6 *Scoring, term weighting and the vector space model*

Thus far we have dealt with indexes that support Boolean queries: a document either matches or does not match a query. In the case of large document collections, the resulting number of matching documents can far exceed the number a human user could possibly sift through. Accordingly, it is essential for a search engine to rank-order the documents matching a query. To do this, the search engine computes, for each matching document, a score with respect to the query at hand. In this chapter we initiate the study of assigning a score to a (query, document) pair. This chapter consists of three main ideas.

1. We introduce parametric and zone indexes in Section 6.1, which serve two purposes. First, they allow us to index and retrieve documents by metadata such as the language in which a document is written. Second, they give us a simple means for scoring (and thereby ranking) documents in response to a query.

2. Next, in Section 6.2 we develop the idea of weighting the importance of a term in a document, based on the statistics of occurrence of the term.

3. In Section 6.3 we show that by viewing each document as a vector of such weights, we can compute a score between a query and each document. This view is known as vector space scoring.

Section 6.4 develops several variants of term-weighting for the vector space model. Chapter 7 develops computational aspects of vector space scoring, and related topics.

As we develop these ideas, the notion of a query will assume multiple nuances. In Section 6.1 we consider queries in which specific query terms occur in specified regions of a matching document. Beginning Section 6.2 we will in fact relax the requirement of matching specific regions of a document; instead, we will look at so-called free text queries that simply consist of query terms with no specification on their relative order, importance or where in a document they should be found. The bulk of our study of scoring will be in this latter notion of a query being such a set of terms.

Preliminary draft (c) 2008 Cambridge UP

6.2 Term frequency and weighting

Thus far, scoring has hinged on whether or not a query term is present in a zone within a document. We take the next logical step: a document or zone that mentions a query term more often has more to do with that query and therefore should receive a higher score. To motivate this, we recall the notion of a free text query introduced in Section 1.4: a query in which the terms of the query are typed freeform into the search interface, without any connecting search operators (such as Boolean operators). This query style, which is extremely popular on the web, views the query as simply a set of words. A plausible scoring mechanism then is to compute a score that is the sum, over the query terms, of the match scores between each query term and the document.

Towards this end, we assign to each term in a document a *weight* for that term, that depends on the number of occurrences of the term in the document. We would like to compute a score between a query term t and a document d, based on the weight of t in d. The simplest approach is to assign the weight to be equal to the number of occurrences of term t in document d. This weighting scheme is referred to as *term frequency* and is denoted $\text{tf}_{t,d}$, with the subscripts denoting the term and the document in order.

TERM FREQUENCY

For a document d, the set of weights determined by the tf weights above (or indeed any weighting function that maps the number of occurrences of t in d to a positive real value) may be viewed as a quantitative digest of that document. In this view of a document, known in the literature as the *bag of words model*, the exact ordering of the terms in a document is ignored but the number of occurrences of each term is material (in contrast to Boolean retrieval). We only retain information on the number of occurrences of each term. Thus, the document "Mary is quicker than John" is, in this view, identical to the document "John is quicker than Mary". Nevertheless, it seems intuitive that two documents with similar bag of words representations are similar in content. We will develop this intuition further in Section 6.3.

BAG OF WORDS

Before doing so we first study the question: are all words in a document equally important? Clearly not; in Section 2.2.2 (page 27) we looked at the idea of *stop words* – words that we decide not to index at all, and therefore do not contribute in any way to retrieval and scoring.

6.2.1 Inverse document frequency

Raw term frequency as above suffers from a critical problem: all terms are considered equally important when it comes to assessing relevancy on a query. In fact certain terms have little or no discriminating power in determining relevance. For instance, a collection of documents on the auto industry is likely to have the term auto in almost every document. To this

Preliminary draft (c) 2008 Cambridge UP

Word	cf	df
try	10422	8760
insurance	10440	3997

► **Figure 6.7** Collection frequency (cf) and document frequency (df) behave differently, as in this example from the Reuters collection.

end, we introduce a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination. An immediate idea is to scale down the term weights of terms with high *collection frequency*, defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the tf weight of a term by a factor that grows with its collection frequency.

