector names are the class labels. There is a shallow hierarchy over

the labels. The first level has about 80 classes, and there are 105 leaves. The labeling is published on *www.marketguide.com*.

5.3.2 Measures of Accuracy

Depending on the application, one of the following assumptions is made:

- Each document is associated with *exactly one* class.
- Each document is associated with a *subset* of classes.

For most topic-based applications, the total number of classes is usually more than two. This is not a problem in the "exactly one" scenario. In this setting, a *confusion matrix M* can be used to show the classifier's accuracy. Entry $M[i, j]$ gives the number of test documents belonging to class i that were assigned to class j by the classifier. If the classifier were perfect, only diagonal elements $M[i, i]$ would be nonzero. If M is large, it is difficult to evaluate a classifier at a glance, so sometimes the ratio of the sum of diagonals to the sum of all elements in the matrix is reported as an accuracy score. The closer this ratio is to 1 the better the classifier.

To avoid searching over the power set of class labels in the "subset" scenario, many systems create a two-class problem for every class. For example, if the original data specified a class *Sports*, a classifier with classes *Sports* and *Not-sports* would be created. Documents labeled *Sports* would be examples of the positive class; all other documents would be examples of the negative class *Not-sports*. A test document would be submitted to all these classifiers to get a class subset. This is also called the *two-way ensemble* or the *one-vs.-rest* technique.

Ensemble classifiers are evaluated on the basis of *recall* and *precision,* similar to ad hoc retrieval (see Chapter 3). Let test document d be hand tagged[3] with a set of classes C_d, and suppose the classifier outputs its estimated set of classes C'_d. Here $C_d, C'_d \subseteq \mathcal{C}$, the universe of class labels.

The recall for class c is the fraction of test documents hand tagged with c that were also tagged with c by the classifier. The precision for class c is the fraction of test documents tagged with c by the classifier that were also hand tagged with c. As in ad hoc retrieval, there is a trade-off between recall and precision.

---

3. By "hand tagged," I mean that these labels are the "ground truth" against which the classifier is evaluated.

Classifier for c_1

		Guess	
		$\bar{c}_1$	c_1
True	$\bar{c}_1$	70	10
	c_1	5	15

Classifier for c_2

		Guess	
		$\bar{c}_2$	c_2
True	$\bar{c}_2$	40	20
	c_2	14	26

Classifier for c_3

		Guess	
		$\bar{c}_3$	c_3
True	$\bar{c}_3$	61	9
	c_3	19	11

Precision $P_1 = 15/(15+10)$ $P_2 = 26/(26+20)$ $P_3 = 11/(11+9)$

Recall $R_1 = 15/(15+5)$ $R_2 = 26/(26+14)$ $R_3 = 11/(11+19)$

Microaveraged precision: $\frac{15+26+11}{(15+10)+(26+20)+(11+9)}$

Microaveraged recall: $\frac{15+26+11}{(15+5)+(26+14)+(11+19)}$

Macroaveraged precision: $\frac{1}{3}(P_1 + P_2 + P_3)$

Macroaveraged recall: $\frac{1}{3}(R_1 + R_2 + R_3)$

FIGURE 5.2 How to evaluate the accuracy of classifiers. "True" is the hand-assigned class label any; "Guess" is the classifier output. See text for details.

Here is a simple notation to understand recall and precision in a precise manner. For each c and each d, we define a 2×2 contingency matrix $M_{d,c}$, as follows (the expression $[E]$ means 1 if the predicate E is true and 0 otherwise):

$$\begin{aligned} M_{d,c}[0,0] &= [c \in C_d \text{ and classifier outputs } c] \\ M_{d,c}[0,1] &= [c \in C_d \text{ and classifier does not output } c] \quad \text{(loss of recall)} \\ M_{d,c}[1,0] &= [c \notin C_d \text{ and classifier outputs } c] \quad \text{(loss of precision)} \\ M_{d,c}[1,1] &= [c \notin C_d \text{ and classifier does not output } c] \end{aligned} \tag{5.1}$$

Thus for each (d, c), $M_{d,c}$ has exactly one nonzero entry out of four.

The *microaveraged* contingency matrix is defined as $M_\mu = \sum_{d,c} M_{d,c}$. The microaveraged *recall* is defined as $\frac{M_\mu[0,0]}{M_\mu[0,0]+M_\mu[0,1]}$. The microaveraged *precision* is defined as $\frac{M_\mu[0,0]}{M_\mu[0,0]+M_\mu[1,0]}$. All this is exactly analogous to ad hoc recall and precision. Consider a three-class problem. For each class $c = c_1, c_2, c_3$, we train one classifier with classes c and $\bar{c}$. Let the total number of test documents be 100, and suppose the three classifiers perform as shown Figure 5.2. The micro- and macroaveraged recall and precision are shown in the same figure.

Microaveraging makes the overall precision and recall depend most on the accuracy observed for the classes with the largest number of (positive) documents: the accuracy can be poor for classes with few positive examples without affecting

the overall numbers much. One may also look at the data aggregated by specific classes, $M_c = \sum_d M_{c,d}$. This will give the recall and precision for each class separately. Suppose M_c is scaled so the four entries add up to one, giving M'_c. The *macroaveraged* contingency matrix can be defined as $(1/|C|) \sum_c M'_c$. The macroaveraged recall and precision can then be defined in the usual way. Macroaveraged measures pay equal importance to each class, whereas microaveraged measures pay equal importance to each document.

For most classifiers, various parameters can be tuned so that the set of classes returned for a test document may be made to trade off recall for precision or vice versa. It is common to report classifier performance by plotting a graph of (micro- or macroaveraged) precision against recall. A better classifier has a higher curve (see Figure 5.14 for an example). One may also plot the line $y = x$ on this graph and note where this intersects the recall-precision plot. This point is called the *break-even point*. Also used is the so-called F_1 *score,* which is defined as the harmonic mean of recall and precision:

$$F_1 = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

where recall and precision may be defined in the various ways mentioned above. The harmonic mean discourages classifiers that sacrifice one measure for another too drastically.

5.4 Nearest Neighbor Learners

The basic intuition behind nearest neighbor (NN) classifiers is that similar documents are expected to be assigned the same class label. The vector-space model introduced in Chapter 3 and the cosine measure for similarity lets us formalize the intuition.

At training time, we simply index each document (as described in Chapter 3) and remember its class label. Given a test document d_q, we use it as a query and let the IR system fetch us the k training documents most similar to d_q (k is a tuned constant). The class that occurs the largest number of times among these k training documents is reported as the class of the test document d_q (see Figure 5.3).

