

4

Text Analysis

Having focused in earlier chapters on the general structure of the Web, in this chapter we will discuss in some detail techniques for analyzing the textual content of individual Web pages. The techniques presented here have been developed within the fields of *information retrieval* (IR) and *machine learning* and include indexing, scoring, and categorization of textual documents.

The focus of IR is that of accessing as efficiently as possible and as accurately as possible a small subset of documents that is maximally related to some user interest. User interest can be expressed for example by a query specified by the user. Retrieval includes two separate subproblems: indexing the collection of documents in order to improve the computational efficiency of access, and ranking documents according to some importance criterion in order to improve accuracy. *Categorization* or classification of documents is another useful technique, somewhat related to information retrieval, that consists of assigning a document to one or more predefined categories. A classifier can be used, for example, to distinguish between relevant and irrelevant documents (where the relevance can be personalized for a particular user or group of users), or to help in the semiautomatic construction of large Web-based knowledge bases or hierarchical directories of topics like the Open Directory (<http://dmoz.org/>).

A vast portion of the Web consists of text documents – thus, methods for automatically analyzing text have great importance in the context of the Web. Of course, retrieval and classification methods for text, such as those reviewed in this chapter can be specialized or modified for other types of Web documents such as images, audio or video (see, for example, Del Bimbo 1999), but our focus in this chapter will be on text.

4.1 Indexing

4.1.1 Basic concepts

In order to retrieve text documents efficiently it is necessary to enrich the collection with specialized data structures that facilitate access to documents in response to

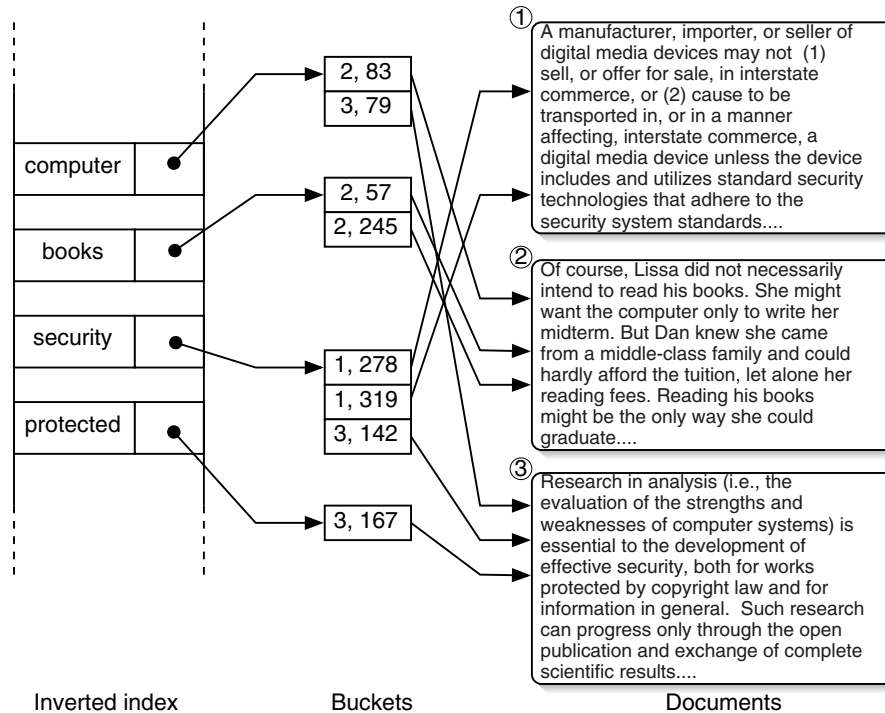


Figure 4.1 Structure of a typical inverted index for information retrieval. Each entry in the occurrence lists (buckets) is a pair of indices that identify the document and the offset within the document where the term associated to the bucket appears. Document (1) is an excerpt from Sen. Fritz Hollings's Consumer Broadband and Digital Television Promotion Act, published at <http://www.politechbot.com/docs/cbdtpa/hollings.s2048.032102.html>. Document (2) is an excerpt from Stallman (1997) and (3) is an excerpt from the declaration submitted by ACM in the District Court of New Jersey on the case of Edward Felten versus the Recording Industry Association of America (http://www.acm.org/usacm/copyright/felten_declaration.html).

user queries. A substring search, even when implemented using sophisticated algorithms like suffix trees or suffix arrays (Manber and Myers 1990), is not adequate for searching very large text collections. Many different methods of text retrieval have been proposed in the literature, including early attempts such as clustering (Salton 1971) and the use of signature files (Faloutsos and Christodoulakis 1984). In practice, *inversion* (Berry and Browne 1999; Witten *et al.* 1999) is the only effective technique for dealing with very large sets of documents. The method relies on the construction of a data structure, called an *inverted index*, which associates lexical items to their occurrences in the collection of documents. Lexical items in text retrieval are called *terms* and may consist of words as well as expressions. The set of terms of interest is called the *vocabulary*, denoted V . In its simplest form, an inverted index is a dictionary where each key is a term $\omega \in V$ and the associated value $b(\omega)$ is a

pointer to an additional intermediate data structure, called a *bucket* or posting list. The bucket associated with a certain term ω is essentially a list of pointers marking all the occurrences of ω in the text collection.

The general structure is illustrated in Figure 4.1. In the simplest case, the entries in each bucket can simply consist of the document identifier (DID), the ordinal number of the document within the collection. However, in many cases it will be more useful to have a separate entry for each occurrence of the term, where each entry consists of the DID and the offset (in characters) of the term's occurrence within the document. The latter approach has two advantages. First, recording all the occurrences of the term allows us to present a user with a short context (i.e. a fragment of surrounding text) in which the term occurs in the retrieved document. Second, the occurrence information enables vicinity queries such as retrieving all documents where two assigned terms are positionally close in the text.

The size of the inverted index is $\Omega(|V|)$ and this data structure can typically be stored in main memory.

It can be implemented using a hash table so that the expected access time is independent of the size of the vocabulary. Buckets and the document repository must typically be stored on disk. This poses a challenge during the construction phase of the inverted index. If the buckets can be stored in the main memory, the construction algorithm is trivial. It simply consists of parsing documents, extracting terms, inserting the term in the inverted index if not present, and finally inserting the occurrence in the bucket. However, if the buckets are on disk, this approach will generate a tremendous amount of disk accesses making the algorithm impractical, since accessing a random byte of information on disk can take up to 10^5 times longer than accessing a random byte in main memory. Thus, for very large collections of text it is necessary to resort to specialized secondary memory algorithms (Witten *et al.* 1999).

Searching for a single term ω in an indexed collection of documents is straightforward. First, we access the inverted index to obtain $b(\omega)$. Then we simply scan the bucket pointed to by $b(\omega)$ to obtain the list of occurrences. The case of Boolean queries with multiple terms is only slightly more complicated. Suppose the query contains literals associated with the presence or absence of k terms. The previous procedure can be repeated separately for each term, yielding k lists of occurrences. These lists are then combined by elementary set operations that correspond to the Boolean operators in the query: intersection for AND, union for OR, and complement for NOT (Harman *et al.* 1992).

4.1.2 Compression techniques

The memory required for each pointer in the buckets can be reduced by sorting the occurrences of each term by its DID. In this way, rather than storing each DID, it suffices to store the sequence of differences between successive DIDs as a list of *gaps*. For example, the sequence of DIDs

(14, 22, 38, 42, 66, 122, 131, 226, 312, 331)

may be stored as a sequence of gaps

(14, 8, 16, 4, 24, 56, 9, 95, 86, 19).

The advantage is that frequent terms will produce many small gaps, and small integers can be encoded by short variable-length codewords. In particular, γ encoding (Elias 1975) represents a gap g as the pair of bit strings (u_g, b_g) , where u_g is the unary code for the number $\lfloor \log_2 g \rfloor + 1$ (i.e. $\lfloor \log_2 g \rfloor$ ones followed by a zero) and b_g is the standard binary code for the number $g - 2^{\lfloor \log_2 g \rfloor}$. For example, the integer ‘7’ would be represented as (110, 11) and ‘9’ would be represented as (1110, 001). In contrast, for a collection of n documents, storing an integer word for a DID would require $\lceil \log_2 n \rceil$ bits. Because terms are distributed according to the Zipf distribution (see Chapter 1), on average this represents a significant memory saving (see Exercise 4.1 for details). Moffat and Zobel (1996) discuss this and other index compression techniques, together with fast algorithms for query evaluation and ranking.

In the case of large collections of Web pages, efficient indexing may be challenging and traditional approaches may quickly become inadequate. The design of an efficient distributed index for Web documents is discussed in Melnik *et al.* (2001).

4.2 Lexical Processing

4.2.1 Tokenization

Tokenization is the extraction of plain words and terms from a document, stripping out administrative metadata and structural or formatting elements, e.g. removing HTML tags from the HTML source file for a Web page. This operation needs to be performed prior to indexing or before converting documents to vector representations that are used for retrieval or categorization (see Section 4.3.1). Tokenization appears to be a straightforward problem but in many practical situations the task can actually be quite challenging.

The case of HTML documents is relatively easy. The simplest approach consists of reducing the document to an *unstructured* representation, i.e. a plain sequence of words with no particular relationship among them other than serial order. This can be achieved by only retaining the text enclosed between `<html>` and `</html>` (see Section 2.1), removing tags, and perhaps converting strings that encode international characters to a standard representation (e.g. Unicode). The resulting unstructured representation allows simple queries related to the presence of terms and to their positional vicinity. More generally, the additional information embedded in HTML can be exploited to map documents to *semi-structured* representations. In this case, the representation is sensitive to the presence of terms in different elements of the document and allows more sophisticated queries like ‘find documents containing *population* in a table header and *famine* in the title’. Extracting semi-structured representations from HTML documents is in principle very easy since tags provide all the necessary

structural information. However, the reader should be aware that, although HTML has a clearly defined formal grammar, real world browsers do not strictly enforce syntax correctness and, as a result, most Web pages fail to rigorously comply to the HTML syntax.¹ Hence, an HTML parser must tolerate errors and include recovery mechanisms to be of any practical usefulness. A public domain parser is distributed with LIBWWW, the W3C Protocol Library <http://www.w3.org/Library/>. An HTML parser written in Perl (with methods for text extraction) is available at <http://search.cpan.org/dist/HTML-Parser/>.

Obviously, after plain text is extracted, punctuation and other special characters need to be stripped off. In addition, the character case may be *folded* (e.g. to all lowercase characters) to reduce the number of index terms.

Besides HTML, textual documents in the Web come in a large variety of formats. Some formats are proprietary and undisclosed and extracting text from such file types is severely limited by the fact that the associated formats have not been disclosed. Other formats are publicly known, such as PostScript² or Portable Document Format (PDF).

Tokenization of PostScript or PDF files can be difficult to handle because these are not data formats but algorithmic descriptions of how the document should be rendered. In particular, PostScript is an interpreted programming language and text rendering is controlled by a set of *show* commands. Arguments to show commands are text strings and two-dimensional coordinates. However, these strings are not necessarily entire words. For example, in order to perform typographical operations such as kerning, ligatures, or hyphenation, words are typically split into fragments and separate show commands are issued for each of the fragments. Show commands do not need to appear in reading order, so it is necessary to track the two-dimensional position of each 'shown' string and use information about the font in order to correctly reconstruct word boundaries. PostScript is almost always generated automatically by other programs, such as typesetting systems and printer drivers, which further complicates matters because different generators follow different conventions and approaches. In fact perfect conversion is not always possible. As an example of efforts in this area, the reader can consult Neville-Manning and Reed (1996) for details on PreScript, a PostScript-to-plain-text converter developed within the New Zealand Digital Library project (Witten *et al.* 1996). Another converter is Pstotext, developed within the Virtual Paper project (<http://research.compaq.com/SRC/virtualpaper/home.html>).

PDF is a binary format that is based on the same core imaging model as PostScript but can contain additional pieces of information, including descriptive and administrative metadata, as well as structural elements, hyperlinks, and even sound or video. In terms of delivered contents, PDF files are therefore much closer in structure to Web pages than PostScript files are. PDF files can (and frequently do, in the case of digital libraries) embed raster images of scanned textual documents. In order to extract text

¹ See <http://validator.w3.org/> for a public HTML validation service.

² PostScript is a registered trademark of Adobe Systems Incorporated.

from raster images, it is necessary to resort to optical character recognition (OCR) systems (Mori *et al.* 1992).

4.2.2 Text conflation and vocabulary reduction

Often it is desirable to reduce all the morphological variants of a given word to a single index term. For example, a document of interest might contain several occurrences of words like *fish*, *fishes*, *fisher*, and *fishers* but would not be retrieved by a query with the keyword *fishing* if the term ‘fishing’ never occurs in the text. Stemming consists of reducing words to their root form (such as *fish* in our example), which becomes the actual index term. Besides its effect on retrieval, stemming can be also useful to reduce the size of the index.

One well known stemming algorithm for English was developed by Porter (1980) as a simplification and systematization of previous work by Lovins (1968). It relies on a preconstructed suffix list with associated rules. An example of a rewriting rule is ‘if the suffix of the word is IZATION and the prefix contains at least one vowel followed by a consonant, then replace the suffix with IZE’. This would transform, for example, the word BINARIZATION into the word BINARIZE. Porter’s stemmer applies rules in five consecutive steps. Source code in several languages and a detailed description of the algorithm are available at <http://www.tartarus.org/~martin/PorterStemmer/>.

Another technique that is commonly used for controlling the vocabulary is the removal of stop words, i.e. common words such as articles, prepositions, and adverbs that are not informative about the semantic content of a document (Fox 1992). Since stop words are very common, removing them from the vocabulary can also significantly help in reducing the size of the index. In practice, stopwords may account for a large percentage of text, up to 20–30%. Naturally, removal of stop words also improves computational efficiency during retrieval.

4.3 Content-Based Ranking

A Boolean query to a Web search engine may return several thousands of matching documents, but a typical user will only be able to examine a small fraction of these. Ranking matching documents according to their relevance to the user is therefore a fundamental problem. In this section we review some classic approaches that do not exploit the hyperlinked structure of the Web. Ranking algorithms that take Web topology into account will be discussed in Chapter 5.

4.3.1 The vector-space model

Text documents can be conveniently represented in a high-dimensional vector space where terms are associated with vector components. More precisely, a text document

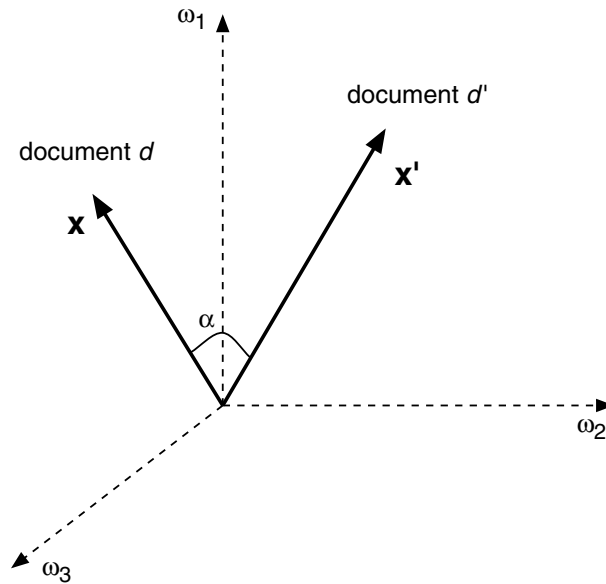


Figure 4.2 Cosine measure of document similarity.

d can be represented as a sequence of terms, $d = (\omega(1), \omega(2), \dots, \omega(|d|))$, where $|d|$ is the length of the document and $\omega(t) \in V$. A vector-space representation of d is then defined as a real vector $\mathbf{x} \in \mathbb{R}^{|V|}$, where each component x_j is a statistic related to the occurrence of the j th vocabulary entry in the document. The simplest vector-based representation is Boolean, i.e. $x_j \in \{0, 1\}$ indicates the presence or the absence of term ω_j in the document being represented.