DOCUMENT FREQUENCY

Instead, it is more commonplace to use for this purpose the *document frequency* df_t, defined to be the number of documents in the collection that contain a term t. This is because in trying to discriminate between documents for the purpose of scoring it is better to use a document-level statistic (such as the number of documents containing a term) than to use a collection-wide statistic for the term. The reason to prefer df to cf is illustrated in Figure 6.7, where a simple example shows that collection frequency (cf) and document frequency (df) can behave rather differently. In particular, the cf values for both try and insurance are roughly equal, but their df values differ significantly. Intuitively, we want the few documents that contain insurance to get a higher boost for a query on insurance than the many documents containing try get from a query on try.

How is the document frequency df of a term used to scale its weight? Denoting as usual the total number of documents in a collection by N, we define the *inverse document frequency* (idf) of a term t as follows:

INVERSE DOCUMENT FREQUENCY

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}. \tag{6.7}$$

Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low. Figure 6.8 gives an example of idf's in the Reuters collection of 806,791 documents; in this example logarithms are to the base 10. In fact, as we will see in Exercise 6.12, the precise base of the logarithm is not material to ranking. We will give on page 227 a justification of the particular form in Equation (6.7).

6.2.2 Tf-idf weighting

We now combine the definitions of term frequency and inverse document frequency, to produce a composite weight for each term in each document.

Preliminary draft (c) 2008 Cambridge UP

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

► **Figure 6.8** Example of idf values. Here we give the idf's of terms with various frequencies in the Reuters collection of 806,791 documents.

TF-IDF

The *tf-idf* weighting scheme assigns to term t a weight in document d given by

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t. \tag{6.8}$$

In other words, $\text{tf-idf}_{t,d}$ assigns to term t a weight in document d that is

1. highest when t occurs many times within a small number of documents (thus lending high discriminating power to those documents);
2. lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
3. lowest when the term occurs in virtually all documents.

DOCUMENT VECTOR

At this point, we may view each document as a *vector* with one component corresponding to each term in the dictionary, together with a weight for each component that is given by (6.8). For dictionary terms that do not occur in a document, this weight is zero. This vector form will prove to be crucial to scoring and ranking; we will develop these ideas in Section 6.3. As a first step, we introduce the *overlap score measure*: the score of a document d is the sum, over all query terms, of the number of times each of the query terms occurs in d. We can refine this idea so that we add up not the number of occurrences of each query term t in d, but instead the tf-idf weight of each term in d.

$$\text{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}. \tag{6.9}$$

In Section 6.3 we will develop a more rigorous form of Equation (6.9).

?

Exercise 6.8

Why is the idf of a term always finite?

Exercise 6.9

What is the idf of a term that occurs in every document? Compare this with the use of stop word lists.

Preliminary draft (c) 2008 Cambridge UP

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

► **Figure 6.9** Table of tf values for Exercise 6.10.

Exercise 6.10

Consider the table of term frequencies for 3 documents denoted Doc1, Doc2, Doc3 in Figure 6.9. Compute the tf-idf weights for the terms car, auto, insurance, best, for each document, using the idf values from Figure 6.8.

Exercise 6.11

Can the tf-idf weight of a term in a document exceed 1?

Exercise 6.12

How does the base of the logarithm in (6.7) affect the score calculation in (6.9)? How does the base of the logarithm affect the relative scores of two documents on a given query?

Exercise 6.13

If the logarithm in (6.7) is computed base 2, suggest a simple approximation to the idf of a term.