As a refinement, rather than accumulate raw counts of classes, we can accumulate *weighted* counts. If training document d has label c_d, c_d accumulates a score of $s(d_q, d)$, the vector-space similarity between d_q and d, on account of d. The class with the maximum score wins. Yet another refinement is to use a per-class

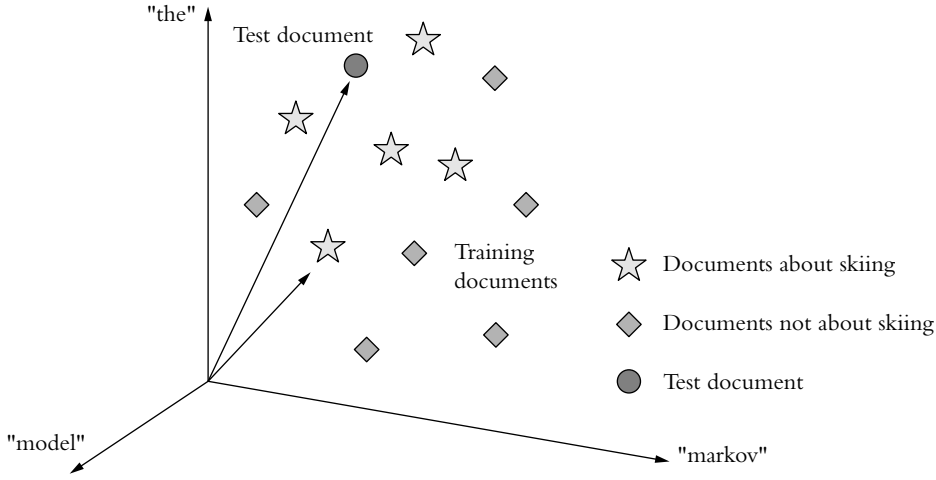

FIGURE 5.3 Nearest neighbor classification.

offset b_c, which is tuned by testing the classifier on a portion of training data held out for this purpose. Combining these ideas, the score of class c for test document d_q is given as

$$\text{score}(c, d_q) = b_c + \sum_{d \in k\text{NN}(d_q)} s(d_q, d) \tag{5.2}$$

where $k\text{NN}(d_q)$ is the set of k training documents most similar to d_q.

Because NN classifiers do very little at training time, they are also called *lazy learners*. Like b_c, the parameter k can be tuned by setting aside a portion of the labeled documents for validation (see Figure 5.1) and trying out various values of k, a process called *cross-validation*. Another approach would be to cluster the training set using some technique from Chapter 4 and choosing a value of k that is related to the size of small clusters.

5.4.1 Pros and Cons

An NN learner has several advantages and disadvantages. The biggest advantage is that very often it comes for free; there may already be an inverted index on the collection to support full-text searching that can be reused for classification. This also makes collection updates trivial to handle (because the classifier is "lazy" and does nothing but indexing at training time). With properly tuned values of k and

b_c for each label c, k-NN classifiers are comparable in accuracy to the best-known classifiers.

On the other hand, classifying a test document d_q involves as many inverted index lookups as there are distinct terms in d_q, followed by scoring the (possibly large number of) candidate documents that overlap with d_q in at least one word, sorting by overall similarity, and picking the best k documents, where k could be very small compared to the number of candidates. Such queries are called *iceberg queries* (because the user is looking for the tip of the iceberg) and are difficult to answer in time that is comparable to output size. In contrast, in the case of naive Bayesian classifiers (introduced in Section 5.6.1), each inverted list has length related to the number of *classes*, which is much smaller than the number of training documents.

A related problem with NN classifiers is the space overhead and redundancy in storing the training information, which is recorded at the level of individual documents. The classifier, being "lazy," does not distill this data into simpler "class models," unlike many of the more sophisticated classifiers we will study later on.

In practice, to reduce space requirements, as well as speed up classification, lazy learners are made to work a little harder at training time. For example, we may find clusters in the data (see Chapter 4) and store only a few statistical parameters per cluster. A test document is first compared with a few cluster representatives and then with the documents in only the most promising clusters. Unfortunately this strategy often leads to various ad hoc choices, for example, number and size of clusters and parameters. In addition, choosing k is a practiced art and the best choice can be sensitive to the specific corpus at hand.

5.4.2 Is TFIDF Appropriate?

Recall that in the computation of similarity s in Equation (5.2), each dimension or term was assigned an inverse document frequency with respect to the *whole* corpus. This may fail to exploit correlations between class labels and the term frequencies. An example with two classes, each having 100 training documents, will make this clear. Consider two terms. One term t_1 occurs in 10 documents in each class, that is, in 10% of the overall corpus. The other term t_2 occurs in 40 documents in the first class but none in the second class, that is, in 20% of the corpus. Thus, t_1 is "rarer," and IDF scoring will downplay the role of t_2 in the distance measure.

Clearly, class labels on training documents should play a central role in making judgments about how well a term can help discriminate between documents from

different classes. Terms that occur *relatively* frequently in some classes compared to others should have higher importance; overall rarity in the corpus is not as important. In the next section, I shall introduce several techniques for estimating the importance of features.

5.5 Feature Selection

We can make a reasonable estimate of a distribution over features when the number of training instances is substantially larger than the number of features. Unfortunately, this is not the case with text. Let us revisit the binary document model where word counts are ignored (see Section 4.4.1). With a vocabulary set W, there are $2^{|W|}$ possible documents. For the Reuters data set, that number would be $2^{30,000} \approx 10^{10,000}$, whereas there are only about 10,300 documents available. In any set of training documents, we will witness a very tiny fraction of these possible documents. Therefore, any estimate of the joint probability distribution over all terms will be very crude.

If we abandon trying to estimate the *joint* distribution of all terms and restrict ourselves, as in Section 4.4.1, to estimating the *marginal* distribution of each term (in each class), the situation improves drastically. However, it may still be nontrivial to judge, from a limited training collection, whether a given term appears more frequently in one class compared to another.

We clarify the issue using a simple two-class example. Let there be N training documents sampled from each class, and fix a term t. Drawing a document and checking if it contains t is like tossing a coin. For the two classes, we can imagine two coins, with $\Pr(\text{head}) = \phi_1$ and ϕ_2, each of which has been tossed N times and produced k_1 and k_2 heads, respectively. It is possible that $\phi_1 < \phi_2$ but $k_1 > k_2$, especially if N is small. If N is too small for us to believe that $\phi_1 < \phi_2$ or $\phi_1 > \phi_2$ with sufficient confidence, it may be better not to use t for classification rather than build an unreliable model, which may lead us to wrong decisions on an ulimited number of test documents. Building an unreliable model that fits limited training data closely, but fails to generalize to unforeseen test data is called *overfitting*.

Feature selection can be heuristic, guided by linguistic and domain knowledge, or statistical. Many classifiers eliminate standard stopwords like *a*, *an*, *the*, and so on. We have seen this to improve classification accuracy a little, even though some stopwords appear to be correlated with the class label. Some classifiers also perform quick-and-dirty approximations to feature selection by ignoring terms that are "too frequent" or "too rare" according to empirically chosen thresholds, which may be corpus- and task-sensitive.

As data sets become larger and more complex, these simple heuristics may not suffice. The challenge looms large especially for hierarchical topic directories, because as one surfs down into detailed topics, terms that would be excellent discriminators with respect to English start resembling stopwords with respect to the specialized collection. Furthermore, in settings such as the Web, jargon and multilingual content makes stopwording difficult.

Feature selection is desirable not only to avoid overfitting and thus *improve* accuracy but also to *maintain* accuracy while discarding as many features as possible, because a great deal of space for storing statistics is saved in this manner. The reduction in space usually results in better classification performance. Sometimes, the reduced class models fit in main memory; even if they don't, caching becomes more effective.