Vector-based representations are sometimes referred to as a ‘bag of words’, emphasizing that document vectors are invariant with respect to term permutations, since the original word order $\omega(1), \dots, \omega(|d|)$ is clearly lost. Representations of this kind are appealing for their simplicity. Moreover, although they are necessarily lossy from an information theoretic point of view, many text retrieval and categorization tasks can be performed quite well in practice using the vector-space model. Note that typically the total number of terms in a set of documents is much larger than the number of distinct terms in any single document, $|V| \gg |d|$, so that vector-space representations tend to be very *sparse*. This property can be advantageously exploited for both memory storage and algorithm design.

4.3.2 Document similarity

We can define similarity between two documents d and d' as a function $s(d, d') \in \mathbb{R}$. This function, among other things, allows us to rank documents with respect to a query (by measuring the similarity between each document and the query). A classic

approach is based on the vector-space representation and the metric defined by the *cosine coefficient* (Salton and McGill 1983). This measure is simply the cosine of the angle formed by the vector-space representations of the two documents, \mathbf{x} and \mathbf{x}' (see Figure 4.2),

$$\cos(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\| \cdot \|\mathbf{x}'\|} = \frac{\mathbf{x}^T \mathbf{x}'}{\sqrt{\mathbf{x}^T \mathbf{x}'} \cdot \sqrt{\mathbf{x}'^T \mathbf{x}'}} \quad (4.1)$$

where the superscript ‘T’ denotes the ‘transpose’ operator and $\mathbf{x}^T \mathbf{y}$ indicates the dot product or inner product between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, defined as

$$\mathbf{x}^T \mathbf{y} \doteq \sum_{i=1}^m x_i y_i. \quad (4.2)$$

Note that in the case of two sparse vectors \mathbf{x} and \mathbf{y} associated with two documents d and d' , the above sum can be computed efficiently in time $\Omega(|d| + |d'|)$ (see Exercise 4.2).

Several refinements can be obtained by extending the Boolean vector model and introducing real-valued weights associated with terms in a document. A more informative weighting scheme consists of counting the actual number of occurrences of each term in the document. In this case $x_j \in \mathbb{N}$ counts term occurrences in the corresponding document (see Figure 4.3). \mathbf{x} may be multiplied by the constant $1/|d|$ to obtain a vector of term frequencies (TF) within the document.

An important family of weighting schemes combines term frequencies (which are relative to each document) with an ‘absolute’ measure of term importance called inverse document frequency (IDF). IDF decreases as the number of documents in which the term occurs increases in a given collection. So terms that are globally rare receive a higher weight.

Formally, let $D = \{d_1, \dots, d_n\}$ be a collection of documents and for each term ω_j let n_{ij} denote the number of occurrences of ω_j in d_i and n_j the number of documents that contain ω_j at least once. Then we define

$$\text{TF}_{ij} \doteq \frac{n_{ij}}{|d_i|}, \quad (4.3)$$

$$\text{IDF}_j \doteq \log \frac{n}{n_j}. \quad (4.4)$$

Here the logarithmic function is employed as a damping factor.

The TF–IDF weight (Salton *et al.* 1983) of ω_j in d_i can be computed as

$$x_{ij} = \text{TF}_{ij} \cdot \text{IDF}_j \quad (4.5)$$

or, alternatively, as

$$x_{ij} = \frac{\text{TF}_{ij}}{\max_{\omega_k \in d_i} \text{TF}_{ik}} \frac{\text{IDF}_j}{\max_{\omega_k \in d_i} \text{IDF}_k}. \quad (4.6)$$

The IDF weighing is commonly used as an effective heuristic. A theoretical justification has been recently proposed by Papineni (2001), who proved that IDF is the

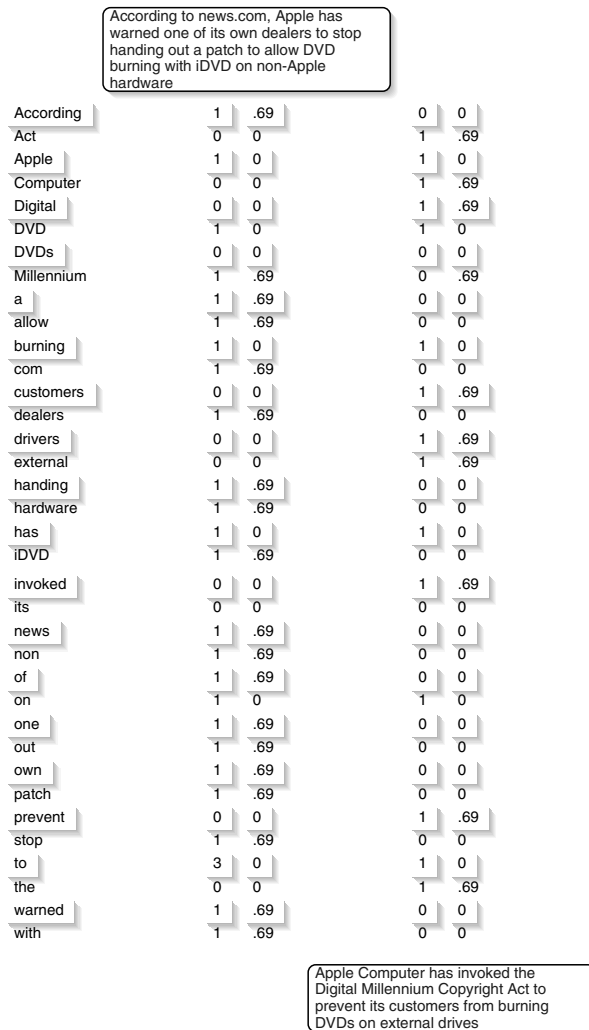


Figure 4.3 Vector-space representations. For each of the two documents, the left vector counts the number of occurrences of each term, while the right vector is based on TF-IDF weights (Equation 4.5).

optimal weight of a term with respect to the minimization of a distance function that generalizes Kullback–Leibler divergence or relative entropy (see Appendix A).

4.3.3 Retrieval and evaluation measures

Let us consider a collection of n documents D . Each document is represented by an m -dimensional vector, where $m = |V|$ and V is the set of terms that occurred in

the collection. Let $\mathbf{q} \in \mathbb{R}^m$ denote the vector associated with a user query (terms that are present in the query but not in V will be stripped off). Each document is then assigned a score, relative to the query, by computing $s(x_i, \mathbf{q})$, $i = 1, \dots, n$. The set R of *retrieved documents* that are presented to the user can be formed by collecting the top-ranking documents according to the similarity measure. The quality of the returned collection can be defined by comparing R to the set of documents R^* that is actually relevant to the query.³

Two common metrics for comparing R and R^* are *precision* and *recall*. Precision π is defined as the fraction of retrieved documents that are actually relevant. Recall that ρ is defined as the fraction of relevant documents that are retrieved by the system. More precisely,

$$\pi \doteq \frac{|R \cap R^*|}{|R|}, \quad \rho \doteq \frac{|R \cap R^*|}{|R^*|}.$$

Note that in this context the ratio between relevant and irrelevant documents is typically very small. For this reason, other common evaluation measures like accuracy or error rate (see Section 4.6.5), where the denominator consists of $|D|$, would be inadequate (it would suffice to retrieve nothing to get very high accuracy). Sometimes precision and recall are combined into a single number called F_β measure defined as

$$F_\beta \doteq \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho}. \quad (4.7)$$

Note that the F_1 measure is the harmonic mean of precision and recall. If β tends to zero (∞) the F_β measure tends to precision (recall).

4.4 Probabilistic Retrieval

Researchers in IR have long posed the question of how to define an optimality criterion for ranking retrieved documents in response to a specified user interest. Robertson (1977) formulated an optimality principle called the probabilistic ranking principle (PRP), stating that

if a reference retrieval system's response to each request is a ranking of the documents in order of decreasing probability of relevance to the user who submitted the request where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data.

³ Of course this is possible only on controlled collections, such as those prepared for the Text REtrieval Conference (TREC) (data, papers describing methodologies and description of evaluation measures and assessment criteria are available at the TREC website <http://trec.nist.gov/>).

The PRP resembles the optimal Bayes decision rule for classification, a concept that is well known for example in the pattern recognition community (Duda and Hart 1973). The Bayes optimal separation is obtained by ensuring that the posterior probability of the correct class (given the observed pattern) should be greater than the posterior probability of any other class. PRP can be mathematically stated by introducing a Boolean variable R (relevance) and by defining a model for the conditional probability $P(R | d, q)$ of relevance of a document d for a given user need (for example, expressed through a query q). Its justification follows from a decision-theoretic argument as follows. Let c denote the cost of retrieving a relevant document and \bar{c} the cost of retrieving an irrelevant document, with $c < \bar{c}$. Then in order to minimize the overall cost, a document d should be retrieved next if

$$cP(R | d, q) + \bar{c}(1 - P(R | d, q)) \leq cP(R | d', q) + \bar{c}(1 - P(R | d', q)) \quad (4.8)$$

for every other document d' that has not yet been retrieved. But, since $c < \bar{c}$, the above condition is equivalent to

$$P(R | d, q) \geq P(R | d', q),$$

that is, documents should be retrieved in order of decreasing probability of relevance.

In order to design a model for the probability of relevance some simplifications are needed. The simplest possible approach is called binary independence retrieval (BIR) (Robertson and Spärck Jones 1976) as also used in the Bernoulli model for the Naive Bayes classifier, which we will discuss later, in Section 4.6.2. This model postulates a form of conditional independence amongst terms. Following Fuhr (1992), let us introduce the odds of relevance and apply Bayes' theorem:

$$O(R | d, q) = \frac{P(R = 1 | d, q)}{P(R = 0 | d, q)} = \frac{P(d | R = 1, q)}{P(d | R = 0, q)} \cdot \frac{P(R = 1 | q)}{P(R = 0 | q)}. \quad (4.9)$$

Note that the last fraction is the odds of R given q , a constant quantity across the collection of documents (it only depends on the query). The BIR assumption concerns the first fraction and was originally misidentified as a marginal form of term independence (Cooper 1991). We can state it as

$$\frac{P(d | R = 1, q)}{P(d | R = 0, q)} = \prod_{j=1}^{|V|} \frac{x_j P(\omega_j | R = 1, q) + (1 - x_j)(1 - P(\omega_j | R = 1, q))}{x_j P(\omega_j | R = 0, q) + (1 - x_j)(1 - P(\omega_j | R = 0, q))}. \quad (4.10)$$

The parameters to be estimated are therefore

$$\theta_j \doteq P(\omega_j | R = 1, q)$$

and $\eta_j \doteq P(\omega_j | R = 0, q)$. If we further assume that $\theta_j = \eta_j$ if ω_j does not appear in q we finally have

$$O(R | d, q) = O(R | q) \prod_{j:\omega_j \in q} x_j \frac{\theta_j}{\eta_j} \cdot (1 - x_j) \frac{1 - \theta_j}{1 - \eta_j}, \quad (4.11)$$

where the product only extends to indices j whose associated terms ω_j appear in the query. This can also be rewritten as

$$O(R | d, q) = O(R | q) \cdot \prod_{j \in q} \frac{1 - \theta_j}{1 - \eta_j} \cdot \prod_{\substack{j: \omega_j \in q, \\ x_j=1}} \frac{\theta_j(1 - \eta_j)}{\eta_j(1 - \theta_j)}, \quad (4.12)$$

where the last factor is the only part that depends on the document. The retrieval status value (RSV) of a document is thus computed by taking the logarithm of the last factor:

$$\text{RSV}(d) = \sum_{j: \omega_j \in d \cup q} \log \frac{\theta_j(1 - \eta_j)}{\eta_j(1 - \theta_j)}. \quad (4.13)$$

Documents are eventually ranked in reverse order of RSV.

4.5 Latent Semantic Analysis

The retrieval approach based on vector-space similarities can reach satisfactory recall rates only if the terms in the query are actually present in the relevant documents. Users, on the other hand, often formulate queries choosing terms that express *concepts* related to the contents they want to retrieve. Natural language has a very rich expressive power and even at the lexical level, the large variability due to synonymy and polysemy causes serious problems to retrieval methods based on term matching. Synonymy means that the same concept can be expressed using different sets of terms (e.g. bandit, brigand, and thief). Synonymy negatively affects recall. Polysemy means that identical terms can be used in very different semantic contexts. For example, in English, the word ‘bank’ refers to both a repository where important material is saved (as in blood bank) and also to the slope beside a body of water (as in the bank of the Thames). Polysemy negatively affects precision. Overall, synonymy and polysemy lead to a complex relation between terms and concepts that cannot be captured through simple matching.

Thus, although a query may conceptually be very close to a given set of documents, its associated vector could be orthogonal or nearly orthogonal to those document vectors, simply because the authors of the document and the user have a different usage of language. Another more statistical way of thinking about this is to observe that the number of terms that are present in a document is a rather small fraction of the entire dictionary, reducing the likelihood that two documents use the same set of words to express the same concept.

Latent semantic indexing (LSI) is a statistical technique that attempts to estimate the hidden structure that generates terms given concepts. It uses a linear algebra technique known as singular value decomposition (SVD) to discover the most important associative patterns between words and concepts. LSI is a data-driven method, where a large collection of sentences or documents is employed to discover the statistically most significant co-occurrences of terms.

4.5.1 LSI and text documents

Let X denote a term–document matrix, defined as

$$X = [x_1 \cdots x_n]^T. \quad (4.14)$$

Each row in the matrix is simply the vector-space representation of a document. In LSI we use occurrence counts as components. Each column contains the occurrences of a term in each document in the data set. The LSI technique consists of computing the SVD of X and setting to zero all the singular values except the largest K ones. In practical applications K is often set to values around between 100 and 1000. This can be seen as analogous to the PCA dimensionality reduction discussed in Section A.3: documents are mapped to a lower-dimensional latent semantic space induced by selecting directions of maximum covariance.