6.3 The vector space model for scoring

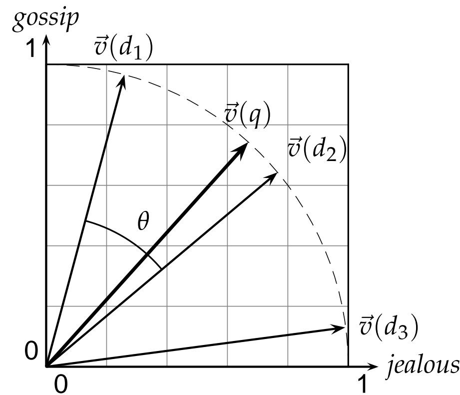
In Section 6.2 (page 117) we developed the notion of a document vector that captures the relative importance of the terms in a document. The representation of a set of documents as vectors in a common vector space is known as the *vector space model* and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering. We first develop the basic ideas underlying vector space scoring; a pivotal step in this development is the view (Section 6.3.2) of queries as vectors in the same vector space as the document collection.

VECTOR SPACE MODEL

6.3.1 Dot products

We denote by $\vec{V}(d)$ the vector derived from document d, with one component in the vector for each dictionary term. Unless otherwise specified, the reader may assume that the components are computed using the tf-idf weighting scheme, although the particular weighting scheme is immaterial to the discussion that follows. The set of documents in a collection then may be viewed as a set of vectors in a vector space, in which there is one axis for

Preliminary draft (c) 2008 Cambridge UP



► **Figure 6.10** Cosine similarity illustrated. $\text{sim}(d_1, d_2) = \cos\theta$.

each term. This representation loses the relative ordering of the terms in each document; recall our example from Section 6.2 (page 117), where we pointed out that the documents *Mary is quicker than John* and *John is quicker than Mary* are identical in such a *bag of words* representation.

How do we quantify the similarity between two documents in this vector space? A first attempt might consider the magnitude of the vector difference between two document vectors. This measure suffers from a drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other. Thus the relative distributions of terms may be identical in the two documents, but the absolute term frequencies of one may be far larger.

To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the *cosine similarity* of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

COSINE SIMILARITY

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \tag{6.10}$$

where the numerator represents the *dot product* (also known as the *inner product*) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their *Euclidean lengths*. The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$. Let $\vec{V}(d)$ denote the document vector for d, with M components $\vec{V}_1(d) \ldots \vec{V}_M(d)$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

DOT PRODUCT

EUCLIDEAN LENGTH

The effect of the denominator of Equation (6.10) is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and

LENGTH-NORMALIZATION

Preliminary draft (c) 2008 Cambridge UP

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

► **Figure 6.11** Euclidean normalized tf values for documents in Figure 6.9.

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

► **Figure 6.12** Term frequencies in three novels. The novels are Austen's *Sense and Sensibility*, *Pride and Prejudice* and Brontë's *Wuthering Heights*.

$\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$. We can then rewrite (6.10) as

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \quad (6.11)$$

Example 6.2: Consider the documents in Figure 6.9. We now apply Euclidean normalization to the tf values from the table, for each of the three documents in the table. The quantity $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$ has the values 30.56, 46.84 and 41.30 respectively for Doc1, Doc2 and Doc3. The resulting Euclidean normalized tf values for these documents are shown in Figure 6.11.

Thus, (6.11) can be viewed as the dot product of the normalized versions of the two document vectors. This measure is the cosine of the angle θ between the two vectors, shown in Figure 6.10. What use is the similarity measure $\text{sim}(d_1, d_2)$? Given a document d (potentially one of the d_i in the collection), consider searching for the documents in the collection most similar to d. Such a search is useful in a system where a user may identify a document and seek others like it – a feature available in the results lists of search engines as a *more like this* feature. We reduce the problem of finding the document(s) most similar to d to that of finding the d_i with the highest dot products (sim values) $\vec{v}(d) \cdot \vec{v}(d_i)$. We could do this by computing the dot products between $\vec{v}(d)$ and each of $\vec{v}(d_1), \ldots, \vec{v}(d_N)$, then picking off the highest resulting sim values.

Example 6.3: Figure 6.12 shows the number of occurrences of three terms (affection, jealous and gossip) in each of the following three novels: Jane Austen's *Sense and Sensibility* (SaS) and *Pride and Prejudice* (PaP) and Emily Brontë's *Wuthering Heights* (WH).