The "perfect" algorithm for feature selection would be goal-directed: it would pick all possible subsets of features, and for each subset train and test a classifier, and retain that subset that resulted in the highest accuracy. For common text collections this is a computational impossibility. Therefore, the search for feature subsets has to be limited to a more manageable extent.

In this section we will study two basic strategies for feature selection. One starts with the empty set and includes good features; the other starts from the complete feature set and excludes irrelevant features.

5.5.1 Greedy Inclusion Algorithms

The most commonly used class of algorithms for feature selection in the text domain share the following outline:

1. Compute, for each term, a measure of discrimination among classes.
2. Arrange the terms in decreasing order of this measure.
3. Retain a number of the best terms or features for use by the classifier.

Often, the measure of discrimination of a term is computed independently of other terms—this is what makes the procedure "greedy." It may result in some terms appearing to be useful that actually add little value given certain other terms that have already been included. In practice, this overinclusion often has mild effects on accuracy.

Several measures of discrimination have been used. The choice depends on the model of documents used by the classifier, the desired speed of training (feature selection is usually considered a part of training), and the ease of updates to documents and class assignments. Although different measures will result in

somewhat different term ranks, the *sets* included for acceptable accuracy will tend to have large overlap. Therefore, most classifiers will tend to be insensitive to the specific choice of discrimination measures. I describe a few commonly used discrimination measures next.

The χ^2 test

Classic statistics provides some standard tools for testing if the class label and a single term are "significantly" correlated with each other. For simplicity, let us consider a two-class classification problem and use the *binary* document model (see Section 4.4.1). Fix a term t, let the class labels be 0 and 1, and let

$k_{i,0}$ = number of documents in class i not containing term t

$k_{i,1}$ = number of documents in class i containing term t

This gives us a 2×2 contingency matrix

		I_t	
		0	1
C	0	k_{00}	k_{01}
	1	k_{10}	k_{11}

where C and I_t denote Boolean random variables and $k_{\ell m}$ denotes the number of observations where $C = \ell$ and $I_t = m$. We would like to test if these random variables are independent or not. Let $n = k_{00} + k_{01} + k_{10} + k_{11}$. We can estimate the marginal distributions as

$$\Pr C = 0 = (k_{00} + k_{01})/n,$$
$$\Pr(C = 1) = 1 - \Pr(C = 0) = (k_{10} + k_{11})/n,$$
$$\Pr(I_t = 0) = (k_{00} + k_{10})/n, \text{ and}$$
$$\Pr(I_t = 1) = 1 - \Pr(I_t = 0) = (k_{01} + k_{11})/n$$

If C and I_t were independent we would expect $\Pr(C = 0, I_t = 0) = \Pr(C = 0) \Pr(I_t = 0)$. Our empirical estimate of $\Pr(C = 0, I_t = 0)$ is k_{00}/n. The same holds for the three other cells in the table. We expect cell (ℓ, m) to have value $n \Pr(C = \ell, I_t = m)$, and its observed value is $k_{\ell m}$. The χ^2 measure aggregates the deviations of observed values from expected values (under the independence hypothesis), as follows:

$$\chi^2 = \sum_{\ell,m} \frac{\left(k_{\ell m} - n \Pr(C = \ell) \Pr(I_t = m)\right)^2}{n \Pr(C = \ell) \Pr(I_t = m)}$$

This can be simplified to

$$\chi^2 = \frac{n(k_{11}k_{00} - k_{10}k_{01})^2}{(k_{11} + k_{10})(k_{01} + k_{00})(k_{11} + k_{01})(k_{10} + k_{00})} \tag{5.3}$$

The larger the value of χ^2, the lower is our belief that the independence assumption is upheld by the observed data. Statisticians use precompiled tables to determine the confidence with which the independence assumption is refuted. This test is similar to the likelihood ratio test, described in Section 3.2.5, for detecting phrases.

For feature selection, it is adequate to sort terms in decreasing order of their χ^2 values, train several classifiers with a varying number of features (picking the best ones from the ranked list), and stopping at the point of maximum accuracy. See Figures 5.4 and 5.5 for details.

Mutual information

This measure from information theory is useful when the multinomial document model (see Section 4.4.1) is used, term occurrences are regarded as discrete events, documents are of diverse length (as is usual), and no length scaling is performed.

If X and Y are discrete random variables taking specific values denoted x, y, then the *mutual information* (MI) between them is defined as

$$\mathrm{MI}(X, Y) = \sum_x \sum_y \Pr(x, y) \log \frac{\Pr(x, y)}{\Pr(x)\Pr(y)} \tag{5.4}$$

where the marginal distributions are denoted $\Pr(x)$ and $\Pr(y)$, shorthand for $\Pr(X = x)$ and $\Pr(Y = y)$ as usual.

MI measures the extent of dependence between random variables, that is, the extent to which $\Pr(x, y)$ deviates from $\Pr(x)\Pr(y)$ (which represents the independence assumption), suitably weighted with the distribution mass at (x, y). (Therefore, deviations from independence at rare values of (x, y) are played down in the measure.) If X and Y are independent, then $\Pr(x, y)/\Pr(x)\Pr(y) = 1$ for all x, y and therefore $\mathrm{MI}(X, Y) = 0$. It can also be shown that $\mathrm{MI}(X, Y)$ is zero only if X and Y are independent. The more positive it is, the more correlated X and Y are.

There are several instructive ways to interpret MI. One interpretation is that it is the *reduction* in the entropy of X if we are told the value of Y, or equivalently, the reduction in the entropy of Y given X. The entropy of a random variable X taking on values from a discrete set of symbols $\{x\}$ is given

by $H(X) = -\sum_x \Pr(x) \log \Pr(x)$. The conditional entropy $H(X|Y)$ is given by $-\sum_{x,y} \Pr(x, y) \log \Pr(x|y) = -\sum_{x,y} \log \frac{\Pr(x,y)}{\Pr(y)}$. It is easy to verify that

$$\mathrm{MI}(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{5.5}$$

If the difference in entropy is large, the value of X tells us a lot about the conditional distribution of Y and vice versa.

Another interpretation of MI uses the Kullback-Leibler (KL) distance [57] between distributions. The KL distance from distribution Θ_1 to Θ_2, each defined over a random variable Z taking values from the domain $\{z\}$, is defined as

$$\mathrm{KL}(\Theta_1 \| \Theta_2) = \sum_z \Pr_{\Theta_1}(z) \log \frac{\Pr_{\Theta_1}(z)}{\Pr_{\Theta_2}(z)}$$

The KL distance gives the average number of bits wasted by encoding events from the "correct" distribution Θ_1 using a code based on a not-quite-right distribution Θ_2. In our case, $Z = (X, Y)$, $z = (x, y)$, and the model $\Theta_2 = \Theta_{\text{independent}}$ corresponds to the hypothesis that X and Y are independent, that means that $\Pr(x, y) = \Pr(x) \Pr(y)$, whereas $\Theta_1 = \Theta_{\text{true}}$ makes no such assumption. Thus,

$$\mathrm{KL}(\Theta_{\text{true}} \| \Theta_{\text{independent}}) = \sum_{x,y} \Pr(x, y) \log \frac{\Pr(x, y)}{\Pr(x) \Pr(y)} = \mathrm{MI}(X, Y) \tag{5.6}$$

If $\mathrm{MI}(X, Y)$ turns out to be zero, $\Theta_{\text{independent}}$ was a perfectly accurate approximation to Θ_{true}. To the extent $\mathrm{MI}(X, Y)$ is large, X and Y are dependent.