We will illustrate more in detail the LSI procedure through an example. Consider the collection of 10 documents in Table 4.1. The first five documents are related to the Linux operating system, and the remaining five to news in the area of genomes. For simplicity we focus only on frequent terms, in particular those occurring at least twice. We remove common words as ‘the’ and ‘goes’. The resulting term–document matrix X^T is shown in the middle of Table 4.1 (we use the transpose of X for the sake of space). Performing SVD in this example yields a singular values matrix $\Sigma = \text{diag}(2.57, 2.49, 1.99, 1.9, 1.68, 1.53, 0.94, 0.66, 0.36, 0.10)$. By setting $\hat{\Sigma} = \text{diag}(2.57, 2.49, 0, 0, 0, 0, 0, 0, 0, 0)$, we obtain the reconstruction of X , $\hat{X} = U\hat{\Sigma}V^T$, shown at the bottom of table 4.1.

As we can see, the reconstructed matrix \hat{X} is not as sparse as the original matrix X . The new term weights account for word co-occurrences and appear to infer relations among words that pertain to synonymy, at least in a loose sense. For example, a query on the term ‘Linux’ would now assign a relatively high score to documents 2 and 3, although they do not contain the search keyword at all. We can explain this result arguing that SVD has inferred a link between ‘Debian’ or ‘Gentoo’ and ‘Linux’ through the co-occurrences of other words (they never occur together in our sample). Similarly, we can note in Table 4.1 that both ‘DNA’ and ‘Dolly’ have document vectors closer to the document vector of ‘genome’.

When performing the numerical computation of the SVD of a very large document matrix, it is important to take advantage of sparsity. A variety of well-known numerical methods for computing the SVD of sparse matrices can be brought to bear (Berry 1992; Berry and Browne 1999; Letsche and Berry 1997).

4.5.2 Probabilistic LSA

The latent semantic indexing method described so far does not have a completely sound probabilistic interpretation. As we have discussed, the approximation of X by \hat{X} obtained by setting to zero all but their first K singular values is optimal in the sense that $\|X - \hat{X}\|_2$ is minimized (amongst all projections in ‘latent spaces’ of dimension

Table 4.1 Example application of LSI. Top: a collection of documents. Terms used in the analysis are underlined. Center: the term–document matrix X^T . Bottom: the reconstructed term–document matrix \hat{X}^T after projecting on a subspace of dimension $K = 2$.

d_1 :	Indian government goes for <u>open-source</u> <u>software</u>
d_2 :	<u>Debian</u> 3.0 Woody <u>released</u>
d_3 :	Wine 2.0 <u>released</u> with fixes for <u>Gentoo</u> 1.4 and <u>Debian</u> 3.0
d_4 :	gnuPOD <u>released</u> : iPod on <u>Linux</u> ... with GPLed <u>software</u>
d_5 :	Gentoo servers running an <u>open-source</u> <u>mysql</u> <u>database</u>
d_6 :	<u>Dolly</u> the <u>sheep</u> not totally identical clone
d_7 :	<u>DNA</u> news: introduced low-cost human <u>genome</u> <u>DNA</u> chip
d_8 :	Malaria-parasite <u>genome</u> <u>database</u> on the Web
d_9 :	UK sets up <u>genome</u> bank to protect rare <u>sheep</u> breeds
d_{10} :	<u>Dolly's</u> <u>DNA</u> Damaged

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
open-source	1	0	0	0	1	0	0	0	0	0
software	1	0	0	1	0	0	0	0	0	0
Linux	1	0	0	1	0	0	0	0	0	0
released	0	1	1	1	0	0	0	0	0	0
Debian	0	1	1	0	0	0	0	0	0	0
Gentoo	0	0	1	0	1	0	0	0	0	0
database	0	0	0	0	1	0	0	1	0	0
Dolly	0	0	0	0	0	1	0	0	0	1
sheep	0	0	0	0	0	1	0	0	1	0
genome	0	0	0	0	0	0	1	1	1	0
DNA	0	0	0	0	0	0	2	0	0	1

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
open-source	0.34	0.28	0.38	0.42	0.24	0.00	0.04	0.07	0.02	0.01
software	0.44	0.37	0.50	0.55	0.31	-0.01	-0.03	0.06	0.00	-0.02
Linux	0.44	0.37	0.50	0.55	0.31	-0.01	-0.03	0.06	0.00	-0.02
released	0.63	0.53	0.72	0.79	0.45	-0.01	-0.05	0.09	-0.00	-0.04
Debian	0.39	0.33	0.44	0.48	0.28	-0.01	-0.03	0.06	0.00	-0.02
Gentoo	0.36	0.30	0.41	0.45	0.26	0.00	0.03	0.07	0.02	0.01
database	0.17	0.14	0.19	0.21	0.14	0.04	0.25	0.11	0.09	0.12
Dolly	-0.01	-0.01	-0.01	-0.02	0.03	0.08	0.45	0.13	0.14	0.21
sheep	-0.00	-0.00	-0.00	-0.01	0.03	0.06	0.34	0.10	0.11	0.16
genome	0.02	0.01	0.02	0.01	0.10	0.19	1.11	0.34	0.36	0.53
DNA	-0.03	-0.04	-0.04	-0.06	0.11	0.30	1.70	0.51	0.55	0.81

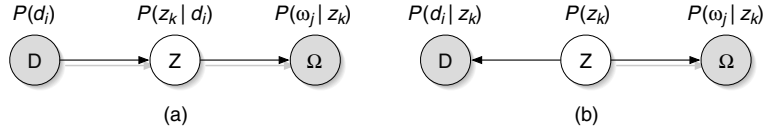


Figure 4.4 Bayesian networks describing the aspect model. Concepts (Z) are never observed in the data but their knowledge would render words (Ω) and documents (D) independent. The two networks are independence equivalent but use different parameterizations.

K). However, the L_2 matrix norm is algebraically well suited for Gaussian variables and is hard to justify its use in the case of the discrete space defined by vectors of counts of words.

We now discuss a probabilistic approach for describing a latent semantic space called the *aspect model* (Hofmann *et al.* 1999) or aggregate Markov model (Saul and Pereira 1997). Let an event in this model be the occurrence of a term ω in a document d . Let $z \in \{z_1, \dots, z_K\}$ denote a latent (hidden) variable associated with each event according to the following generative scheme. First, select a document from a density $P(d)$. Second, select a latent concept z with probability $P(z | d)$. Third, choose a term ω to describe the concept linguistically, sampling from $P(\omega | z)$, i.e. assume that once the latent concept has been specified the document and the term are conditionally independent, as shown by the Bayesian network representation of Figure 4.4. The probability of each event (ω, d) is therefore

$$P(\omega, d) = P(d) \sum_z P(\omega | z) P(z | d). \quad (4.15)$$

The aspect model has a direct interpretation in terms of dimensionality reduction. Each document is uniquely determined by the mixing coordinates $P(z_k | d)$, $k = 1, \dots, K$, that form a nonnegative vector belonging to a probabilistic latent semantic space. In other words, rather than being represented through terms, a document is represented through latent variables that in turn are responsible for generating terms. So if $K \ll |V|$ dimensionality reduction occurs.

The analogy with respect to LSI can be established by introducing the matrices U , V , and Σ whose elements are $u_{ik} = P(d_i | z_k)$, $v_{j,k} = P(\omega_j | z_k)$ and $\sigma_{kk} = P(z_k)$ for $i = 1, \dots, n$, $j = 1, \dots, m$, and $k = 1, \dots, K$. In this way, all the $n \times m$ document–term joint probabilities can be collected in the matrix

$$P = U \Sigma V^T. \quad (4.16)$$

This matrix can be directly compared to the SVD representation used in traditional LSI. Unlike \hat{X} in LSI, P is a properly normalized probability distribution and its entries cannot be negative. Moreover, the coordinates of terms in the latent space can be interpreted as the distribution of terms, conditional on the hidden z_k values.

Hofmann (2001) shows how to fit the parameters of this model from data. In the simplest case, they are estimated by maximum likelihood using the EM algorithm (see Chapter 1) because of the latent variable Z . We assume the parameterization of

Figure 4.4b. During the E step, the probability that the occurrence of ω_j in d_i is due to latent concept z_k is computed as

$$P(z_k | d_i, \omega_j) \propto P(\omega_j | z_k)P(d_i | z_k)P(z_k). \quad (4.17)$$

This can be seen as a special case of belief propagation (see Chapter 1) in the network of Figure 4.4b. The M step simply consists of updating parameters as follows:

$$P(\omega_j | z_k) \propto \sum_{i=1}^n n_{ij} P(z_k | d_i, \omega_j), \quad (4.18)$$

$$P(z_k | d_i) \propto \sum_{j=1}^{|V|} n_{ij} P(z_k | d_i, \omega_j), \quad (4.19)$$

$$P(z_k) \propto \sum_{i=1}^n \sum_{j=1}^{|V|} n_{ij} P(z_k | d_i, \omega_j). \quad (4.20)$$

A possible refinement of the above EM algorithm consists of replacing Equation (4.17) by

$$P(z_k | d_i, \omega_j) \propto (P(\omega_j | z_k)P(d_i | z_k))^\beta P(z_k). \quad (4.21)$$

where β is a positive parameter that is initialized to unity and exponentially decreased in an outer loop that wraps around a full EM optimization (performed with constant β). This outer loop iteration continues as long as predictive performance on a validation set keeps improving. This refinement can help to prevent overfitting and has analogies with the annealing process in physics. The resulting algorithm is called *tempered EM* (TEM). Intuitively, smaller values of β in Equation (4.21) increase the entropy of the posterior distribution of Z performing a special form of regularization.

Once the model is trained, to define a similarity measure based on PLSA, we need to estimate which concepts are being expressed in the query q . As in LSA, documents and queries that are not in the original matrix need to be folded in. In the case of PLSA a point in the latent semantic space is a (posterior) distribution on Z , so folding in consists of calculating $P(z_k | q)$ for $k = 1, \dots, K$. This is done by using the above TEM procedure while keeping $P(\omega_j | z_k)$ fixed.

The original aspect model described above is not without its limitations. The fact that there is a separate set of parameters for each document in the collection means both that there are a very large number of parameters to be estimated and that the model does not directly contain the concept of ‘a new as yet unseen document’ – this can lead to overfitting of the data. More recent work by Blei *et al.* (2002a) has begun to address some of these issues by placing the aspect model in a more Bayesian context.

4.6 Text Categorization

Text categorization consists of grouping textual documents into different fixed classes or categories. This may be useful, for example, to predict the topic of a Web page, or to decide whether it is relevant with respect to the interests of a given user. This problem is particularly well suited for a machine-learning solution. In the following we briefly review three of the most important and effective machine-learning algorithms that are often applied to text categorization: k nearest neighbors (k -NN), Naive Bayes, and support vector machines. Readers not familiar with general concepts in machine learning are urged to read Section 1.5 before continuing this section.

Several additional machine-learning methods not covered here have been applied to text categorization including decision trees (Lewis and Catlett 1994), neural networks (Wiener *et al.* 1995), and symbolic approaches such as inductive logic programming (Cohen 1995) and the induction of classification rules (Apté *et al.* 1994; Cohen 1996). A review of these and other methods can be found in Sebastiani (2002).

4.6.1 k nearest neighbors

k -NN is a memory based classifier that learns by simply storing all the training instances. During prediction, k -NN first measures the distances between a new point \mathbf{x} and all the training instances, returning the set $N(\mathbf{x}, D, k)$ of the k points that are closest to \mathbf{x} . For example, if training instances are represented by real-valued vectors \mathbf{x} , we could use Euclidean distance to measure the distance between \mathbf{x} and all other points in the training data, i.e. $\|\mathbf{x} - \mathbf{x}_i\|_2, i = 1, \dots, n$. After calculating the distances, the algorithm predicts a class label for \mathbf{x} by a simple majority voting rule using the labels in the elements of $N(\mathbf{x}, D, k)$, breaking ties arbitrarily. In spite of its apparent simplicity, k -NN is known to perform well in many domains. An early result by Cover and Hart (1967) shows that the asymptotic error rate of the 1-NN classifier (as the size of the training data set gets infinitely large) is always less than twice the optimal Bayes error (which is the lowest possible error rate achievable by *any* classifier in a particular feature space \mathbf{x}).

In the case of text, majority voting can be replaced by a smoother metric where, for each class c , a scoring function

$$s(c | \mathbf{x}) = \sum_{\mathbf{x}' \in N_c(\mathbf{x}, D, k)} \cos(\mathbf{x}, \mathbf{x}') \quad (4.22)$$

is computed through vector-space similarities between the new documents and the subset of the k neighbors that belong to class c (Yang 1999), where $N_c(\mathbf{x}, D, k)$ is the subset of $N(\mathbf{x}, D, k)$ containing only points of class c . Despite the simplicity of the method, the performance of k -NN in text categorization is quite often satisfactory in practice (Joachims 1998; Lam and Ho 1998; Yang 1999; Yang and Liu 1999). Han *et al.* (2001) have proposed a variant of k -NN where the weights associated with features are learned iteratively – other statistically motivated techniques that extend the basic k -NN classifier are discussed in Hastie *et al.* (2001).

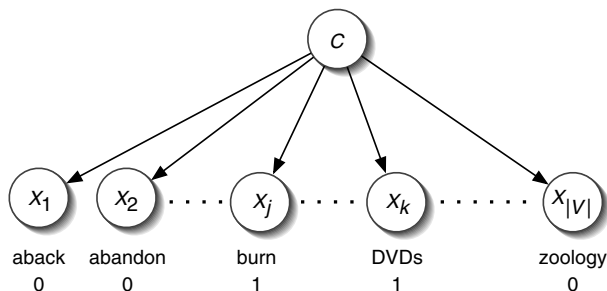


Figure 4.5 A Bayesian network for the Naive Bayes classifier under the Bernoulli document-based event model. The example document is the sentence ‘burn DVDs and we will burn you.’

4.6.2 The Naive Bayes classifier

This classifier attempts to estimate the conditional probability of the class given the document, namely $P(c | d)$, for $c = 1, \dots, S$. Using Bayes’ theorem, we can write this probability as

$$P(c | d, \theta) = \frac{P(d | c, \theta)P(c | \theta)}{P(d | \theta)} \propto P(d | c, \theta)P(c | \theta), \quad (4.23)$$

where θ are the parameters of the model. In what follows we only include the dependence on θ when necessary – otherwise it can be implicitly assumed to be present.

Note that, since the classes are assumed to be mutually exclusive, we do not really need to know $P(d)$, a term that can be thought of as a normalization factor guaranteeing that $\sum_c P(c | d) = 1$. The key idea behind the Naive Bayes classifier is the assumption that the terms in a document are conditionally independent given the class. This assumption is clearly false in many if not most practical situations, but it is often adequate to first-order for the bag-of-words representation, where word order in the document is not taken into account. Moreover, we should keep in mind that we are interested in a good approximation of $P(c | d)$, which does not necessarily mean that we need a perfect model of $P(d | c)$. In other words, we are interested in discrimination among classes, not in a high quality generative model of the document given the class (Ng and Jordan 2002). In practice, the classifier is known to work satisfactorily even when the conditional independence assumption is known not to hold (Domingos and Pazzani 1997).