Preliminary draft (c) 2008 Cambridge UP

term	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

► **Figure 6.13** Term vectors for the three novels of Figure 6.12. These are based on raw term frequency only and are normalized as if these were the only terms in the collection. (Since affection and jealous occur in all three documents, their tf-idf weight would be 0 in most formulations.)

Of course, there are many other terms occurring in each of these novels. In this example we represent each of these novels as a unit vector in three dimensions, corresponding to these three terms (only); we use raw term frequencies here, with no idf multiplier. The resulting weights are as shown in Figure 6.13.

Now consider the cosine similarities between pairs of the resulting three-dimensional vectors. A simple computation shows that sim($\vec{v}$(SAS), $\vec{v}$(PAP)) is 0.999, whereas sim($\vec{v}$(SAS), $\vec{v}$(WH)) is 0.888; thus, the two books authored by Austen (SaS and PaP) are considerably closer to each other than to Brontë's *Wuthering Heights*. In fact, the similarity between the first two is almost perfect (when restricted to the three terms we consider). Here we have considered tf weights, but we could of course use other term weight functions.

TERM-DOCUMENT MATRIX

Viewing a collection of N documents as a collection of vectors leads to a natural view of a collection as a *term-document matrix*: this is an $M \times N$ matrix whose rows represent the M terms (dimensions) of the N columns, each of which corresponds to a document. As always, the terms being indexed could be stemmed before indexing; for instance, jealous and jealousy would under stemming be considered as a single dimension. This matrix view will prove to be useful in Chapter 18.

6.3.2 Queries as vectors

There is a far more compelling reason to represent documents as vectors: we can also view a *query* as a vector. Consider the query $q =$ jealous gossip. This query turns into the unit vector $\vec{v}(q) = (0, 0.707, 0.707)$ on the three coordinates of Figures 6.12 and 6.13. The key idea now: to assign to each document d a score equal to the dot product

$$\vec{v}(q) \cdot \vec{v}(d).$$

In the example of Figure 6.13, *Wuthering Heights* is the top-scoring document for this query with a score of 0.509, with *Pride and Prejudice* a distant second with a score of 0.085, and *Sense and Sensibility* last with a score of 0.074. This simple example is somewhat misleading: the number of dimen-

Preliminary draft (c) 2008 Cambridge UP

sions in practice will be far larger than three: it will equal the vocabulary size M.

To summarize, by viewing a query as a "bag of words", we are able to treat it as a very short document. As a consequence, we can use the cosine similarity between the query vector and a document vector as a measure of the score of the document for that query. The resulting scores can then be used to select the top-scoring documents for a query. Thus we have

$$\text{score}(q,d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}. \tag{6.12}$$

A document may have a high cosine score for a query even if it does not contain all query terms. Note that the preceding discussion does not hinge on any specific weighting of terms in the document vector, although for the present we may think of them as either tf or tf-idf weights. In fact, a number of weighting schemes are possible for query as well as document vectors, as illustrated in Example 6.4 and developed further in Section 6.4.

Computing the cosine similarities between the query vector and each document vector in the collection, sorting the resulting scores and selecting the top K documents can be expensive — a single similarity computation can entail a dot product in tens of thousands of dimensions, demanding tens of thousands of arithmetic operations. In Section 7.1 we study how to use an inverted index for this purpose, followed by a series of heuristics for improving on this.

Example 6.4: We now consider the query best car insurance on a fictitious collection with N = 1,000,000 documents where the document frequencies of auto, best, car and insurance are respectively 5000, 50000, 10000 and 1000.

term	query				document			product
	tf	df	idf	$w_{t,q}$	tf	wf	$w_{t,d}$	
auto	0	5000	2.3	0	1	1	0.41	0
best	1	50000	1.3	1.3	0	0	0	0
car	1	10000	2.0	2.0	1	1	0.41	0.82
insurance	1	1000	3.0	3.0	2	2	0.82	2.46

In this example the weight of a term in the query is simply the idf (and zero for a term not in the query, such as auto); this is reflected in the column header $w_{t,q}$ (the entry for auto is zero because the query does not contain the termauto). For documents, we use tf weighting with no use of idf but with Euclidean normalization. The former is shown under the column headed wf, while the latter is shown under the column headed $w_{t,d}$. Invoking (6.9) now gives a net score of $0 + 0 + 0.82 + 2.46 = 3.28$.