To apply MI to feature selection, we will map the above definition to document models in a natural way. Fix a term t and let I_t be an event associated with that term. The definition of the event will vary depending on the document model. For the binary model, $i_t \in \{0, 1\}$, whereas for the multinomial model, i_t is a nonnegative integer. $\Pr(i_t)$ is the empirical fraction of documents in the training set in which event i_t occurred. For example, in the multinomial model, $\Pr(I_t = 2)$ is the empirical fraction of documents in which term t occurred twice. Let c range over the set of classes. $\Pr(i_t, c)$ is the empirical fraction of training documents that are in class c with $I_t = i_t$. $\Pr(c)$ is the fraction of training documents belonging to class c.

For the binary document model and two classes (as in the case of the χ^2 test), the MI of term t with regard to the two classes can be written as

$$\mathrm{MI}(I_t, C) = \sum_{\ell,m \in \{0,1\}} \frac{k_{\ell,m}}{n} \log \frac{k_{\ell,m}/n}{(k_{\ell,0} + k_{\ell,1})(k_{0,m} + k_{1,m})/n^2} \tag{5.7}$$

A possible problem with this approach is that document lengths are not normalized. If a term occurs roughly at the same rate (say, five times per 10,000 words) in two classes, but one class has longer documents than the other, the term may appear to be a good feature using this measure. For this reason, length-normalized feature selection algorithms are sometimes preferred.

Fisher's discrimination index

This measure is useful when documents are scaled to constant length, and therefore, term occurrences are regarded as fractional real numbers. For simplicity let us again consider a two-class learning problem. Let X and Y be the sets of document vectors corresponding to the two classes. The components of these document vectors may be raw term counts scaled to make each document vector unit length, or we may already have applied some term-weighting scheme.

Let $\mu_X = (\sum_X x)/|X|$ and $\mu_Y = (\sum_Y y)/|Y|$ be the mean vectors, or centroids, for each class. Each document vector and these mean vectors are column vectors in $\mathbb{R}^m$, say. Further, let the respective covariance matrices be $S_X = (1/|X|) \sum_X (x - \mu_X)(x - \mu_X)^T$ and $S_Y = (1/|Y|) \sum_Y (y - \mu_Y)(y - \mu_Y)^T$. The covariance matrices are $m \times m$ in size.

Fisher's discriminant seeks to find a column vector $\alpha \in \mathbb{R}^m$ such that the ratio of the square of the difference in mean vectors projected onto it, that is, $(\alpha^T(\mu_X - \mu_Y))^2$, to the average projected variance $\frac{1}{2}\alpha^T(S_X + S_Y)\alpha$, is maximized. Noting that both the numerator and denominator are scalar numbers, and that $\alpha^T S_X \alpha$ is a simple way of writing $(1/|X|) \sum_X \alpha^T (x - \mu_X)(x - \mu_X)^T \alpha$, we can write

$$\alpha^* = \arg\max_{\alpha} J(\alpha) = \arg\max_{\alpha} \frac{(\alpha^T(\mu_X - \mu_Y))^2}{\alpha^T(S_X + S_Y)\alpha} \tag{5.8}$$

Informally, Fisher's discriminant finds a projection of the data sets X and Y onto a line such that the two projected centroids are far apart compared to the spread of the point sets projected onto the same line.

With $S = (S_X + S_Y)/2$, it can be shown that $\alpha = S^{-1}(\mu_X - \mu_Y)$ achieves the extremum when S^{-1} exists. (We will omit the "2" where it won't affect the optimization.) Also, if X and Y for both the training and test data are generated from multivariate Gaussian distributions with $S_X = S_Y$, this value of α induces the optimal (minimum error) classifier by suitable thresholding on $\alpha^T q$ for a test point q.

Fisher's discriminant in the above form has been used in signal-processing applications, in which the number of dimensions in the x and y vectors is on the order of hundreds at most. Inverting S would be unacceptably slow for tens of thousands of dimensions. To make matters worse, although the raw data set is sparse (most words occur in few documents), the linear transformations would destroy sparsity. In any case, our goal in feature selection is not to arrive at linear projections involving multiple terms but to eliminate terms from consideration.

Therefore, instead of looking for the best single direction α, we will regard each term t as providing a candidate direction α_t, which is parallel to the corresponding axis in the vector-space model. That is, $\alpha_t = (0, \ldots, 1, \ldots, 0)^T$, with a 1 in the tth position alone. We will then compute the *Fisher's index* (FI) of t, defined as

$$\mathrm{FI}(t) = J(\alpha_t) = \frac{(\alpha_t^T(\mu_X - \mu_Y))^2}{\alpha_t^T S \alpha_t} \tag{5.9}$$

Because of the special form of α_t, the expression above can be greatly simplified. $\alpha_t^T \mu_X = \mu_{X,t}$, the tth component of μ_X, and $\alpha_t^T \mu_Y = \mu_{Y,t}$, the tth component of μ_Y. $\alpha^T S_X \alpha$ can also be simplified to $(1/|X|) \sum_X (x_t - \mu_{X,t})^2$, and $\alpha^T S_Y \alpha$ can be simplified to $(1/|Y|) \sum_Y (y_t - \mu_{Y,t})^2$. Thus we can write

$$\mathrm{FI}(t) = \frac{(\mu_{X,t} - \mu_{Y,t})^2}{(1/|X|) \sum_X (x_t - \mu_{X,t})^2 + (1/|Y|) \sum_Y (y_t - \mu_{Y,t})^2} \tag{5.10}$$

This measure can be generalized to a set $\{c\}$ of more than two classes to yield the form

$$\mathrm{FI}(t) = \frac{\sum_{c_1,c_2} (\mu_{c_1,t} - \mu_{c_2,t})^2}{\sum_c \frac{1}{|D_c|} \sum_{d \in D_c} (x_{d,t} - \mu_{c,t})^2} \tag{5.11}$$

where D_c is the set of training documents labeled with class c. Terms are sorted in decreasing order of $\mathrm{FI}(t)$ and the best ones chosen as features.

Validation

Merely ranking the terms does not complete the process; we have to decide a cutoff rank such that only terms that pass the bar are included in the feature set. We can do this by validation or cross-validation. In the validation approach, a portion of the training documents are held out, the rest being used to do term ranking. Then the held-out set is used as a test set. Various cutoff ranks can be tested using

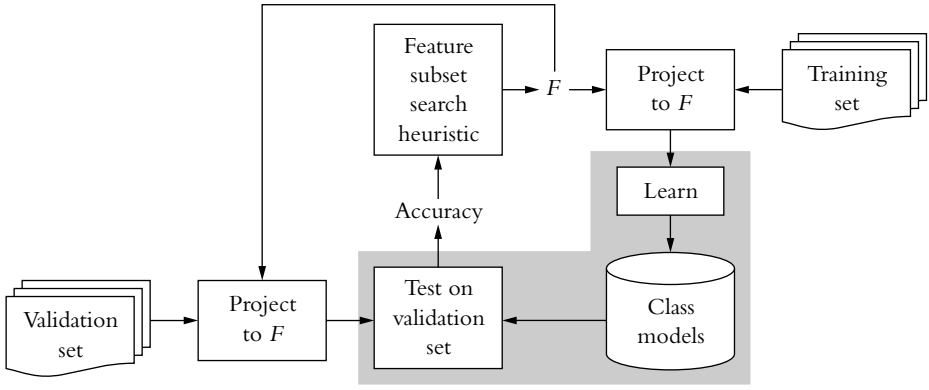

FIGURE 5.4 A general illustration of wrapping for feature selection.

the same held-out set. In *leave-one-out* cross-validation, for each document d in the training set D, a classifier is trained over $D \setminus \{d\}$ and then tested on d. If this takes too much time, a simpler form of cross-validation can be used. The training set is partitioned into a few parts. In turn, one part is taken to be the test set, and the remaining parts together form the training set. An aggregate accuracy is computed over all these trials.