There is a subtle issue concerning the interpretation of the document in terms of a probabilistic event (Lewis 1998; McCallum and Nigam 1998). If the document as a whole is considered to be an event, then it should be naturally described by a bag of words, and the words are the *attributes* of this event. In this case, each vocabulary term is associated with a Bernoulli attribute whose realization is unity if the term appears in the document, and zero otherwise. In addition to reducing documents to bags of words, the Naive Bayes model postulates that binary attributes are mutually independent given the class. This approach is substantially the same as the binary

independence retrieval model that was popular for probabilistic retrieval in the 1970s (Robertson and Spärck Jones 1976) and has even more ancient origins (Maron 1961). The conditional independence assumption in this model can be depicted graphically using a Bayesian network such as that in Figure 4.5, suggesting that the class is the only cause of the appearance of each word in a document. Under this model, generating a document is like tossing $|V|$ independent coins and the occurrence of each word in the document is a Bernoulli event. Therefore we can rewrite the generative portion of Equation (4.23) as

$$P(d | c, \theta) = \prod_{j=1}^{|V|} x_j P(\omega_j | c) + (1 - x_j)[1 - P(\omega_j | c)], \quad (4.24)$$

where $x_j = 1$ [0] means that word ω_j does [does not] occur in d and $P(\omega_j | c)$ is the probability of observing word ω_j in documents of class c . Here θ represents the set of probabilities (or parameters) $P(\omega_j | c)$, which is the probability of the binary event that word ω_j is ‘turned on’ within class c .

Alternatively, we may think of a document as a sequence of events $W_1, \dots, W_{|d|}$. Each observed W_t has a vocabulary entry (from 1 to $|V|$) as an admissible realization. Note that the number of occurrences of each word, as well as the length of the document, are taken into account under this interpretation. In addition, since the document is a sequence, serial order among words should also be taken into account when modeling $P(W_1, \dots, W_{|d|} | c)$. This could be done, for example, by using a Markov chain. A simplifying assumption, however, is that word occurrences are independent of their (relative) positions, given the class (Lewis 1992; Lewis and Gale 1994; Mitchell 1997). Equivalently, we assume that the bag-of-words representation retains all the relevant information for assessing the probability of a document whose class is known.

Under the word-based event model, generating a document is like throwing a die with $|V|$ faces $|d|$ times, and the occurrence of each word in the document is a multinomial event. Hence, the generative portion of the model is a multinomial distribution

$$P(d | \theta) = GP(|d|) \prod_{j=1}^{|V|} P(\omega_j | c)^{n_j}, \quad (4.25)$$

where n_j is the number of occurrences of word ω_j in d , and $P(\omega_j | c)$ is the probability that word ω_j occurs at any position $t \in [1, \dots, |d|]$; because of the bag-of-words assumption this does not depend on t . Here the parameters θ are the set of probabilities $P(\omega_j | c)$, where now $\sum_{j=1}^{|V|} P(\omega_j | c) = 1$. McCallum and Nigam (1998) found empirically that the multinomial model outperforms the Bernoulli model in several evaluation benchmarks. Note that as in the case of the document-based event model of Equation (4.24), the ‘bag-of-words’ assumption results in a factorization of $P(W_1, \dots, W_{|d|} | c)$ explaining, perhaps, why ‘Naive Bayes’ is often used in the literature to refer to both event models.

The normalization factor is the multinomial coefficient

$$G = \frac{|d|!}{\prod_j n_j!}.$$

Neither $P(|d|)$ nor G are needed for classification, since $|d|$, the number of words or terms in a document, is assumed to be independent of the class. This last assumption can be removed and $P(|d| \mid c)$ explicitly modeled. Models of document length (e.g. based on Poisson distributions) have been used for example in the context of probabilistic retrieval (Robertson and Walker 1994). Note that in the case of the Bernoulli model there are $2^{|V|}$ possible different documents, while in the case of the multinomial model there is an infinite (but countable) number of different documents.

An additional model that may be developed and that lies somewhat in between the Bernoulli and the multinomial models consists of keeping the document-based event model but extending Bernoulli distributions to integer distributions, such as the Poisson (Lewis 1998). Finally, extensions of the basic Naive Bayes approach that allow limited dependencies among features have been proposed (Friedman and Goldszmidt 1996; Pazzani 1996). However, these models are characterized by a larger set of parameters and may overfit the data (Koller and Sahami 1997).

Learning a Naive Bayes classifier consists of estimating the parameters θ from the available data. We will assume that the training data set consists of a collection of labeled documents $\{(d_i, c_i), i = 1, \dots, n\}$. In the Bernoulli model, the parameters θ include $\theta_{c,j} = P(\omega_j \mid c)$, $j = 1, \dots, |V|$, $c = 1, \dots, K$. These are estimated as normalized sufficient statistics

$$\hat{\theta}_{c,j} = \frac{1}{N_c} \sum_{i:c_i=c}^n x_{ij}, \quad (4.26)$$

where $N_c = |\{i : c_i = c\}|$ and $x_{ij} = 1$ if ω_j occurs in d_i . Additional parameters are the class prior probabilities $\theta_c = P(c)$, which are estimated as

$$\hat{\theta}_c = \frac{N_c}{n}. \quad (4.27)$$

Note that the estimates above correspond to ML estimates of the parameters. In Chapter 1 we discussed how we can also derive Bayesian parameter estimates that combine the observed data with prior knowledge. This can be achieved by using (for example) Dirichlet priors for the Bernoulli likelihood model. Bayesian estimates can be quite useful when estimating parameters from sparse data. For words in documents, we might have a weak noninformative prior that any word can in theory appear in documents from any class. This would avoid problems in making predictions on future documents when such a word shows up in some class c where it did not appear in the training data. A model based on ML parameter estimates would assign a probability of zero to that document, irrespective of how well the other words in the document matched class c . On the other hand, a model based on Bayesian estimates would assign that word-class combination a low nonzero probability and still allow the other words

to play a role in determining the final class prediction for the document. We might also have more informative priors available, e.g. a Dirichlet prior on the distribution of words in each class that reflect word distributions as estimated from previous studies, or that reflect an expert's belief in what words are likely to appear in each class.

In the case of the multinomial model of Equation (4.25), the generative parameters are $\theta_{c,j} = P(\omega_j | c)$. Note that these parameters must satisfy $\sum_j \theta_{c,j} = 1$ for each class c . To estimate these parameters it is common practice to introduce Dirichlet priors (see Chapter 1 for details). The resulting estimation equations are derived as follows. In the case of the distributions of terms given the class, we introduce a Dirichlet prior with hyperparameters q_j and α , resulting in the estimation formula

$$\hat{\theta}_{c,j} = \frac{\alpha q_j + \sum_{i:c_i=c}^n n_{ij}}{\alpha + \sum_{l=1}^{|V|} \sum_{i:c_i=c} n_{il}}, \quad (4.28)$$

where n_{ij} is the number of occurrences of ω_j in d_i . A simple noninformative prior assigns $q_j = 1/|V|$ and $\alpha = |V|$. Intuitively, this prior corresponds to the assumption that we have observed each word exactly once in one document of each class. This method (also known as Laplace smoothing) prevents the problem of estimating a null value for a parameter if a certain term ω_j never occurs in documents of a given class c in the training set (without the smoothing, the probability that a new document containing ω_j belong to c would be zero). Similarly, the estimation formula for the (unconditional) class probabilities is

$$\hat{\theta}_c = \frac{q'_c \alpha' + N_c}{\alpha' + n}. \quad (4.29)$$

Also in this case we could assume a noninformative Dirichlet hyperparameters $q'_c = 1/K$ and $\alpha' = K$.

4.6.3 Support vector classifiers

Support vector machines (SVMs) were introduced in Cortes and Vapnik (1995), extending earlier seminal work by Vapnik on statistical learning theory (Vapnik 1982). The basic underlying idea, often referred to as *structural risk minimization* is closely related to the theory of regularization but also to Bayesian approaches to learning (Evgeniou *et al.* 2000) and is essentially guided by the principle that the hypothesis that explains a finite set of examples should be searched in an appropriately 'small' hypothesis space.

SVMs are particularly well suited to deal with high-dimensional data such as vector-space representations of text documents. In their standard formulation, they deal with binary classification problems where the number of classes is restricted to two (see later for a discussion on multiclass SVM).

Consider a training set $D = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$ with $\mathbf{x}_i \in \mathbb{R}^m$, and where $y_i \in \{-1, 1\}$ is an integer that specifies whether \mathbf{x}_i is a positive or a negative example. A

linear discriminant classifier is then defined by introducing the *separating hyperplane*

$$\{x : f(x) = \mathbf{w}^T \mathbf{x} + w_0 = 0\}, \quad (4.30)$$

where $\mathbf{w} \in \mathbb{R}^m$ and $w_0 \in \mathbb{R}$ are adjustable coefficients that play the role of model parameters. A binary classification function $h : \mathbb{R}^m \rightarrow \{0, 1\}$ can be obtained by taking the sign of $f(x)$, i.e.

$$h(x) = \begin{cases} 1, & \text{if } f(x) > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (4.31)$$

Learning in this class of models consists of determining \mathbf{w} and w_0 from data. The training examples are said to be *linearly separable* if there exists a hyperplane whose associated classification function is consistent with all the labels, i.e. if $y_i f(\mathbf{x}_i) > 0$ for each $i = 1, \dots, n$. Under this hypothesis, Rosenblatt (1958) proved that the following simple iterative algorithm terminates and returns a separating hyperplane:

```

PERCEPTRON( $D$ )
1   $\mathbf{w} \leftarrow \mathbf{0}$ 
2   $w_0 \leftarrow 0$ 
3  repeat
4       $e \leftarrow 0$ 
5      for  $i \leftarrow 1, \dots, n$ 
6          do  $s \leftarrow \text{sign}(y_i(\mathbf{w}^T \mathbf{x}_i + w_0))$ 
7              if  $s < 0$ 
8                  then  $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ 
9                       $w_0 \leftarrow w_0 + y_i$ 
10                  $e \leftarrow e + 1$ 
11  until  $e = 0$ 
12  return  $(\mathbf{w}, w_0)$ .

```

It can be shown that a sufficient condition for D to be linearly separable is that the number of training examples $n = |D|$ is less than or equal to $m + 1$ (see Exercise 4.4). This is particularly likely to be true in the case of text categorization, where the vocabulary typically includes several thousands of terms, and is often larger than the number of available training examples n (see Exercise 4.5 for details).

Unfortunately, learning with the Perceptron algorithm offers little defense against overfitting. A thorough explanation of this problem requires concepts from statistical learning theory that are beyond the scope of this book. To gain an intuition of why this is the case, consider the scenario in Figure 4.6. Here we suppose that positive and negative examples are generated by two Gaussian distributions (see Appendix A) with the same covariance matrix and that positive and negative points are generated with the same probability. In such a setting, the optimal (Bayes) decision boundary is the one that minimizes the posterior probability that a new point is misclassified and, as it turns out, this boundary is the hyperplane that is orthogonal to the segment

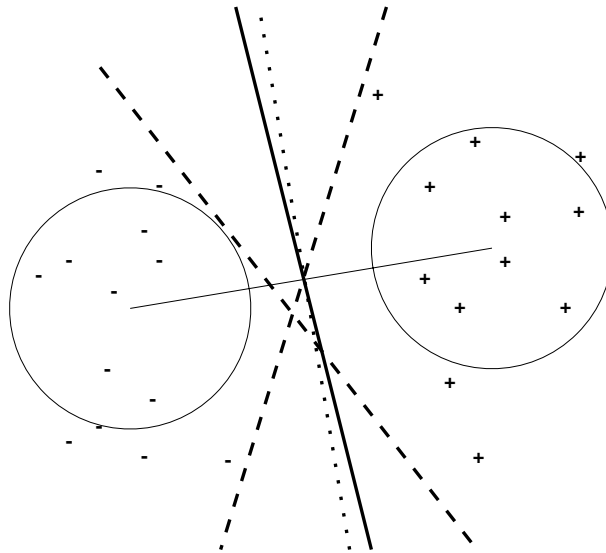


Figure 4.6 Alternative linear decision boundaries for a binary classification problem (see text for explanation).

connecting the centers of mass of the two distributions (dotted line in Figure 4.6). Clearly, a random hyperplane that just happens to separate training points (dashed line) can be substantially far away from the optimal separation boundary, leading to poor generalization to new data. The difficulty grows with the dimensionality of the input space m since for a fixed n the set of separating hyperplanes grows exponentially with m (a problem known as the *curse of dimensionality*). Remember that in the case of text categorization m may be significantly large (several thousands).

The statistical learning theory developed by Vapnik (1998) shows that we can define an *optimal separating hyperplane* (relative to the training set) having two important properties: it is unique for each linearly separable data set, and its associated risk of overfitting is smaller than for any other separating hyperplane. We define the *margin* M of the classifier to be the distance between the separating hyperplane and the closest training examples. The optimal separating hyperplane is then the one having maximum margin (see Figure 4.7). Going back to Figure 4.6, the theory suggests that the risk of overfitting for the maximum margin hyperplane (solid line) is smaller than for the dashed hyperplane. Indeed, in our example the maximum margin hyperplane is significantly closer to the Bayes optimal decision boundary.

In order to compute the maximum margin hyperplane, we begin by observing that the distance of a point \mathbf{x} from the separating hyperplane is

$$\frac{1}{\|\mathbf{w}\|}(\mathbf{w}^T \mathbf{x} + w_0)$$

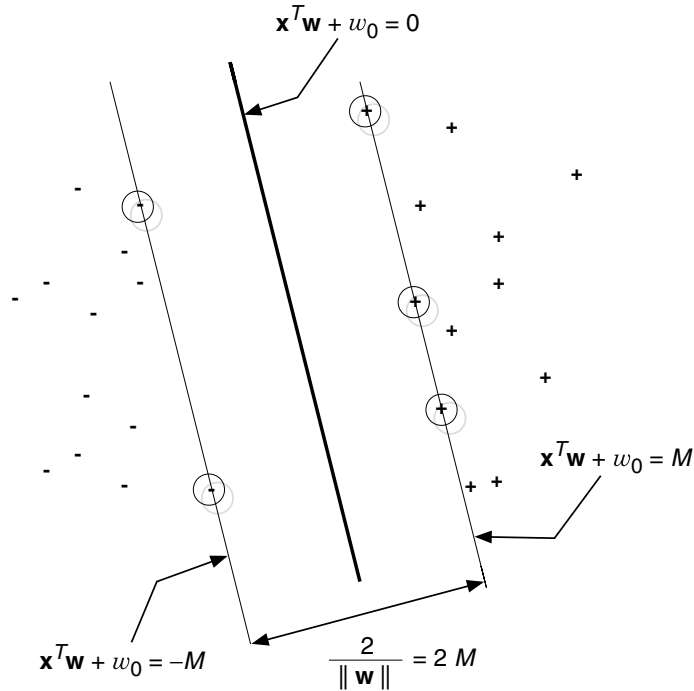


Figure 4.7 Illustration of the optimal separating hyperplane and margin. Circled points are support vectors.