6.3.3 Computing vector scores

In a typical setting we have a collection of documents each represented by a vector, a free text query represented by a vector, and a positive integer K. We

Preliminary draft (c) 2008 Cambridge UP

```
COSINESCORE(q)
 1  float Scores[N] = 0
 2  Initialize Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5      for each pair(d, tf_{t,d}) in postings list
 6      do Scores[d] += wf_{t,d} × w_{t,q}
 7  Read the array Length[d]
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

► **Figure 6.14** The basic algorithm for computing vector space scores.

seek the K documents of the collection with the highest vector space scores on the given query. We now initiate the study of determining the K documents with the highest vector space scores for a query. Typically, we seek these K top documents in ordered by decreasing score; for instance many search engines use $K = 10$ to retrieve and rank-order the first page of the ten best results. Here we give the basic algorithm for this computation; we develop a fuller treatment of efficient techniques and approximations in Chapter 7.

Figure 6.14 gives the basic algorithm for computing vector space scores. The array Length holds the lengths (normalization factors) for each of the N documents, while the the array Scores holds the scores for each of the documents. When the scores are finally computed in Step 11, all that remains in Step 12 is to pick off the K documents with the highest scores.

The outermost loop beginning Step 3 repeats the updating of Scores, iterating over each query term t in turn. In Step 5 we calculate the weight in the query vector for term t. Steps 6-8 update the score of each document by adding in the contribution from term t. This process of adding in contributions one query term at a time is sometimes known as *term-at-a-time* scoring or accumulation, and the N elements of the array *Scores* are therefore known as *accumulators*. For this purpose, it would appear necessary to store, with each postings entry, the weight $\mathrm{wf}_{t,d}$ of term t in document d (we have thus far used either tf or tf-idf for this weight, but leave open the possibility of other functions to be developed in Section 6.4). In fact this is wasteful, since storing this weight may require a floating point number. Two ideas help alleviate this space problem. First, if we are using inverse document frequency, we need not precompute idf_t; it suffices to store N/df_t at the head of the postings for t. Second, we store the term frequency $\mathrm{tf}_{t,d}$ for each postings entry. Finally, Step 12 extracts the top K scores – this requires a priority queue

TERM-AT-A-TIME

ACCUMULATOR

Preliminary draft (c) 2008 Cambridge UP

data structure, often implemented using a heap. Such a heap takes no more than $2N$ comparisons to construct, following which each of the K top scores can be extracted from the heap at a cost of $O(\log N)$ comparisons.

Note that the general algorithm of Figure 6.14 does not prescribe a specific implementation of how we traverse the postings lists of the various query terms; we may traverse them one term at a time as in the loop beginning at Step 3, or we could in fact traverse them concurrently as in Figure 1.6. In such a concurrent postings traversal we compute the scores of one document at a time, so that it is sometimes called *document-at-a-time* scoring. We will say more about this in Section 7.1.5.

DOCUMENT-AT-A-TIME

?

Exercise 6.14

If we were to stem jealous and jealousy to a common stem before setting up the vector space, detail how the definitions of tf and idf should be modified.

Exercise 6.15

Recall the tf-idf weights computed in Exercise 6.10. Compute the Euclidean normalized document vectors for each of the documents, where each vector has four components, one for each of the four terms.

Exercise 6.16

Verify that the sum of the squares of the components of each of the document vectors in Exercise 6.15 is 1 (to within rounding error). Why is this the case?

Exercise 6.17

With term weights as computed in Exercise 6.15, rank the three documents by computed score for the query car insurance, for each of the following cases of term weighting in the query:

1. The weight of a term is 1 if present in the query, 0 otherwise.
2. Euclidean normalized idf.

6.4 Variant tf-idf functions

For assigning a weight for each term in each document, a number of alternatives to tf and tf-idf have been considered. We discuss some of the principal ones here; a more complete development is deferred to Chapter 11. We will summarize these alternatives in Section 6.4.3 (page 128).

6.4.1 Sublinear tf scaling