The training and test sets, derived using any of the approaches described above, may be used with a *wrapper*, shown in Figure 5.4, to search for the set of features that yield the highest accuracy. A simple "search heuristic" shown in the diagram is to keep adding one feature at every step until the classifier's accuracy ceases to improve. For certain kinds of classifiers (e.g., maximum entropy classifiers, see Section 5.8, or support vector machines, see Section 5.9.2), such a search would be very inefficient: it would essentially involve training a classifier from scratch for each choice of the cutoff rank. Luckily, some other classifiers (like the naive Bayesian classifier, see Section 5.6) can be evaluated on many choices of feature sets at once.

Figure 5.5 shows the effect of feature selection on the accuracy of Bayesian classifiers, which I will discuss in detail in Section 5.6. The corpus is a selection of 9600 patents sampled from the U.S. Patent database. The terms were ordered using Fisher's discriminant. The classifiers use the binary and multinomial document models, discussed in Section 4.4.1. Only 140 out of about 20,000 raw features

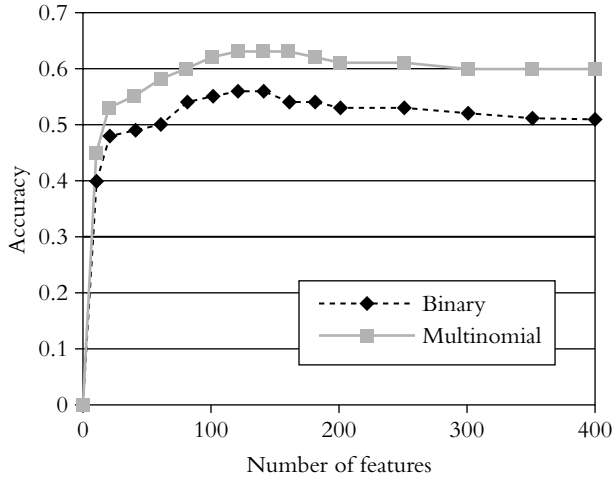

FIGURE 5.5 Effect of feature selection on Bayesian classifiers.

suffice for the best feasible accuracy, which cuts down statistics storage and access costs dramatically. For reasons given later, Bayesian classifiers cannot overfit much, although there is a visible degradation in accuracy beyond the best choice of the number of features. The accuracy varies quite smoothly in the vicinity of the maximum. Barring minor fluctuations, the accuracy increases sharply as the very best features are included one by one, then fluctuates slightly near the crest (which is quite wide) before showing a small drop-off.

5.5.2 Truncation Algorithms

Another approach to feature selection is to start from the complete set of terms T and drop terms from consideration, ending up with a feature subset $F \subseteq T$. What is the desirable property relating F to T?

Most probabilistic classifiers must, one way or another, derive a conditional probability distribution of class labels given data, which we denote as $\Pr(C|T)$, where C is the class label and T is the multivariate term vector. As a result of restricting the training data to F, the distribution changes to $\Pr(C|F)$. We would like to keep the distorted distribution $\Pr(C|F)$ as similar as possible to the original $\Pr(C|T)$ while minimizing the size of F. The similarity or distance between two distributions can be measured in various ways; a well-known measure is the KL distance discussed above.

1: **while** truncated $\Pr(C|F)$ is reasonably close to original $\Pr(C|T)$ **do**
2: **for** each remaining feature X **do**
3: *Identify a candidate Markov blanket M:*
4: For some tuned constant k, find the set M of k variables in $F \setminus X$ that are most strongly correlated with X
5: *Estimate how good a blanket M is:*
6: Estimate

$$\sum_{x_M, x} \Pr(X_M = x_M, X = x)\mathrm{KL}\big(\Pr(C|X_M = x_M, X = x), \Pr(C|X_M = x_M)\big)$$

7: **end for**
8: Eliminate the feature having the best surviving Markov blanket
9: **end while**

FIGURE 5.6 Pseudocode for a heuristic algorithm for feature truncation.

Two random variables P and Q are said to be *conditionally independent* given R, if for any value assignments p, q, r, $\Pr(P = p|Q = q|R = r) = \Pr(P = p|R = r)$. Thus, Q gives no information about P over and above that which we gain by knowing the value of R.

Let X be a feature in T. Let $M \subseteq T \setminus \{X\}$. M is called a *Markov blanket* for $X \in T$ if X is conditionally independent of $(T \cup C) \setminus (M \cup \{X\})$, given M. Intuitively, the presence of M renders the presence of X unnecessary as a feature. It can be shown that eliminating a variable because it has a Markov blanket contained in other existing features does not increase the KL distance between $\Pr(C|T)$ and $\Pr(C|F)$. In practice, there may not be a perfect Markov blanket for any variable, but only an approximate one, and finding it may be difficult. To control the computational cost, we may limit our search for Markov blankets M to those with at most k features. As another cost-cutting heuristic, given feature X, we may restrict our search for the members of M to those features that are most strongly correlated (using tests similar to the χ^2 or MI tests) with X. A sample pseudocode is shown in Figure 5.6. In experiments with the Reuters data set, over two-thirds of T could be discarded while *increasing* classification accuracy by a few percentage points.

5.5.3 Comparison and Discussion

I have presented a variety of measures of association between terms and class labels, and two generic approaches to selecting features. The preferred choice of

association measure and selection algorithm depends on the nature and difficulty of the classification task.

In my experience with several kinds of classifiers and standard benchmarks, I have found that the choice of association measures does not make a dramatic difference, provided the issue of document length is addressed properly. Although different association measures induce different orderings on the terms, by the time we have included enough terms for acceptable accuracy, the *set* of terms included under all the orderings show significant overlap.

Greedy inclusion algorithms scale nearly linearly with the number of features, whereas the Markov blanket technique is much more elaborate and general, taking time proportional to at least $|T|^k$. Markov blankets seek to improve upon greedy inclusion in two important ways, illustrated by these simplified examples:

- The correlation between C and X_1, and between C and X_2, may be individually strong, while X_1's power to predict C may render X_2 unnecessary as a feature, or vice versa. A greedy inclusion algorithm may include them both.
- The correlation between C and X_1, and between C and X_2, may be individually weak, but collectively, X_1, X_2 may be an excellent predictor of C. This might happen if X_1, X_2 are associated with phrases whose constituent term(s) also appear in other contexts. A greedy inclusion approach might discard both X_1 and X_2.