(see Figure 4.7). Thus, the optimal hyperplane can be obtained by solving the constrained optimization problem

$$\max_{\mathbf{w}, w_0} M \quad \text{subject to} \quad \frac{1}{\|\mathbf{w}\|} y_i (\mathbf{w}^T \mathbf{x}_i + w_0) \geq M, \quad i = 1, \dots, n, \quad (4.32)$$

where the constraints require that each training point should lie in the correct semispace and at a distance not less than M from the separating hyperplane. Note that although (\mathbf{w}, w_0) comprise $n + 1$ real numbers, there are only n degrees of freedom since multiplying \mathbf{w} and w_0 by a scalar constant does not change the hyperplane. Thus we can arbitrarily set $\|\mathbf{w}\| = 1/M$ and rewrite the optimization problem (4.32) as

$$\min_{\mathbf{w}, w_0} \|\mathbf{w}\| \quad \text{subject to} \quad y_i (\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i = 1, \dots, n. \quad (4.33)$$

The above problem can be transformed to its dual by first introducing the vector of Lagrangian multipliers $\boldsymbol{\alpha} \in \mathbb{R}^n$ and writing the Lagrangian function

$$\mathcal{L}(D) = -\frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + w_0) - 1] \quad (4.34)$$

and subsequently setting to zero the derivatives of (4.34) with respect to \mathbf{w} , w_0 , obtaining

$$\max_{\boldsymbol{\alpha}} -\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{\Lambda}\boldsymbol{\alpha} + \sum_{i=1}^n \alpha_i \quad \text{subject to } \alpha_i \geq 0, \quad i = 1, \dots, n, \quad (4.35)$$

where $\boldsymbol{\Lambda}$ is an $n \times n$ matrix with $\lambda_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$. This is a quadratic programming (QP) problem that can be solved, in principle, using standard optimization packages. For each training example i , the solution must satisfy the Karush–Kuhn–Tucker condition,

$$\alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + w_0) - 1] = 0, \quad (4.36)$$

and therefore either $\alpha_i = 0$ or $y_i (\mathbf{w}^T \mathbf{x}_i + w_0) = 1$. In other words, if $\alpha_i > 0$ then the distance of point \mathbf{x}_i from the separating hyperplane must be exactly M (see Figure 4.7).

Points with associated $\alpha_i > 0$ are called *support vectors*. The decision function $h(\mathbf{x})$ can be computed via Equation (4.30) or, equivalently, from the following dual form:

$$f(\mathbf{x}) = \sum_{i=1}^n y_i \alpha_i \mathbf{x}^T \mathbf{x}_i. \quad (4.37)$$

We remark that in the case of text categorization it may be very important to exploit the sparseness of the data vectors while computing dot products in Equations (4.35) and (4.37).

If the training data are not linearly separable, then this analysis can be generalized by introducing m nonnegative *slack variables* ξ_i and replacing the optimization problem in Equation (4.33) with

$$\min_{\mathbf{w}, w_0} \|\mathbf{w}\| + C \sum_{i=1}^n \xi_i \quad \text{subject to } \begin{cases} y_i (\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i, & i = 1, \dots, n, \\ \xi_i \geq 0, & i = 1, \dots, n, \end{cases} \quad (4.38)$$

where the constant C controls the cost associated with misclassifications.

This problem can also be dualized in the form

$$\max_{\boldsymbol{\alpha}} -\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{\Lambda}\boldsymbol{\alpha} + \sum_{i=1}^n \alpha_i \quad \text{subject to } 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n, \quad (4.39)$$

yielding another QP problem. The classifier obtained in this way is commonly referred to as a support vector machine (SVM).

Note that solving the QP problem using standard optimization packages would take time $O(n^3)$ (assuming the number of support vector grows linearly with the number of training example). This time complexity is a practical drawback for SVMs. However, several approaches that can reduce complexity substantially have been proposed in the literature (Joachims 1999a; Platt 1999).

If the data are considerably nonlinearly separable then an SVM classifier will have a low accuracy, i.e. even the best linear hyperplane may be quite inferior in terms

of prediction accuracy relative to a good nonlinear decision boundary. The methods developed in this section can be further extended to accommodate nonlinear separation by using kernel functions that map points $\mathbf{x} \in \mathbb{R}^m$ into a higher-dimensional (possibly infinite-dimensional) space called the *feature space*, where data are linearly separable. Details on kernel methods can be found in Cristianini and Shawe-Taylor (2000); Schoelkopf and Smola (2002) and further details on the application of SVM to text categorization can be found in Joachims (2002).

The SVM presented here can only deal with binary classification problems. For more than two classes, a standard approach consists of *reducing* a multiclass problem into several binary sub-problems, coding each class with a binary string. The simplest (redundant) coding strategy is often referred to as ‘one shot’ or ‘one vs all’ and consists of defining as many dichotomies of the instance space as there are classes, where each class is considered as ‘positive’ in one and only one dichotomy. A more general reduction scheme is the information theoretic method based on error-correcting output codes, introduced by Dietterich and Bakiri (1995) and more recently extended in Allwein *et al.* (2000). Typically, these binary classifiers are trained independently but recent work (Crammer and Singer 2000; Guermeur *et al.* 2000) also considers the case where classifiers are trained simultaneously. The extraction of conditional probabilities from multiclass SVM is studied in Passerini *et al.* (2002). Note that the use of error-correcting codes is not limited to inherently binary classifiers like SVMs. For example, Ghani (2000) apply error-correcting codes in conjunction with the Naive Bayes classifier, reporting good results in text categorization tasks with many classes.

4.6.4 Feature selection

Even methods like SVMs that are especially well suited for dealing with high dimensional data (such as vectorial representations of text) can suffer if many terms are *irrelevant* for class discrimination. Feature selection is a dimensionality reduction technique that attempts to limit overfitting by identifying irrelevant components of data points and has been extensively studied in pattern recognition (Kittler 1986) and in machine learning (Kira and Rendell 1992; Koller and Sahami 1996; Langley 1994; Liu and Motoda 1998).

Methods essentially fall into one of two categories: filters and wrappers. Filters attempt to determine which features are relevant *before* learning actually takes place. Wrapper methods, on the other hand, are based on estimates of the generalization error computed by running a specific learning algorithm and searching for relevant features by minimizing the estimated error (Kohavi and John 1997). Although wrapper methods are in principle more powerful, in practice their usage is often hindered by the computational cost. Moreover, they can overfit the data if used in conjunction with classifiers having high capacity.

Information theoretic approaches are known to perform well for filter methods. In particular, information gain is a popular metric in several machine-learning contexts, including the induction of decision trees (Quinlan 1986). It measures the information

about the class that is provided by the observation of each term. More precisely, let us denote by W_j an indicator variable such that $W_j = 1$ means that ω_j appears in a certain document. The information gain of W_j is then the mutual information $\mathcal{I}(C, W_j)$ between the class C and W_j (see Appendix A for a review of the main concepts in information theory). This is also the difference between the marginal entropy of the class $\mathcal{H}(C)$ and the conditional entropy $\mathcal{H}(C | W_j)$ of the class, given W_j :

$$G(W_j) = \mathcal{H}(C) - \mathcal{H}(C | W_j) = \sum_{c=1}^K \sum_{\omega_j=0}^1 P(c, \omega_j) \log \frac{P(c, \omega_j)}{P(c)P(\omega_j)}. \quad (4.40)$$

Note that if C and W_j are independent, the mutual information is zero. Filtering index terms simply consists of sorting terms by information gain and keeping only the k terms with highest gain. Information gain has been used extensively for text categorization (Craven *et al.* 2000; Joachims 1997; Lewis and Ringuette 1994; McCallum and Nigam 1998; Yang 1999) and has been generally found to improve classification performance compared to using all words.

One limitation of information gain is that relevance assessment is done separately for each attribute and the effect of co-occurrences is not taken into account. However, terms that individually bring little information about the class might bring significant information when considered together. In the framework proposed by Koller and Sahami (1996), whole sets of features are tested for relevance about the class. Let \mathbf{x} denote a point in the complete feature space (i.e. the entire vocabulary V in the case of text documents) and \mathbf{x}_G be the projection of \mathbf{x} into a subset $G \subset V$. In order to evaluate the quality of G to represent the class, we measure the distance between $P(c | \mathbf{x})$ and $P(c | \mathbf{x}_G)$ using the average relative entropy:

$$\Delta_G = \sum_{\mathbf{x}} P(\mathbf{x}) P(c | \mathbf{x}) \log \frac{P(c | \mathbf{x})}{P(c | \mathbf{x}_G)}. \quad (4.41)$$

The optimal set of features should yield a small Δ_G . Clearly this setup is only theoretical, since Equation (4.41) is computationally intractable and the distributions involved are hard to estimate accurately. Koller and Sahami (1996) use the notion of a *Markov blanket* to reduce complexity. A set of features $M \subset V$ (with W_j not in M) is a Markov blanket for W_j if W_j is conditionally independent of all the features in $V \setminus (M \cup \{W_j\})$ given M (see also the theory of Bayesian networks in Chapter 1). In other words, once the features in M are known, W_j and the remaining features are independent. Thus the class C is also conditionally independent of W_j given M and, as a result, if G contains a Markov blanket for W_j then $\Delta_{G_j} = \Delta_G$, where $G_j = G \setminus \{W_j\}$. The feature selection algorithm can then proceed by removing those features for which a Markov blanket can be found. Koller and Sahami (1996) prove that a greedy approach where features are removed iteratively is correct in the sense that a feature deemed irrelevant and removed during a particular iteration cannot become relevant again after some other features are later removed. Moreover, since

finding a Markov blanket may be computationally very hard and an ‘exact’ blanket may not exist, they suggest the following approximate algorithm:

```

APPROXMARKOVBLANKET( $D, V, k, n'$ )
1   $G \leftarrow V$ 
2  repeat
3      for  $W_j \in G$ 
4          do for  $W_i \in G \setminus \{W_j\}$ 
5              do  $\rho_{ij} \leftarrow \frac{\text{cov}[W_i, W_j]}{\sqrt{\text{var}[W_i]\text{var}[W_j]}}$ 
6                   $M_j \leftarrow k$  features having highest  $\rho_{ij}$ 
7                   $\mathbf{p}_j \leftarrow P(c|\mathbf{X}_{M_j} = \mathbf{x}_{M_j}, W_j = x_j)$ 
8                   $\mathbf{p}'_j \leftarrow P(c|\mathbf{X}_{M_j} = \mathbf{x}_{M_j})$ 
9                   $D(\mathbf{x}_{M_j}, x_j) \leftarrow \mathcal{H}(\mathbf{p}_j, \mathbf{p}'_j)$ 
10                  $\Delta(W_j|M_j) \leftarrow \sum_{\mathbf{x}_{M_j}, x_j} P(\mathbf{x}_{M_j}, x_j) D(\mathbf{x}_{M_j}, x_j)$ 
11                  $j^* \leftarrow \arg \min_j \Delta(W_j|M_j)$ 
12                  $G \leftarrow G \setminus \{W_{j^*}\}$ 
13     until  $|G| = n'$ 
14 return  $G$ 

```

At each step, the algorithm computes for each feature W_j the set $M_j \subset (G \setminus \{W_j\})$ containing the k features that have the highest correlation with W_j , where k is a parameter of the algorithm and where correlation is measured by the Pearson correlation coefficient in line 5. Then, in line 8, the quantity $\Delta(W_j | M_j)$ is computed as the average cross entropy between the conditional distributions of the class that result from the inclusion and the exclusion of feature W_j . This quantity is clearly zero if M_j is a Markov blanket. Thus, picking j^* that minimizes it (line 9) selects a feature for which an approximate Markov blanket exists. Such a feature is removed and the process iterated until n' features remain in G .

Several other filter approaches have been proposed in the context of text categorization, including the use of minimum description length (Lang 1995), and symbolic rule learning (Raskinis and Ganascia 1996). A comparison of alternative techniques is reported in Yang (1999).

4.6.5 Measures of performance

The performance of a hypothesis function $h(\cdot)$ with respect to the true classification function $f(\cdot)$ can be measured by comparing $h(\cdot)$ and $f(\cdot)$ on a set of documents D_t whose class is known (test set). In the case of two categories, the hypothesis can be completely characterized by the confusion matrix

Predicted Category	Actual category	
	-	+
-	TN	FN
+	FP	TP

where TP, TN, FP, and FN mean true positives, true negatives, false positives, and false negatives, respectively. In the case of balanced domains (i.e. where the unconditional probabilities of the classes are roughly the same) *accuracy* A is often used to characterize performance. Under the 0-1 loss (see Section 1.5 for a discussion), accuracy is defined as

$$A = \frac{\text{TN} + \text{TP}}{|D_t|}. \quad (4.42)$$

Classification error is simply $E = 1 - A$. If the domain is unbalanced, measures such as precision and recall are more appropriate. Assuming (without losing generality) that the number of positive documents is much smaller than the number of negative ones, precision is defined as

$$\pi = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.43)$$

and recall is defined as

$$\rho = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (4.44)$$

A complementary measure that is sometimes used is *specificity*

$$\sigma = \frac{\text{TN}}{\text{TN} + \text{FN}}. \quad (4.45)$$

As in retrieval, there is clearly a trade-off between false positives and false negatives. For example, when using a probabilistic classifier like Naive Bayes we might introduce a decision function that assigns a document the class ‘+’ if and only if $P(c | d) > t$. In so doing, small values of the threshold t yield higher recall and larger values yield higher precision. Something similar can be constructed for an SVM classifier using a threshold on the distance between the points and the separating hyperplane. Often this trade-off is visualized on a parametric plot where precision and recall values, $\pi(t)$ and $\rho(t)$ are evaluated for different values of the threshold (see Figure 4.9 for some examples). Sometimes these plots are also called ROC curves (from Receiver Operating Characteristic) neglecting the caveat that the original name was coined in clinical research for diagrams plotting specificity versus sensitivity (an alias for precision). A very common measure of performance that synthesizes a precision-recall curve is the *breakeven point*, defined as the best⁴ point where $\pi(t^*) = \rho(t^*)$ and can be seen as an alternative to the F_β measure (discussed earlier in Section 4.3.3) for reducing performance to a single number.

In the case of multiple categories we may define precision and recall separately for each category c , treating the remaining classes as a single negative class (see Table 4.2 for an example). Interestingly, this approach also makes sense in domains where the same document may belong to more than one category. In the case of multiple categories, a single estimate for precision and a single estimate for recall can

⁴ Since in general there may be more than one value of the threshold τ at which $\pi(t) = \rho(t)$, we take the value t^* that maximizes $\pi(t^*) = \rho(t^*)$.

be obtained by averaging results over classes. Averages, however, can be obtained in two ways. When *microaveraging*, correct classifications are first summed individually:

$$\pi^\mu = \frac{\sum_{c=1}^K \text{TP}_c}{\sum_{c=1}^K \text{TP}_c}, \quad (4.46)$$

$$\rho^\mu = \frac{\sum_{c=1}^K \text{TP}_c + \text{FP}_c}{\sum_{c=1}^K \text{TP}_c + \text{FN}_c}. \quad (4.47)$$

When *macroaveraging*, precision and recall are averaged over categories:

$$\pi^M = \frac{1}{K} \sum_{c=1}^K \pi_c, \quad (4.48)$$

$$\rho^M = \frac{1}{K} \sum_{c=1}^K \rho_c. \quad (4.49)$$

Compared to microaverages, macroaverages tend to assign a higher weight to classes having a smaller number of documents.