It seems unlikely that twenty occurrences of a term in a document truly carry twenty times the significance of a single occurrence. Accordingly, there has been considerable research into variants of term frequency that go beyond counting the number of occurrences of a term. A common modification is

Preliminary draft (c) 2008 Cambridge UP

to use instead the logarithm of the term frequency, which assigns a weight given by

$$\text{wf}_{t,d} = \begin{cases} 1 + \log \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{6.13}$$

In this form, we may replace tf by some other function wf as in (6.13), to obtain:

$$\text{wf-idf}_{t,d} = \text{wf}_{t,d} \times \text{idf}_t. \tag{6.14}$$

Equation (6.9) can then be modified by replacing tf-idf by wf-idf as defined in (6.14).

6.4.2 Maximum tf normalization

One well-studied technique is to normalize the tf weights of all terms occurring in a document by the maximum tf in that document. For each document d, let $\text{tf}_{\max}(d) = \max_{\tau \in d} \text{tf}_{\tau,d}$, where τ ranges over all terms in d. Then, we compute a normalized term frequency for each term t in document d by

$$\text{ntf}_{t,d} = a + (1-a)\frac{\text{tf}_{t,d}}{\text{tf}_{\max}(d)}, \tag{6.15}$$

SMOOTHING

where a is a value between 0 and 1 and is generally set to 0.4, although some early work used the value 0.5. The term a in (6.15) is a *smoothing* term whose role is to damp the contribution of the second term – which may be viewed as a scaling down of tf by the largest tf value in d. We will encounter smoothing further in Chapter 13 when discussing classification; the basic idea is to avoid a large swing in $\text{ntf}_{t,d}$ from modest changes in $\text{tf}_{t,d}$ (say from 1 to 2). The main idea of maximum tf normalization is to mitigate the following anomaly: we observe higher term frequencies in longer documents, merely because longer documents tend to repeat the same words over and over again. To appreciate this, consider the following extreme example: supposed we were to take a document d and create a new document d' by simply appending a copy of d to itself. While d' should be no more relevant to any query than d is, the use of (6.9) would assign it twice as high a score as d. Replacing $\text{tf-idf}_{t,d}$ in (6.9) by $\text{ntf-idf}_{t,d}$ eliminates the anomaly in this example. Maximum tf normalization does suffer from the following issues:

1. The method is unstable in the following sense: a change in the stop word list can dramatically alter term weightings (and therefore ranking). Thus, it is hard to tune.

2. A document may contain an outlier term with an unusually large number of occurrences of that term, not representative of the content of that document.

Preliminary draft (c) 2008 Cambridge UP

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

► **Figure 6.15** SMART notation for tf-idf variants. Here *CharLength* is the number of characters in the document.

3. More generally, a document in which the most frequent term appears roughly as often as many other terms should be treated differently from one with a more skewed distribution.

6.4.3 Document and query weighting schemes

Equation (6.12) is fundamental to information retrieval systems that use any form of vector space scoring. Variations from one vector space scoring method to another hinge on the specific choices of weights in the vectors $\vec{V}(d)$ and $\vec{V}(q)$. Figure 6.15 lists some of the principal weighting schemes in use for each of $\vec{V}(d)$ and $\vec{V}(q)$, together with a mnemonic for representing a specific combination of weights; this system of mnemonics is sometimes called SMART notation, following the authors of an early text retrieval system. The mnemonic for representing a combination of weights takes the form *ddd.qqq* where the first triplet gives the term weighting of the document vector, while the second triplet gives the weighting in the query vector. The first letter in each triplet specifies the term frequency component of the weighting, the second the document frequency component, and the third the form of normalization used. It is quite common to apply different normalization functions to $\vec{V}(d)$ and $\vec{V}(q)$. For example, a very standard weighting scheme is *lnc.ltc*, where the document vector has log-weighted term frequency, no idf (for both effectiveness and efficiency reasons), and cosine normalization, while the query vector uses log-weighted term frequency, idf weighting, and cosine normalization.

Preliminary draft (c) 2008 Cambridge UP

13 *Text classification and Naive Bayes*