The first concern is primarily one of efficiency. Greedy inclusion may overestimate the number of features required. If the classifier has high quality, feature redundancy does not affect accuracy; it is purely a performance issue. Even for crude classifiers, the effect on accuracy is generally quite small (see, e.g., Figure 5.5).

The second concern is potentially more serious, but practical experience [138] seems to indicate that there is enough natural redundancy among features in text that we need not be too concerned with missing weak signals. In particular, it is rare to find X_1 and X_2 weakly correlated with C but jointly predicting C much better than other single features.

In my experience, the binary view of a feature being either useful or not is not the best possible, especially for hypertext applications where artificial features need to be synthesized out of markup or hyperlinks. As Joachims [119] and others point out, textual features are many in number, each being of low quality. Most have tiny amounts of information for predicting C, but these tiny amounts vary a great deal from one feature to another. To accommodate that view, a classifier might

transform and combine features into fewer, simpler ones, rather than just discard a large number of features. A common technique is to represent the documents in vector space (see Chapter 3) and then project the document vectors to a lower-dimensional space using a variety of approaches (see Chapter 4). Investigating the effect of such transformations on classification accuracy can be an interesting area of research.

5.6 Bayesian Learners

Once feature selection is performed, nonfeature terms are removed from the training documents, and the resulting "clean" documents are used to train the learner. In this section we will study *Bayesian learners,* a practical and popular kind of statistical learner. In spite of their crude approximations, Bayesian classifiers remain some of the most practical text classifiers used in applications.

We will assume, for simplicity of exposition, that a document can belong to exactly one of a set of classes or topics. Document creation is modeled as the following process:

1. Each topic or class c has an associated *prior* probability $\Pr(c)$, with $\sum_c \Pr(c) = 1$. The author of a document first picks a topic at random with its corresponding probability.
2. There is a class-conditional document distribution $\Pr(d|c)$ for each class. Having earlier fixed a class c, its document distribution is used to generate the document.

Thus the overall probability of generating the document from class c is $\Pr(c)\ \Pr(d|c)$. Finally, given the document d, the *posterior* probability that d was generated from class c is seen, using Bayes's rule, to be

$$\Pr(c|d) = \frac{\Pr(c)\ \Pr(d|c)}{\sum_\gamma \Pr(\gamma)\ \Pr(d|\gamma)} \tag{5.12}$$

γ ranges over all classes so that $\Pr(c|d)$ becomes a proper probability measure.

$\Pr(d|c)$ is estimated by modeling the class-conditional term distribution in terms of a set of parameters that we can collectively call Θ. Our estimate of Θ is based on two sources of information:

- Prior knowledge that exists before seeing any training documents for the current problem. This is characterized by a distribution on Θ itself.
- Terms in the training documents D.

After observing the training data D, our posterior distribution for Θ is written as $\Pr(\Theta|D)$. Based on this discussion we can elaborate

$$\begin{aligned}\Pr(c|d) &= \sum_{\Theta} \Pr(c|d, \Theta) \Pr(\Theta|D) \\ &= \sum_{\Theta} \frac{\Pr(c|\Theta) \Pr(d|c, \Theta)}{\sum_{\gamma} \Pr(\gamma|\Theta) \Pr(d|\gamma, \Theta)} \Pr(\Theta|D) \end{aligned} \tag{5.13}$$

The sum may be taken to an integral in the limit for a continuous parameter space, which is the common case. In effect, because we only know the training data for sure and are not sure of the parameter values, we are summing over all possible parameter values. Such a classification framework is called *Bayes optimal*. In practice, taking the expectation over $\Pr(\Theta|D)$ is computationally infeasible for all but the smallest number of dimensions. A common practice is to replace the integral above with the value of the integrand ($\Pr(c|d, \Theta)$) for one specific value of Θ. For example, we can choose $\arg\max_{\Theta} \Pr(\Theta|D)$, called the *maximum likelihood estimate* (MLE). MLE turns out not to work well for text classification; alternatives are suggested shortly.

5.6.1 Naive Bayes Learners

A statistical learner that is widely used for its simplicity and speed of training, applying, and updating is the *naive Bayes learner*. The epithet "naive" signifies the assumption of independence between terms—that is, that the joint term distribution is the product of the marginals. The models for the marginals depend on the document model being used. Here I will use the binary and multinomial models first introduced in Section 4.4.1.

In the *binary* model, the parameters are $\phi_{c,t}$, which indicates the probability that a document in class c will mention term t at least once. With this definition,

$$\Pr(d|c) = \prod_{t \in d} \phi_{c,t} \prod_{t \in W, t \notin d} (1 - \phi_{c,t}) \tag{5.14}$$

W being the set of features. We do not wish to calculate $\prod_{t \in W, t \notin d}(1 - \phi_{c,t})$ for every test document, so we rewrite Equation (5.14) as

$$\Pr(d|c) = \prod_{t \in d} \frac{\phi_{c,t}}{1 - \phi_{c,t}} \prod_{t \in W} (1 - \phi_{c,t})$$

precompute and store $\prod_{t \in W}(1 - \phi_{c,t})$ for all c, and only compute the first product at testing time.

In the *multinomial* model, each class has an associated die with $|W|$ faces. The $\phi_{c,t}$ parameters are replaced with $\theta_{c,t}$, the probability of the face $t \in W$ turning up on tossing the die. Let term t occur $n(d, t)$ times in document d, which is said to have *length* $\ell_d = \sum_t n(d, t)$. The document length is a random variable denoted L and assumed to follow a suitable distribution for each class. For this model,

$$\begin{aligned}\Pr(d|c) &= \Pr(L = \ell_d|c)\ \Pr(d|\ell_d, c) \\ &= \Pr(L = \ell_d|c)\binom{\ell_d}{\{n(d,t)\}} \prod_{t \in d} \theta_t^{n(d,t)} \end{aligned} \tag{5.15}$$

where $\binom{\ell_d}{\{n(d,t)\}} = \frac{\ell_d!}{n(d,t_1)!\, n(d,t_2)!\cdots}$ is the multinomial coefficient, which can be dropped if we are just *ranking* classes, because it is the same for all c. It is also common (but questionable) to assume that the length distribution is the same for all classes and thus drop the $\Pr(L = \ell_d|c)$ term as well.

Both forms of naive Bayes classifiers multiply together a large number of small probabilities, resulting in extremely tiny probabilities as answers. Care is needed to store all numbers as logarithms and guard against unwanted underflow. Another effect of multiplying many tiny ϕ or θ values is that the class that comes out at the top wins by a huge margin, with a score very close to 1, whereas all other classes have negligible probability. The extreme score skew can be unintuitive in case two or more classes are reasonable candidates.

For two-class problems, a *logit function* is sometimes used to sanitize the scores. Let the classes be $+1$ and -1. The logit function is defined as

$$\text{logit}(d) = \frac{1}{1 + e^{-\text{LR}(d)}} \tag{5.16}$$

where

$$\text{LR}(d) = \frac{\Pr(C = +1|d)}{\Pr(C = -1|d)}$$

is the likelihood ratio. Note that as $\text{LR}(d)$ stretches from 0 to ∞, $\text{logit}(d)$ ranges from $\frac{1}{2}$ to 1. The $\text{logit}(x)$ function has a steep slope near $x = 0$ and levels off rapidly for large x. Finding a suitable threshold on the logit function may reduce the problem of score skew [175].