4.6.6 Applications

Up to this point we have largely discussed the classification of generic text documents as represented by a ‘bag-of-words’ vector representation. From this viewpoint it did not really matter whether the bag of words represented a technical article, an email, or a Web page. In practice, however, when applying text classification to Web documents there are several domain-specific aspects of the Web that can be leveraged to improved classification performance. In this section we review some typical examples that demonstrate how ideas from text classification can be applied and extended in a Web context.

Classification of Web pages

Craven *et al.* (2000) have tackled the difficult task of extracting information from Web documents in order to automatically generate knowledge bases (KB). In their WEB \rightarrow KB system, information extraction is carried out by first training machine-learning subsystems to make predictions about classes and relations, and then using the trained subsystems to populate an actual KB starting from data collected from the Web (see Chapter 6 for details on gathering collections of Web pages). Users are expected to provide as input an *ontology* (i.e. a formal description of classes and relations of interest) and training examples of actual instances of classes and relations. A sample ontology for modeling academic websites is shown in Figure 4.8.

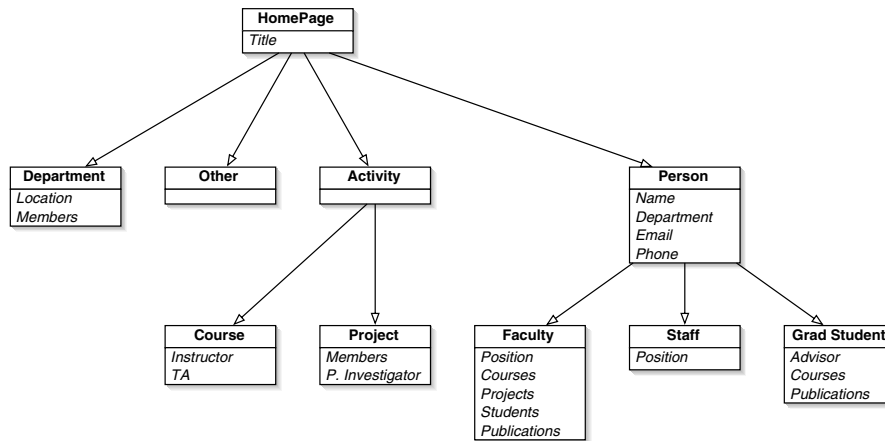


Figure 4.8 A sample ontology for representing knowledge about academic websites. The diagram indicates the class hierarchy and the main attributes of each class. Figure adapted from Craven *et al.* (2000).

Knowledge extraction consists essentially of

- (1) assigning a new Web page to one node of the class hierarchy, and
- (2) filling in the class attributes by extracting the relevant information from the document.

For example, once the system has gathered a home page of a professor at your university, it should first assign the page to the correct class (faculty) and then it should determine the value of attributes such as the email address, the courses taught, and the published papers. The first problem can be conveniently formulated as a text categorization task. In particular, Craven *et al.* (2000) study a Naive Bayes classifier to discriminate between the seven categories that appear as the leaves of the hierarchy shown in Figure 4.8. Their results are shown in Table 4.2 in the form of a confusion matrix. Each entry γ_{kl} for $k, l = 1, \dots, K$ is the number of documents in class l that are assigned to class k , i.e. nondiagonal entries represent misclassifications. Precision for each class k is obtained by dividing γ_{kk} (the number of documents in class k that are correctly classified) by the total number of documents assigned to class k , $\sum_{l=1}^K \gamma_{kl}$. Similarly the recall for class l is obtained dividing γ_{kl} by the number of documents that actually belong to class l , i.e. $\sum_{k=1}^K \gamma_{kl}$.

Classification of news stories

The Reuters-21578 collection is perhaps the most widely used benchmark for text categorization systems (Lewis 1997). It consists of 21 578 stories that appeared in 1987 in the Reuters newswire, assembled and manually labeled by personnel from Reuters Ltd and Carnegie Group, Inc. There are 672 categories and each story may belong to more than one category. 148 of the categories contain at least 20 docu-

Table 4.2 Experimental results obtained in Craven *et al.* (2000) using the Naive Bayes classifier on the WEB \rightarrow KB domain.

Predicted category	Actual category							Precision
	Cou	Stu	Fac	Sta	Pro	Dep	Oth	
Cou	202	17	0	0	1	0	552	26.2
Stu	0	421	14	17	2	0	519	43.3
Fac	5	56	118	16	3	0	264	17.9
Sta	0	15	1	4	0	0	45	6.2
Pro	8	9	10	5	62	0	384	13.0
Dep	10	8	3	1	5	4	209	1.7
Oth	19	32	7	3	12	0	1064	93.6
Recall	82.8	75.4	77.1	8.7	72.9	100.0	35.0	

Table 4.3 Results reported by Joachims (1998) and Weiss *et al.* (1999) (last row) on 90 classes of the Reuters-21578 collection. Performance is measured by microaveraged breakeven point.

Prediction method	Performance breakeven (%)
Naive Bayes (linear)	73.4
Rocchio (linear)	78.7
Decision tree C4.5	78.9
k -NN	82.0
Rule induction	82.0
Support vector (RBF)	86.3
Voting multiple decision trees	87.8

ments. The data set has been split into training and test data according to several alternative conventions (see Lewis (1997) and Sebastiani (2002) for discussion). In the so-called ModApte split, 9603 documents are reserved for training and 3299 for testing (the rest being discarded). 90 categories possess at least one training and one test example. In this setting, Joachims (1998) experimented with several alternative classifiers, including those described in this chapter (see Table 4.3). Note that the support vector classifier used in Joachims (1998) applies a radial basis function (RBF) kernel to learn nonlinear separation surfaces (see Schoelkopf and Smola (2002) for a thorough discussion of kernel methods). It is worth noting that the simple k -NN classifier achieves relatively good performance and outperforms Naive Bayes in this problem. On the same data set, Weiss *et al.* (1999) have reported better results using multiple decision trees trained with AdaBoost (Freund and Schapire 1996; Schapire and Freund 2000) – specific results are provided in the last row of Table 4.3.

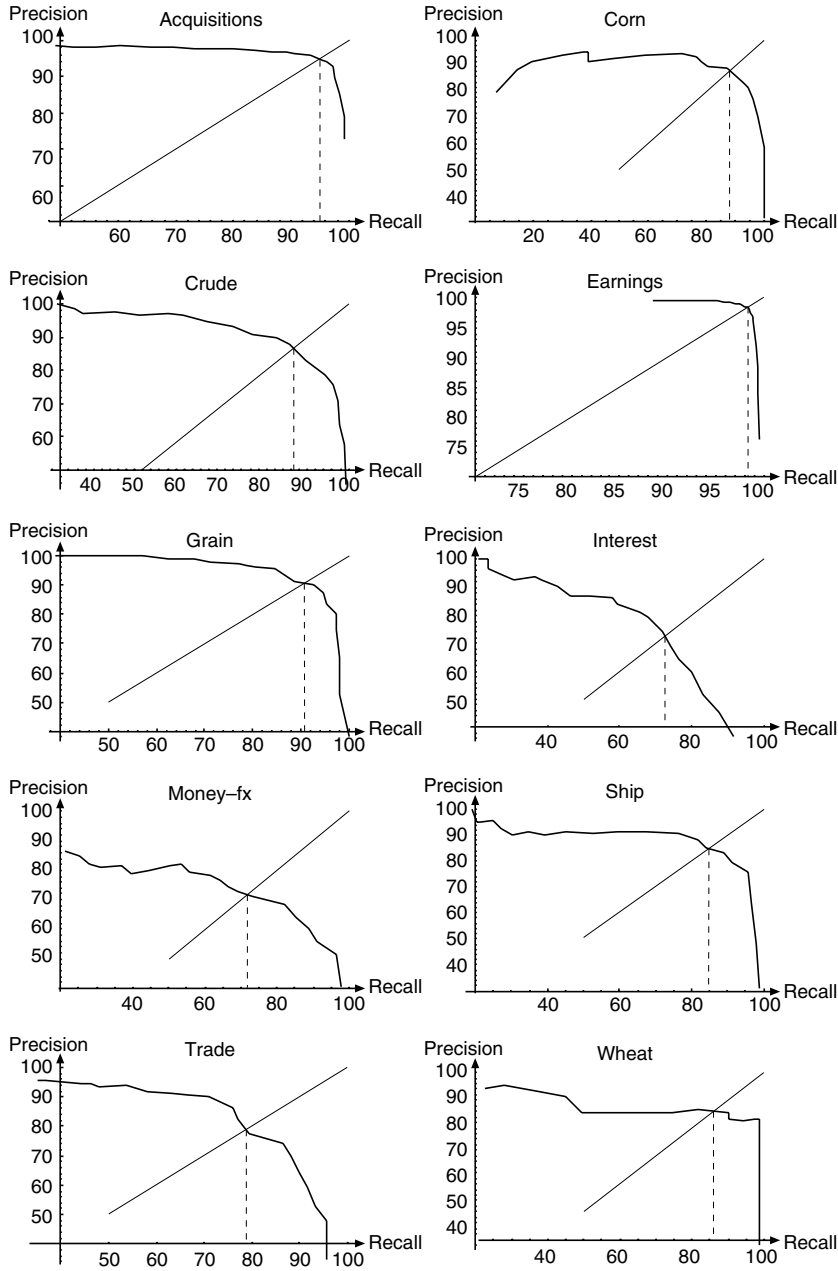


Figure 4.9 Precision-recall curves obtained by training an SVM classifier on the Reuters-21578 data set (ModApte split, 10 most frequent classes), where each plot is for one of the 10 classes.

Table 4.4 Experimental results obtained by Sahami *et al.* (1998) in a junk email filtering problem.

Features	Junk		Not junk	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
Words	97.1	94.3	87.7	93.4
Terms	97.6	94.3	87.8	93.4
Terms and features	100.0	98.3	96.2	100.0

A simplified problem in the Reuters data set is obtained by removing all but the 10 most populated categories. In this setting, Dumais *et al.* (1998) report a comparison of several alternative learners obtaining their best performance (92% microaveraged breakeven point) with support vector machines. Figure 4.9 shows precision-recall curves we have obtained on this data set using an SVM classifier.

Email and news filtering

Sahami *et al.* (1998) have applied the Naive Bayes classifier to the problem of junk email filtering. This application is an example of a problem where hand-programming for data preprocessing (before learning) is particularly important. In this domain, the bag-of-words representation removes important information related to word proximity. Better features can be obtained by assessing the occurrence of terms such as ‘confidential message’, ‘urgent and personal’, or ‘hot to make \$’, or perhaps by having features associated with regular expressions that detect unusual usage of punctuation. Sahami *et al.* (1998) manually identified a set of such terms and, in addition, enriched the textual representation with other features such as the TLD of the sender’s address, attachments, timestamps, etc.

Feature selection in Sahami *et al.* (1998) was performed by eliminating rare words and then retaining the 500 most informative features as determined by mutual information. Results obtained on a data set of 1538 training messages and 251 test messages are reported in Table 4.4. The data set contained 1578 junk messages and 211 legitimate messages. Since the loss of false positives is clearly higher than the loss associated with false negatives, Sahami *et al.* (1998) proposed to classify a message as junk only if the probability predicted by Naive Bayes was greater than 99.9% (an empirically determined threshold). Good results with Naive Bayes have also been obtained by Androutsopoulos *et al.* (2000) on a publicly available data set (<http://www.aueb.gr/users/ion/publications.html>). A comparison of alternative learning methods, including SVM, is reported in Drucker *et al.* (1999).

NewsWeeder is a personalized filtering system for Usenet articles which uses relevance feedback to score news according to user interests (Lang 1995). The data set collected for training the NewsWeeder system contains 19 997 articles evenly sampled from 20 newsgroups and has been widely used as a benchmark for text categorization

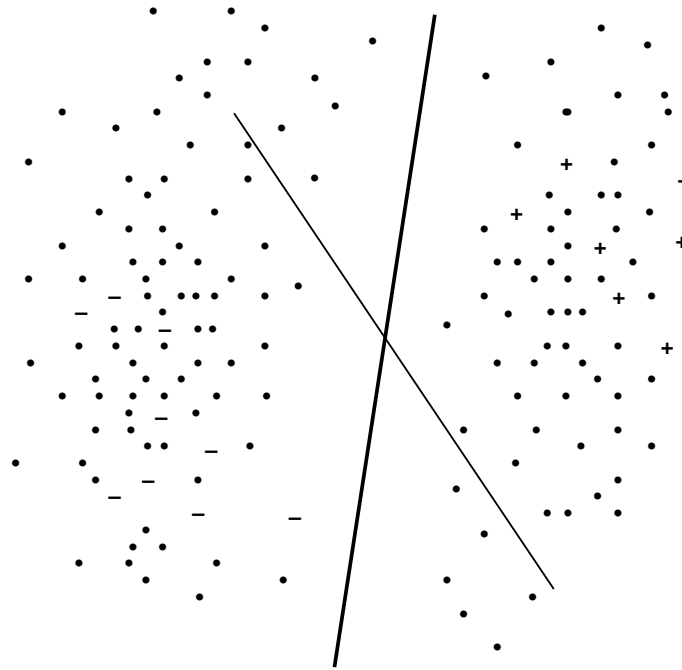


Figure 4.10 How unlabeled data (circles) may help categorization in the case of large margin classifiers.

systems (the task being to predict which newsgroup a message was posted to). The data set of 20 newsgroups is available at <http://www.ai.mit.edu/people/jrennie/20Newsgroups/>.

4.6.7 Supervised learning with unlabeled data

The classifiers studied so far learn from labeled examples of text documents, i.e. the category of the examples must be known. Labels must be assigned by human experts and the process is clearly costly and time consuming. On the other hand, unlabeled documents are abundantly available in many domains, particularly in the case of Web documents. The available data can be written as $D \cup D'$, where $D = (d_i, f(d_i))$, $i = 1, \dots, n$, is the labeled data set and $D' = \{d'_j, j = 1, \dots, n'\}$ are unlabeled documents. Is there any useful information about classes in D' ? Although it may seem counterintuitive, the answer is yes, at least in principle, since unlabeled points allow us to better characterize the shape of the data distribution.

There are several intuitive explanations of why this is the case. First, suppose positive and negative examples are generated by two separate distributions, for example, two Gaussians, and that a very large number of samples are available. In this case we may expect that the parameters of the two distributions can be estimated well, for

example, by training a mixture of two Gaussians. Then, just a few labeled points may be sufficient to decide which Gaussian is associated with the positive and negative class (see Castelli and Cover (1995) for a theoretical discussion). A different intuition, in terms of discriminant classifiers, can be gained by considering Figure 4.10, where unlabeled data points are shown as dots. The thicker hyperplane is clearly a better solution than the thin maximum margin hyperplane computed from labeled points only. Finally, as noted by Joachims (1999b), learning from unlabeled data is reasonable in text domains because categories can be guessed using term co-occurrences. For example, consider the 10 documents of Table 4.1 and suppose category labels are only known for documents d_1 and d_{10} ('Linux' and 'DNA', respectively). Then it should be clear that term co-occurrences allow us to infer categories for the remaining eight unlabeled documents. This observation links text categorization to LSI, a connection that has been exploited for example in Zelikovitz and Hirsh (2001).