Thus far, this book has mainly discussed the process of *ad hoc retrieval* where users have transient information needs, which they try to address by posing one or more queries to a search engine. However, many users have ongoing information needs. For example, you might need to track developments in *multicore computer chips*. One way of doing this is to issue the query multicore AND computer AND chip against an index of recent newswire articles each morning. In this and the following two chapters we examine the question: how can this repetitive task be automated? To this end, many systems support *standing queries*. A standing query is like any other query except that it is periodically executed on a collection to which new documents are incrementally added over time.

STANDING QUERY

If your standing query is just multicore AND computer AND chip, you will tend to miss many relevant new articles which use other terms such as *multicore processors*. To achieve good recall, standing queries thus have to be refined over time and can gradually become quite complex. In this example, using a Boolean search engine with stemming, you might end up with a query like (multicore OR multi-core) AND (chip OR processor OR microprocessor).

To capture the generality and scope of the problem space to which standing queries belong, we now introduce the general notion of a *classification* problem. Given a set of *classes*, we seek to determine which class(es) a given object belongs to. In the example, the standing query serves to divide new newswire articles into the two classes: documents about multicore computer chips and documents not about multicore computer chips. We refer to this as *two-class classification*. Classification using standing queries is also called *routing* or *filtering* and will be discussed further in Section 15.3.1 (page 335).

CLASSIFICATION

ROUTING

FILTERING

A class need not be as narrowly focused as the standing query *multicore computer chips*. Often, a class is a more general subject area like *China* or *coffee*. Such more general classes are usually referred to as *topics*, and the classification task is then called *text classification*, *text categorization*, *topic classification* or *topic spotting*. An example for *China* appears in Figure 13.1. Standing queries and topics differ in their degree of specificity, but the methods for solving

TEXT CLASSIFICATION

Preliminary draft (c) 2008 Cambridge UP

routing, filtering and text classification are essentially the same. We therefore include routing and filtering under the rubric of text classification in this and the following chapters.

The notion of classification is very general and has many applications within and beyond information retrieval. For instance in computer vision, a classifier may be used to divide images into classes such as *landscape*, *portrait* and *neither*. We focus here on examples from information retrieval such as:

- Several of the preprocessing steps necessary for indexing as discussed in Chapter 2: detecting a document's encoding (ASCII, Unicode UTF-8 etc; page 20); word segmentation (Is the whitespace between two letters a word boundary or not? page 25); truecasing (page 30); and identifying the language of a document (page 46)

- The automatic detection of spam pages (which then are not included in the search engine index)

- The automatic detection of sexually explicit content (which is included in search results only if the user turns an option such as SafeSearch off)

SENTIMENT DETECTION

- *Sentiment detection* or the automatic classification of a movie or product review as positive or negative. An example application is a user searching for negative reviews before buying a camera to make sure it has no undesirable features or quality problems.

EMAIL SORTING

- Personal *email sorting*. A user may have folders like *talk announcements*, *electronic bills*, *email from family and friends* etc. and may want a classifier to classify each incoming email and automatically move it to the appropriate folder. It is easier to find messages in sorted folders than in a very large inbox. The most common case of this application is a spam folder that holds all suspected spam messages.

VERTICAL SEARCH ENGINE

- Topic-specific or *vertical* search. *Vertical search engines* restrict searches to a particular topic. For example, the query computer science on a vertical search engine for the topic *China* will return a list of Chinese computer science departments with higher precision and recall than the query computer science China on a general purpose search engine. This is because the vertical search engine does not include web pages in its index that contain the term china in a different sense (e.g., referring to a hard white ceramic), but does include relevant pages even if they don't explicitly mention the term China.

- Finally, the ranking function in ad hoc information retrieval can also be based on a document classifier as we will explain in Section 15.4 (page 341).