Parameter smoothing

MLE cannot be used directly in the naive Bayes classifier. For example, in the binary model, if a test document d_q contains a term t that never occurred in any

training document in class c, $\phi_{c,t}^{\text{MLE}} = 0$. As a result $\Pr(c|d_q)$ will be zero, even if a number of other terms clearly hint at a high likelihood of class c generating the document. Unfortunately, such "accidents" are not rare at all.

There is a rich literature, dating back to Bayes in the 18th century and Laplace in the 19th century, on the issue of estimating probability from insufficient data. We can start delving into this issue by posing the following question: If you toss a coin n times and it always comes up heads, what is the probability that the $(n+1)$st toss will also come up heads? Although MLE leads to the answer 1, it is not appealing from real-life experience. Furthermore, we certainly expect the answer to change with n: if $n = 1$, we are still quite agnostic about the fairness of the coin; if $n = 1000$, we have a firmer belief. MLE cannot distinguish between these two cases.

In our setting, each coin toss is analogous to inspecting a document (in some fixed class c) to see if term t appears in it. The MLE estimate of $\phi_{c,t}$ is simply the fraction of documents in class c containing the term t. When c and/or t are omitted, they are assumed to be fixed for the rest of this section. Also for this section let k out of n documents contain the term; we denote this event by the notation $\langle k, n\rangle$.

The Bayesian approach to parameter smoothing is to posit a *prior* distribution on ϕ, called $\pi(\phi)$, before any training data is inspected. An example of π is the uniform distribution $\mathcal{U}(0, 1)$. The *posterior* distribution of ϕ is denoted by

$$\pi(\phi|\langle k, n\rangle) = \frac{\pi(\phi)\ \Pr(\langle k, n\rangle|\phi)}{\int_0^1\ dp\, \pi(p)\ \Pr(\langle k, n\rangle|p)} \tag{5.17}$$

Usually, the smoothed estimate $\tilde{\phi}$ is some property of the posterior distribution $\pi(\phi|\langle k, n\rangle)$. There is a *loss function* $L(\phi, \tilde{\phi})$, which characterizes the penalty for picking a smoothed value $\tilde{\phi}$ as against the "true" value. Often, the loss is taken as the square error, $L(\phi, \tilde{\phi}) = (\phi - \tilde{\phi})^2$. For this choice of loss, the best choice of the smoothed parameter is simply the expectation of the posterior distribution on ϕ having observed the data

$$\begin{aligned}\tilde{\phi} = E(\pi(\phi|\langle k, n\rangle)) &= \frac{\int_0^1 p\ dp\, \pi(p)\ \Pr(\langle k, n\rangle|p)}{\int_0^1\ dp\, \pi(p)\ \Pr(\langle k, n\rangle|p)} = \frac{\int_0^1 p^{k+1}(1-p)^{n-k}dp}{\int_0^1 p^k(1-p)^{n-k}dp} \\ &= \frac{\mathbb{B}(k+2, n-k+1)}{\mathbb{B}(k+1, n-k+1)} = \frac{\Gamma(k+2)}{\Gamma(k+1)}\frac{\Gamma(n+2)}{\Gamma(n+3)} = \frac{k+1}{n+2}\end{aligned} \tag{5.18}$$

where $\mathbb{B}$ and Γ are the standard beta and gamma functions. Although the derivation is nontrivial, the end result is simple in a misleading way: just "combine" a prior belief of fairness ($\frac{1}{2}$) with observed data ($\frac{k}{n}$). This is called *Laplace's law of succession*. Heuristic alternatives exist; one example is *Lidstone's law of succession,* which sets $\phi = (k + \lambda)/(n + 2\lambda)$, where λ is a tuned constant trading off between prior belief and data. (In Laplace's law, they have equal say.)

The derivation for the multinomial document model is quite similar, except that instead of two possible events in the binary model discussed above, there are $|W|$ possible events, where W is the vocabulary. Thus

$$\tilde{\theta}_{c,t} = \frac{1 + \sum_{d \in D_c} n(d,t)}{|W| + \sum_{d \in D_c, \tau \in d} n(d,\tau)} \tag{5.19}$$

Comments on accuracy and performance

The multinomial naive Bayes classifier generally outperforms the binary variant for most text-learning tasks. Figure 5.5 shows an example. A well-tuned k-NN classifier may outperform a multinomial naive Bayes classifier [217], although the naive Bayes classifier is expected to produce far more compact models and take less time to classifiy test instances.

Any Bayesian classifier partitions the multidimensional term space into regions separated by what are called *decision boundaries*. Within each region, the probability (or probability density, if continuous random variables are modeled) of one class is higher than others; on the boundaries, the probabilities of two or more classes are exactly equal. Two or more classes have comparable probabilities near the boundaries, that are therefore the regions of potential confusion. Little confusion is expected in those parts of a region that have a dense collection of examples, all from the associated class.

To make this more concrete, consider a two-class problem with training data $\{(d_i, c_i), i = 1, \ldots, n\}$, where $c_i \in \{-1, 1\}$. As we have seen before, the multinomial naive Bayes model assumes that a document is a bag or multiset of terms, and the term counts are generated from a multinomial distribution after fixing the document length ℓ_d, which, being fixed for a given document, lets us write

$$\Pr(d|c, \ell_d) = \binom{\ell_d}{\{n(d,t)\}} \prod_{t \in d} \theta_{c,t}^{n(d,t)} \tag{5.20}$$

where $n(d, t)$ is the number of times t occurs in d, and $\theta_{c,t}$ are suitably estimated multinomial probability parameters with $\sum_t \theta_{c,t} = 1$ for each c (see Section 4.4.1).

For the two-class scenario, we only need to compare $\Pr(c = -1|d)$ against $\Pr(c = 1|d)$, or equivalently, $\log \Pr(c = -1|d)$ against $\log \Pr(c = 1|d)$, which simplifies to a comparison between

$$\log \Pr(c = 1) + \sum_{t \in d} n(d, t) \log \theta_{1,t} \tag{5.21}$$

and

$$\log \Pr(c = -1) + \sum_{t \in d} n(d, t) \log \theta_{-1,t}$$

where $\Pr(c = \ldots)$, called the class *priors*, are the fractions of training instances in the respective classes. Simplifying (5.21), we see that NB is a linear classifier: it makes a decision between $c = 1$ and $c = -1$ by thresholding the value of $\alpha_{\text{NB}} \cdot d + b$ for a suitable vector α_{NB} (which depends on the parameters $\theta_{c,t}$) and a constant b depending on the priors. Here d is overloaded to denote the vector of $n(d, t)$ term counts and "$\cdot$" denotes a dot-product.

One notable problem with naive Bayes classifiers is their strong *bias*. A machine learning algorithm is biased if it restricts the space of possible hypotheses from which it picks a hypothesis to fit the data, before assessing the data itself. Although a naive Bayes classifier picks linear discriminants, it cannot pick from the *entire* set of possible linear discriminants, because it fixes the policy that $\alpha_{\text{NB}}(t)$, the tth component of the discriminant, depends only on the statistics of term t in the corpus. In Sections 5.8 and 5.9, you shall see other classifiers that do not suffer from this form of bias.