None of the classification algorithms we have studied so far can deal directly with unlabeled data. We now present two approaches that develop the intuition above. The use of unlabeled data is further discussed in the context of co-training in Section 4.7.

EM and Naive Bayes

Nigam *et al.* (2000) propose a solution based on the Naive Bayes classifier and the EM algorithm for handling unlabeled data. In their setting, the class variable C for unlabeled documents is interpreted as a missing variable that can be filled in using EM. More precisely, the E step consists of computing, for each document $d_i \in D_u$, the conditional probability $P(c | d_i)$ (in the case of labeled documents this probability is clearly unity if $c = c_i$ and zero otherwise). This probability is then used to compute the expected sufficient statistics for the parameters, which are in turn used as 'soft' counts for parameters re-estimation. Under the word-based multinomial event model of Equation (4.25), the re-estimation equations in the M step are

$$\hat{\theta}_{c,j} = \frac{\alpha q_j + \sum_{i=1}^n n_{ij} P(c | d_i)}{\alpha + \sum_{k=1}^{|V|} \sum_{i=1}^n n_{ik} P(c | d_i)} \quad (4.50)$$

and

$$\hat{\theta}_c = \frac{q'_c \alpha' + \sum_{i=1}^n P(c | d_i)}{\alpha' + n}. \quad (4.51)$$

Nigam *et al.* (2000) validated the above method on three data sets: the 20 news-groups, WEB \rightarrow KB, and Reuters-21578 (ModApte split). For brevity we only report their results on the first data set in Figure 4.11. Note the significant difference of accuracy between Naive Bayes with no unlabeled documents and EM with 10 000 unlabeled documents when the number of labeled documents in the training set is small. As the number of labeled documents increases, the parameters of the Naive Bayes classifier are estimated more reliably and the two approaches tend to the same accuracy.

Transductive SVM

Support vector machines can be also be extended to handle nonlabeled data in the *transductive* learning framework of Vapnik (1998). In this setting, the optimization problem of Equation (4.33) (that leads to computing the optimal separating hyperplane for linearly separable data) becomes:

$$\min_{y'_1, \dots, y'_n, \mathbf{w}, w_0} \|\mathbf{w}\| \quad \text{subject to} \quad \begin{cases} y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, & i = 1, \dots, n, \\ y'_j(\mathbf{w}^T \mathbf{x}'_j + w_0) \geq 1, & j = 1, \dots, n'. \end{cases} \quad (4.52)$$

The solution is found by determining a label assignment (y'_1, \dots, y'_n) of the training examples so that the separating hyperplane maximizes the margin of both the data in D and D' . In other words missing values (y'_1, \dots, y'_n) are ‘filled in’ using maximum margin separation as a guiding criterion. A similar extension of Equation (4.38) for nonlinearly separable data is

$$\min_{y'_1, \dots, y'_n, \mathbf{w}, w_0} \|\mathbf{w}\| + C \sum_{i=1}^n \xi_i + C' \sum_{j=1}^{n'} \xi'_j$$

$$\text{subject to} \quad \begin{cases} y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i, & i = 1, \dots, n, \\ y'_j(\mathbf{w}^T \mathbf{x}'_j + w_0) \geq 1 - \xi'_j, & j = 1, \dots, n', \\ \xi_i \geq 0, & i = 1, \dots, n, \\ \xi'_j \geq 0, & j = 1, \dots, n'. \end{cases} \quad (4.53)$$

Clearly an exact solution of the above transductive inference problem is intractable, as it involves searching in the space of all possible label assignments on n' points.

Joachims (1999b) proposes a heuristic algorithm that starts assigning points in D' , using the labeling induced by the classifier obtained by training a standard SVM on D . The algorithm then iteratively improves the solution by switching the labels of a positive and negative example in D' if the switch leads to a reduction of the objective function. Joachims (1999b) reports empirical results on Reuters-21578 (ModApte split with 10 categories) and on a subset of WEB \rightarrow KB data. The results show that using a very small number of labeled documents still yields acceptable performance. For example, Joachims found that the average breakeven point of the transductive SVM was 60.8% on the Reuters-21578 data set (ModApte split, 10 most frequent classes) when using a data set D of 17 labeled documents and a test set D' of 3299 documents. By comparison, training a standard SVM on the same D and testing on the same D' yields an average breakeven score of 48.4%. Similarly, in the WEB \rightarrow KB domain the average breakeven score was found to improve from 48.6% to 53.5% using a training set $|D| = 9$ and a test set $|D'| = 3957$. Although positive, these results are obtained in a quite specific experimental setup: the labeled data set D is a subset of the training data specified in the ModApte split. However, the unlabeled data D' are the ModApte test documents, i.e. the same documents where precision and recall are later estimated. Zhang and Oles (2000) repeated the same experiment using a different data split and

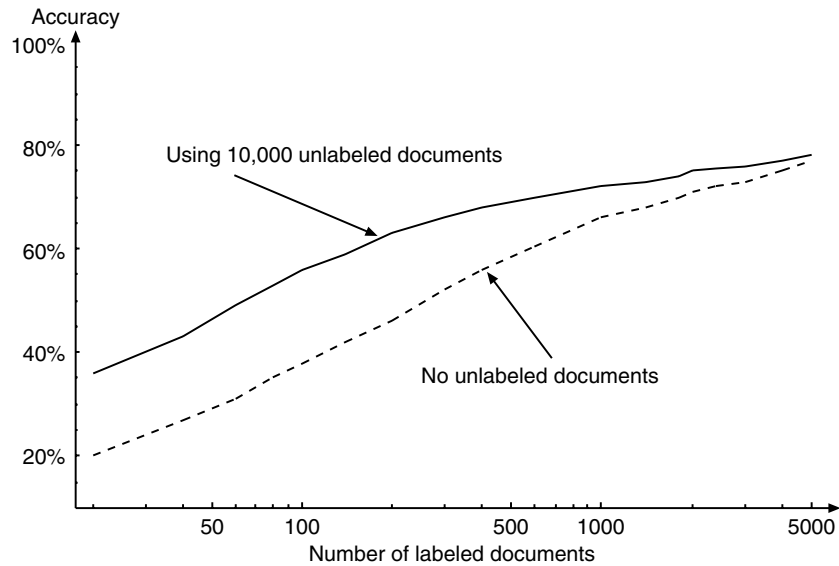


Figure 4.11 Classification accuracy as a function of the number of labeled documents using EM and 10 000 unlabeled documents (solid line) and using no unlabeled documents and the Naive Bayes classifier (dotted line). Experimental results reported in (Nigam *et al.* 2000) for the 20 newsgroups data set.

found that the transductive SVM actually maximizes the ‘wrong margin’ i.e. pushes toward a large separation of unlabeled data but achieve this by actually mislabeling the unlabeled data. In the setting of Zhang and Oles (2000) the performance of the transductive SVM was found to be worse than that of the inductive SVM.

4.7 Exploiting Hyperlinks

Text analysis in the case of Web documents can take advantage of the hyperlinked structure to gain additional information about a document that is not contained in the document itself. For example, the text in the source anchor text is very often a synthetic description of the contents of the page being linked to. Even if we did not observe a target document $d(v)$, we can gain information about this document from source anchors text in all the documents $d(w)$ such that $w \in \text{pa}[v]$. In the case of retrieval, for example, anchor text has been very usefully exploited to score target pages (Brin and Page 1998).

4.7.1 Co-training

Links in hypertexts offer additional information that can also be exploited for improving categorization and information extraction. The co-training framework introduced

by Blum and Mitchell (1998) especially addresses this specific property of the Web, although it has more general applicability. In co-training, each instance is observed through two alternative sets of attributes or ‘views’ and it is assumed that each view is sufficient to determine the class of the instance. More precisely, the instance space \mathcal{X} is factorized into two subspaces $\mathcal{X}_1 \times \mathcal{X}_2$ and each instance \mathbf{x} is given as a pair $(\mathbf{x}_1, \mathbf{x}_2)$. For example, \mathbf{x}_1 could be the bag of words of a document and \mathbf{x}_2 the bag of words obtained by collecting all the text in the anchors pointing to that document. Blum and Mitchell (1998) assume that

- (1) the labeling function that classifies examples is the same as applied to \mathbf{x}_1 or to \mathbf{x}_2 , and
- (2) \mathbf{x}_1 and \mathbf{x}_2 are conditionally independent given the class.

Under these assumptions, unlabeled documents can be exploited in a special way for learning. Specifically, suppose two sets of labeled and unlabeled documents D and D' are given. The iterative algorithm described in Blum and Mitchell (1998) then proceeds as follows. Labeled data are used to infer two Naive Bayes classifiers, one that only considers the \mathbf{x}_1 portion of \mathbf{x} and one that only considers the portion \mathbf{x}_2 . Then these two classifiers are used to guess class labels of documents in a subset of D' . A fixed amount of such instances that are classified as positive with highest confidence is then added as positive examples to D . A similar procedure is used to add ‘self-labeled’ negative examples to D and the process iterated, retraining the two Naive Bayes classifiers on both labeled and self-labeled data. Blum and Mitchell (1998) report a significant error reduction in an empirical test on $\text{WEB} \rightarrow \text{KB}$ documents. Nigam and Ghani (2000) have shown that co-training offers significant advantages over EM if there is a true independence between the two feature sets.

4.7.2 Relational learning

Relational learning is another interesting approach for exploiting hypertextual information. In this setting, data are assumed to be available in a relational format and a learning algorithm can exploit relations among data items. In the case of Web data, there are obvious relations that are encoded in the hyperlinked structure of Web pages, but also local relations associated with the semi-structured organization of text as determined by the descriptive markup in HTML. As in the co-training framework described above, the general intuition is that, as well as the text on the pages being relevant to the classification of the page, the hyperlinks also contain useful information about the category a page belongs to.

FOIL (Quinlan 1990) is a classic algorithm for dealing with relational data and learning first-order clauses such as those used in the Prolog programming language. Although a description of FOIL is out of the scope of this book, it is interesting to remark that these approaches have been exploited with success to take hypertextual information into account. Craven *et al.* (2000) applied FOIL to learn classification rules in the $\text{WEB} \rightarrow \text{KB}$ domain (Figure 4.8 shows an ontology for that domain).

Table 4.5 Examples of first-order logic classification rules learned from WEB \rightarrow KB data (Craven *et al.* 2000). Note that terms such as *jame* and *faculti* result from text conflation.

student(A)	$\text{:- not(has_data(A)), not(has_comment(A)), link_to(B,A), has_jame(B), has_paul(B), not(has_mail(B))}$.
faculty(A)	$\text{:- has_professor(A), has_ph(A), link_to(B,A), has_faculti(B)}$.

Two sample rules are reported in Table 4.5. For example, a Web page is classified as a ‘student’ page if it does not contain the terms *data* and *comment* and it is linked by another page that contains the terms *jame* and *paul* but not the term *mail*. The method was later refined by characterizing documents and links in a statistical way using the Naive Bayes model in combination with FOIL Craven and Slattery (2001).

Graphical models such as Bayesian networks that are traditionally conceived for propositional data (i.e. each instance of case is a tuple of attributes) have been more recently generalized to deal with relational data as well. Relational Bayesian networks have been introduced in Jaeger (1997) and learning algorithms for probabilistic relational models are presented in Friedman *et al.* (1999). Taskar *et al.* (2002) propose a probabilistic relational model for classifying entities, such as Web documents, that have known relationships. This leads to the notion of *collective classification*, where classification of a set of documents is viewed as a global classification problem rather than classifying each page individually. Taskar *et al.* (2002) use a general probabilistic Markov network model for this class of problems and demonstrate that taking relational information (such as hyperlinks) into account clearly improves classification accuracy. Cohn and Hofmann (2001) describe a probabilistic model that jointly accounts for contents and connectivity integrating ideas from PLSA (see Section 4.5.2) and PHITS (see Section 5.6.2).

Classification and probabilistic modeling of relational data in general is still a relatively less explored research area at this time but is clearly a promising methodology for classification and modeling of Web documents.

4.8 Document Clustering

4.8.1 Background and examples

As discussed in Chapter 1, clustering is the process of finding natural groups in data where there are no class labels present, i.e. all of the training data are unsupervised. In document clustering the objects to be clustered are typically represented using a bag-of-words representation. Document clustering is useful across a variety of applications related to the Web. We may for example want to automatically group Web pages on a website into clusters based on their content for the purposes of automatically generating links in real time to different clusters. If we have a relatively small number of pages, it may be possible to manually assign pages to clusters. However, many

websites have tens of thousands of pages. In these cases it is clearly impractical to manually label all of the Web pages and automated clustering is an attractive approach.

Another application of clustering of Web documents is the automated grouping of the results of search engine queries. Search engines typically return a ranked list of URLs relative to a specified query. For example, issuing the query ‘World Cup’ to the Google search engine on 2 December 2002 yielded the following results.

- The top ranked URL was <http://www.fifaworldcup.com/> and indeed the next 10 ranked URLs, as well as many of the top 100, were also about soccer. Clearly ‘soccer’ is the largest cluster of URLs for this query.
- However, the 12th ranked URL was <http://www.dubaiworldcup.com>, a Web page about ‘the richest horse race in the world’ and obviously a different topic from soccer. As such it could be assigned to a different cluster.
- Further down the list we find items such as <http://www.wcsk8.com/> for skateboarding, <http://www.cricketworldcup.com/> for cricket, <http://www.pwca.org/> for paragliding, <http://www.rugby2003.com.au/> for rugby, and so on. Each of these could reasonably be expected to be in different small clusters.
- Robot soccer also appears on the list, <http://www.robocup.org/>, and could perhaps be a subcluster of the main soccer cluster.
- Skiing appears multiple times, both downhill skiing with <http://www.fis-ski.com/> and <http://www.skiworldcup.org/> and cross-country skiing with <http://www.uwasa.fi/~hkn/WC-Skiing/>. Again perhaps these could be viewed as two subclusters of a cluster about skiing.

Naturally it would be quite useful to automatically group these URLs in real time into meaningful clusters. In fact some search engines already support this feature, one example being <http://www.vivisimo.com/>. For the same query of ‘World Cup’ on 2 December 2002, the Vivisimo engine returned 175 URLs with the following cluster titles and number of URLs in each cluster in parentheses: FIFA World Cup (44); Soccer (42); Sports (24); History (19); Rugby World Cup (13); Women’s World Cup (10); Betting (8); Cricket (6); Skiing World Cup (3); and so on. The details of this particular algorithm are not published but it can cluster about 200 URLs in response to a query in ‘human real time’ and automatically assign labels to the clusters (such as those assigned to the World Cup clusters above).

4.8.2 Clustering algorithms for documents

There are a variety of classic clustering algorithms that can be applied to document clustering including hierarchical clustering, K -means type clustering, or probabilistic clustering (see the earlier discussion in Chapter 1 on clustering). In what follows we provide a brief discussion of applying these algorithms to document clustering.

Hierarchical clustering

Hierarchical clustering algorithms do not presume a fixed number of clusters K in advance but instead produce a binary tree where each node is a subcluster of the parent node. Assume that we are clustering n objects. The root node consists of a ‘cluster’ containing all objects and the n leaf nodes correspond to ‘clusters’ where each contains one of the n original objects. This tree structure is known as a dendrogram.

Agglomerative hierarchical clustering algorithms start with a pairwise matrix of distances or similarities between all pairs of objects. At the start of the algorithm all objects are considered to be in their own cluster. The algorithm operates by iteratively and greedily merging the two closest clusters into a new cluster. The resulting merging process results in the gradual bottom-up construction of a dendrogram. The definition of ‘closest’ depends on how we define distance between two sets (clusters) of objects in terms of their individual distances. For example, if we define closest as being the smallest pairwise distance between any pair of objects (where the first is in the first cluster, and the second is in the second cluster) we will tend to get clusters where some objects can be relatively far away from each other. This leads to the well-known ‘chaining effect’ if, for example, the distances correspond to Euclidean distance in some d -dimensional space, where the cluster shapes in d -space can become quite elongated and ‘chain-like.’ This minimum-distance algorithm is also known as single-link clustering.

In contrast we could define closest as being the maximum distance between all pairs of objects. This leads naturally to very compact clusters if viewed in a Euclidean space, since we are ensuring at each step that the maximum distance between any pair of objects in the cluster is minimized. This maximum-distance method is known as complete-link clustering. Other definitions for ‘closest’ are possible, such as computing averages of pair-wise distances, providing algorithms that can be thought of as between the single-line and complete-link methods.

A useful feature of hierarchical clustering for documents is that we can build domain-specific knowledge into the definition of the pairwise distance measure between objects. For example, the cosine distance in Equation (4.1) or weighting schemes such as TF-IDF (Equation (4.5)) could be used to define distance measures. Alternatively, for HTML documents where we believe that some documents are structurally similar to others, we could define an edit distance to reflect structural differences.

However, a significant disadvantage of hierarchical clustering is that the agglomerative algorithms have a time complexity between $O(n^2)$ and $O(n^3)$ depending on the particular algorithm being implemented. All agglomerative algorithms are at least $O(n^2)$ due the requirement of starting with a pairwise distance matrix between all n objects. For small values of n , such as the clustering of a few hundred Web pages that are returned by a search engine, an $O(n^2)$ algorithm may be feasible. However, for large values of n , such as 1000 or more, hierarchical clustering is somewhat impractical from a computational viewpoint.

Probabilistic model-based clustering

An alternative approach to hierarchical clustering is that of probabilistic model-based clustering. Earlier in Chapter 1 we briefly reviewed clustering using the naive Bayes model, and earlier in this chapter we discussed using the same model for classification. Recall from Chapter 1 that model-based clustering assumes the existence of a generative probabilistic model for the data, typically in the form of a mixture model with K components, where each component corresponds to a probability distribution model for one of the clusters. The problem of clustering in this context amounts to one of learning the parameters of this mixture model, specifically, the parameters of each component model and the mixture weights for each component.

As we saw with classification earlier in this chapter, the naive Bayes model is attractive as a component model for document clustering, since it contains only one parameter per dimension and the dimensionality of our document vectors is likely be high, e.g. as high as 5000–50 000 if we are using the bag-of-words representation. Even with a model such as the naive Bayes that is linear in the number of parameters, these dimensionalities are still rather high. In practice a number of heuristics can be used to reduce dimensionality, such as removing very common words as well as very rare words. We cannot use standard class-based feature selection criteria directly (as we did for classification earlier in this chapter), since in clustering we do not know *a priori* what the classes (clusters) are.

4.8.3 Related approaches

Tantrum *et al.* (2002) discuss various strategies for combining ideas from hierarchical clustering and model-based clustering with applications to sets of documents. In this work the original bag-of-words representation is reduced to a 50-dimensional space by applying latent semantic indexing before clustering, and then performing clustering in the reduced-dimensional space. Dhillon and Modha (2001) investigate a spherical K -means algorithm for clustering of sparse high-dimensional document vectors and again use ideas from linear algebra to reduce dimensionality improve the quality of clustering. The motivation in both of these approaches is to combine dimension reduction with clustering, with the promise of being able to achieve more reliable clustering in the lower-dimensional space. A caveat here is that dimension-reduction techniques could in fact destroy the very cluster structure that we are seeking, unless the objective function for dimension reduction somehow incorporates the notion of what constitutes a good clustering of the data.

From a computational viewpoint, Dhillon *et al.* (2001) describe a number of practical methods for efficient implementation of the spherical K -means algorithm for document clustering. McCallum *et al.* (2000b) propose the use of approximate distance measures that are cheap to compute and that can divide a data set into subsets called ‘canopies.’ The general idea is that only objects within the same canopy need to be considered when updating clusters associated with that canopy and this is particularly efficient when both the dimensionality of the problem and the number of objects

are both very large. The authors describe speed-ups of up to an order of magnitude for greedy agglomerative clustering, K -means clustering, and probabilistic model-based clustering, using their approach.

A number of other representations (besides hierarchical clustering, K -means, or mixtures models) have also been proposed for clustering documents. Taskar *et al.* (2002) describe how their Markov network model for relational data can be used to incorporate both text on the page as well as hyperlink structure for clustering of Web pages. Slonim and Tishby (2000) propose a very general information-theoretic technique for clustering called the *information bottleneck*. The technique appears to work particularly well for document clustering (Slonim *et al.* 2002).

Zamir and Etzioni (1998) describe a specific algorithm for the problem discussed earlier of clustering the results of search engines that was . They use ‘snippets’ returned by Web search engines as the basis for clustering and propose a clustering algorithm that uses suffix tree data structures based on phrases between algorithms. In the data sets used in their experiments they show that the resulting algorithm is both computationally efficient (linear in the number of documents) and finds clusters that are approximately as good as those obtained from clustering the full text of the Web pages.

4.9 Information Extraction

An interesting application of machine-learning and artificial intelligence techniques to text documents in general (and to text data on the Web in particular) is that of information extraction (Cohen *et al.* 2000). The general idea is to automatically extract unstructured text data from Web pages and to represent this extracted information in some well-defined schema (e.g. in a form suitable to enter into a relational database table). For example, we might want to extract information on authors and books from various online bookstore and publisher pages. Furthermore we might want to search for online reviews of these books, in newspaper and magazine articles, at online stores, etc. In another type of application a company might want to continuously crawl the Web searching for information about certain technologies or products of interest.

In general the problem of information extraction can often be characterized as that of detecting information about ‘entities’ (individuals, organizations, objects, etc.) in Web pages and then performing some form of ‘parsing’ to extract the relevant information such as the name of the entity, attributes of the entity, and relationships among multiple entities. This is a rich research topic – here we only provide the briefest of overviews but hopefully this will whet the reader’s interest in exploring some of the references below and learning more about this topic. Our overview is loosely based on Cohen and McCallum (2002).

Early efforts in information extraction relied on hand-built models, for example the FASTUS system (Appelt *et al.* 1995) that uses ‘cascades’ of finite-state machines to parse text sequences into lexical units, entity names, groups of words associated with verbs, and so forth. More recent work, including, for example, the Web-KB system

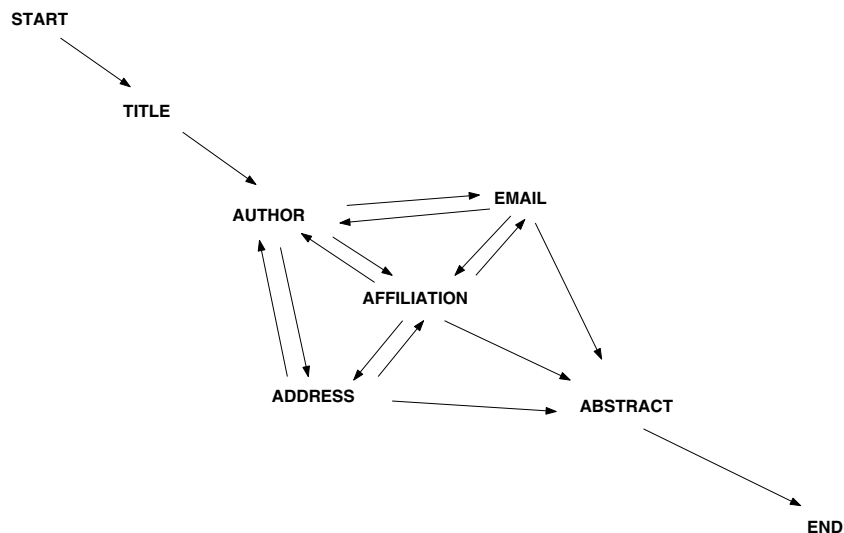


Figure 4.12 A toy example of states and transitions in an HMM for extracting various fields from the beginning of research papers. Not shown are various possible self-transition probabilities (e.g. multiple occurrences of the state ‘author’), transition probabilities on the edges, and the probability distributions of words associated with each state. See Figure 7 in McCallum *et al.* (2000b) for a more detailed and realistic example of an HMM for this problem.

described earlier in this chapter, has focused largely on the use of machine-learning and statistical techniques that leverage human-labelled data to automate the process of constructing information extractors. In effect these systems use the human-labelled text to learn how to extract the relevant information (Cardie 1997; Kushmerick *et al.* 1997).

One general approach is to define information extraction as a classification problem. For example, we might want to classify short segments of text in terms of whether they correspond to the title, author names, author addresses and affiliations, and so forth, from research papers that are found by crawling the Web (for example this is one component of the functionality in the CiteSeer digital library system (Lawrence *et al.* 1999)). A classification approach to this problem would be to represent each document as a sequence of words and then use a ‘sliding window’ of fixed width k as input to a classifier – each of the k inputs to the classifier is a word in a specific position. The system can be trained on positive and negative examples that are typically manually labeled. A variety of different classifiers such as naive Bayes, decision trees, and relational rule representations have been used for sliding-window based classification (Baluja *et al.* 2000; Califf and Mooney 1998; Freitag 1998; Soderland 1999).

A limitation of the sliding window is that it does not take into account sequential constraints that are naturally present in the data, e.g. the fact that the ‘author’ field almost always precedes the ‘address’ field in the header for a research paper. To take this type of structure into account, one approach is to train stochastic finite-

state machines that can incorporate sequential dependence. One popular choice has been hidden Markov models (HMMs), which were mentioned briefly in Chapter 1 during our discussion of graphical models. An HMM contains a finite-state Markov model, where each state in the model corresponds to one of the fields that we wish to extract from the text. For example, in parsing headers of research papers we could have states corresponding to the title of the paper, author name, etc. An example of a simple Markov state diagram is shown in Figure 4.12. The key idea in HMMs is that the true Markov state sequence is unknown at parse-time – instead we see noisy observations from each state, in this case the sequence of words from the document. Each state has a characteristic probability distribution over the set of all possible words, e.g. the distribution of words from the state ‘title’ will be quite different from the distribution of words for the state ‘author names.’ Thus, given a sequence of words, and an HMM, we can ‘parse’ the observed sequence into a corresponding set of inferred states – the Viterbi algorithm provides an efficient method (linear in the number of observed words) for producing the most likely state-sequence, given the observations.

For information extraction, the HMM can be trained in a supervised manner with manually labeled data, or bootstrapped using a combination of labeled and unlabeled data (where the EM algorithm is used for training HMMs on unlabeled data). An early application of this approach to the problem of named entity extraction (automatically finding names of people, organizations, and places in free text) is described in Bikel *et al.* (1997). A variety of related ideas and extensions are described in Leek (1997), Freitag and McCallum (2000), McCallum *et al.* (2000a) and Lafferty *et al.* (2001). For a detailed description of the use of machine-learning methods (and HMMs in particular) to automatically create a large-scale digital library see McCallum *et al.* (2000c).

Information extraction is a broad and growing area of research at the intersection of Web research, language modeling, machine learning, information retrieval, and database theory. In this section we have barely scratched the surface of the many research problems, techniques, and applications that have been proposed, but nonetheless, we hope that the reader has gained a general high-level understanding of some of the basic concepts involved.

4.10 Exercises

Exercise 4.1. Given a collection of m documents to be indexed, compare the memory required to store pointers as integer document identifiers and as the difference between consecutive document identifiers using Elias’s γ coding. (Hint: use Zipf’s Law for term frequency and note that the most frequent term occurs in n documents, the second most frequent in $n/2$ documents, the third in $n/4$ documents and so on.)

Exercise 4.2. A vector $\mathbf{x} \in \mathbb{R}^m$ is said to be *sparse* if

$$\ell = |\{j = 1, \dots, m : x_j \neq 0\}| \ll m.$$

The dot product of two vectors \mathbf{x} and \mathbf{x}' normally requires $\Omega(m)$ time. In the case of sparse vectors, this can be reduced to $\Omega(\ell + \ell')$. Describe an efficient algorithm for computing the dot product of sparse vectors. (Hint: use linear lists to store the vectors in memory.)

Exercise 4.3. Consider the vector-space representation of documents and compare the cosine distance (Equation (4.1)) to the ordinary Euclidean distance. Show that for vectors of unit length the ranking induced by the two distances is the same.

Exercise 4.4. Show that a data set of n vectors in \mathbb{R}^m is linearly separable if $n \leq m + 1$.

Exercise 4.5. Consider a data set of n text documents. Suppose L is the average number of words in each document. How many documents must be collected, on average, before the sufficient condition of Exercise 4.4 fails? (Hint: use Heap's law to estimate vocabulary growth.)

Exercise 4.6. Collect subjects of your email mailbox and build a term–document matrix X using a relatively small set of frequent terms. Compute the SVD decomposition of the matrix using a software package capable of linear algebra computations (e.g. octave) and examine the resulting reconstructed matrix \hat{X} . Try querying your messages and compare the results obtained with a simple Boolean model where you simply retrieve matching subjects and the LSI model. Experiment with different values of k (the dimension of the latent semantic space).

Exercise 4.7. Build your own version of the Naive Bayes classifier and evaluate its performance on the Reuters-21578 data set. In a first setting, keep all those classes having at least one document in the training set and one in the test set. Repeat the experiment using only the top 10 most frequent categories. Beware that categories in the collection are not mutually exclusive (some documents belong to several categories).

Exercise 4.8. Write a program that selects the most informative features using mutual information. Test your program on the 20 Newsgroups corpus. Train your Naive Bayes classifier using the most k informative terms for different values of k and plot your generalization accuracy. What is your best value of k ?

Repeat the exercise by using Zipf's Law on text, removing words that are either too frequent or too rare. Try different cutoff values for the k_1 most frequent words and the k_2 most rare words. Compare your results to those obtained with mutual information.

Finally combine the two methods above and compare your results.

Exercise 4.9. Obtain a public domain implementation of Support Vector Machines (e.g. Joachims's SVM^{light}, <http://svmlight.joachims.org/>) and set up an experiment to classify documents in the WEB \rightarrow KB corpus. Since SVMs are binary classifiers you need to devise some coding strategy to tackle the multiclass problem. Compare results obtained using the two easiest strategies: one-against-all and all-pairs.