5.6.2 Small-Degree Bayesian Networks

The naive Bayes model asserts that fixing the class label of a document imposes a class-conditional distribution on the terms that occur in the document, but that there are no other statistical dependencies between the terms themselves (which is a gross approximation). This simple dependency structure can be represented as a simple hub-and-spoke graph, shown in Figure 5.7(a). Each random variable, including the class label and each term, is a node, and dependency edges are drawn from c to t for each t. If we wish to represent additional dependencies between terms, more edges have to be introduced as shown in Figure 5.7(b), creating a *Bayesian network*.

A Bayesian network is a directed acyclic graph that represents dependencies between a set of random variables and models their joint distribution. Each node

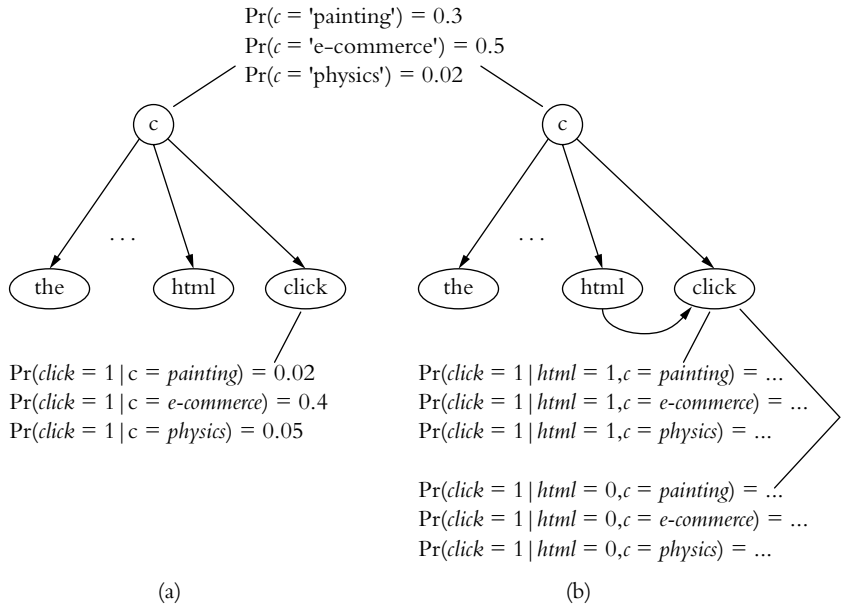

FIGURE 5.7 Bayesian networks. For the naive Bayes assumption, the only edges are from the class variable to individual terms (a). Toward better approximations to the joint distribution over terms, the probability of a term occurring may now depend on observations about other terms as well as the class variable (b).

in the graph represents a random variable. The set of nodes that are connected by directed edges to a node X are called the *parents* of X, denoted $\mathbf{Pa}(X)$. A specific set of values for these parents is denoted $\mathbf{pa}(X)$. Fixing the values of the parent variables completely determines the conditional distribution of X in the sense that information about any other variable would not affect this distribution. For discrete variables, the distribution data for X can be stored in the obvious way as a table with each row showing a set of values of the parents, the value of X, and a conditional probability.

Unlike in the naive models expressed by Equations (5.14) and (5.15), $\Pr(d|c)$ is not a simple product over all terms. Instead it is expressed as a product of conditional probabilities:

$$\Pr(\mathbf{x}) = \prod_{x} \Pr(x|\mathbf{pa}(X)) \tag{5.22}$$

```
1: Compute mutual information MI(X_t, C) between class labels C and each
   feature X_t
2: For each pair of distinct variables X_i and X_j, calculate MI(X_i, X_j|C)
3: Initialize the network with class node C
4: while all X_t has not been added to the network do
5:    Find X_j with maximum MI(X_j, C)
6:    Add X_j to the network
7:    Add directed edge (C, X_j)
8:    while in-degree of X_j is less than k + 1 and there is an X_i not connected
      to X_j do
9:       Find an X_i with highest MI(X_i, X_j|C)
10:      Add directed edge (X_i, X_j)
11:   end while
12: end while
```

FIGURE 5.8 Inducing limited-dependence Bayesian networks.

Using Bayesian networks for text classification addresses the clearly crude approximations made by naive Bayes classifiers regarding term independence.

Given the graph structure of the network and training data, it is in principle simple to derive the probability tables. What is difficult is to derive the structure itself, especially if the number of nodes is large. One way to limit the complexity is to limit the number of parents that each node can have. In our context of text classification, a k-dependence Bayesian network has one node for the class variable C and a node X_t for each term t. There is a directed edge from C to each X_t. In addition, each X_t is permitted to have up to k incident edges from other $X_{t'}$s.

Figure 5.8 shows the pseudocode for constructing such a limited-degree Bayesian network. Generally speaking, the difficult part is to get a good network *structure*. For a specified network, estimating the parameters is relatively straightforward. To enumerate all pairs of features, the algorithm takes at least quadratic time, which makes it difficult to apply this algorithm to large text corpora unless some preelimination of features is performed.

We only know of Bayesian networks designed for the binary document model; the size of the conditional probability tables (see Figure 5.7) can be prohibitive for the multinomial model. While accuracy improvements for structured machine learning data sets (from the U.C. Irvine repository) have been clearly visible, they are surprisingly mild for text data (Reuters). There is room to suspect that the test

problems were too simple, and Bayesian network induction will shine in the face of more complex data sets, where it is harder to discriminate between the classes.

5.7 Exploiting Hierarchy among Topics

In standard classification problems that arise in the structured data scenario, such as data warehouses, the class labels form a discrete set. For example, credit card transactions may be classified as "normal" or "fraudulent." Sometimes there is a mild ordering between the class labels, such as high, medium, or low cancer-risk patients. In contrast, for text classification, the class labels themselves are related by a large and complex class hierarchy, sometimes called a *taxonomy* (although the term "taxonomy" is sometimes reserved for single-word or concept interrelationships). In this section, we will restrict ourselves to hierarchies that are trees. Tree-structured hierarchies are widely used in directory browsing, provide an intuitive interface for representing refinements and generalizations, and are often the output of clustering algorithms. The usual semantics of tree-structured hierarchies is *inheritance*: if class c_0 is the parent of class c_1, any training document that belongs to c_1 also belongs to c_0.

5.7.1 Feature Selection

An important issue that needs to be revisited is feature selection. The discriminating ability of a term is obviously influenced by the set of training documents involved, and therefore the ability should also be quite sensitive to the node (or class) in the hierarchy at which it is evaluated. Note that the measure of discrimination of a term can be evaluated with respect only to *internal* nodes of the hierarchy. To cite a simple example, the (ambiguous) word "can" may be a noisy word at the root node of Yahoo!, but may be a great help in classifying documents under the subtree of `/Science/Environment/Recycling`. (In this particular example, a part-of-speech analysis might have helped, but that is not true in general.)

5.7.2 Enhanced Parameter Estimation

The "uniform prior assumption" made in Section 5.6.1 is unrealistic. For example, in the binary model, the minimum loss parameter value would be $\phi = \frac{1}{2}$ for *all* terms in the absence of data, whereas experience with languages tells us that words are rare and differ greatly in how rare they are. I also introduced one technique toward better smoothing by exploiting document-length distributions.