Preliminary draft (c) 2008 Cambridge UP

This list shows the general importance of classification in information retrieval. Most retrieval systems today contain multiple components that use some form of classifier. The classification task we will use as an example in this book is text classification.

A computer is not essential for classification. Many classification tasks have traditionally been solved manually. Books in a library are assigned Library of Congress categories by a librarian. But manual classification is expensive to scale. The *multicore computer chips* example illustrates one alternative approach: classification by the use of standing queries – which can be thought of as *rules* – most commonly written by hand. As in our example (multicore OR multi-core) AND (chip OR processor OR microprocessor), rules are sometimes equivalent to Boolean expressions.

RULES!IN TEXT CLASSIFICATION

A rule captures a certain combination of keywords that indicates a class. Hand-coded rules have good scaling properties, but creating and maintaining them over time is labor-intensive. A technically skilled person (e.g., a domain expert who is good at writing regular expressions) can create rule sets that will rival or exceed the accuracy of the automatically generated classifiers we will discuss shortly. But it can be hard to find someone with this specialized skill.

Apart from manual classification and hand-crafted rules, there is a third approach to text classification, namely, machine learning-based text classification. It is the approach that we focus on in the next several chapters. In machine learning, the set of rules or, more generally, the decision criterion of the text classifier is learned automatically from training data. This approach is also called *statistical text classification* if the learning method is statistical. In statistical text classification, we require a number of good example documents (or training documents) for each class. The need for manual classification is not eliminated since the training documents come from a person who has labeled them – where *labeling* refers to the process of annotating each document with its class. But labeling is arguably an easier task than writing rules. Almost anybody can look at a document and decide whether or not it is related to China. Sometimes such labeling is already implicitly part of an existing workflow. For instance, you may go through the news articles returned by a standing query each morning and give relevance feedback (cf. Chapter 9) by moving the relevant articles to a special folder like *multicore-processors*.

STATISTICAL TEXT CLASSIFICATION

LABELING

We begin this chapter with a general introduction to the text classification problem including a formal definition (Section 13.1); we then cover Naive Bayes, a particularly simple and effective classification method (Sections 13.2–13.4). All of the classification algorithms we study represent documents in high-dimensional spaces. To improve the efficiency of these algorithms, it is generally desirable to reduce the dimensionality of these spaces; to this end, a technique known as *feature selection* is commonly applied in text clas-

Preliminary draft (c) 2008 Cambridge UP

sification as discussed in Section 13.5. Section 13.6 covers evaluation of text classification. In the following chapters, Chapters 14 and 15, we look at two other families of classification methods, vector space classifiers and support vector machines.

13.1 The text classification problem

DOCUMENT SPACE
CLASS

In text classification, we are given a description $d \in \mathbb{X}$ of a document, where $\mathbb{X}$ is the *document space*; and a fixed set of *classes* $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$. Classes are also called *categories* or *labels*. Typically, the document space $\mathbb{X}$ is some type of high-dimensional space, and the classes are human-defined for the needs of an application, as in the examples *China* and *multicore computer chips* above. We are given a *training set* $\mathbb{D}$ of labeled documents $\langle d, c \rangle$,where $\langle d, c \rangle \in \mathbb{X} \times \mathbb{C}$. For example:

TRAINING SET

$$\langle d, c \rangle = \langle \text{Beijing joins the World Trade Organization}, \textit{China} \rangle$$

for the one-sentence document *Beijing joins the World Trade Organization* and the class (or label) *China*.

LEARNING METHOD
CLASSIFIER

Using a *learning method* or *learning algorithm*, we then wish to learn a *classifier* or *classification function* γ that maps documents to classes:

$$\gamma : \mathbb{X} \rightarrow \mathbb{C} \tag{13.1}$$

SUPERVISED LEARNING

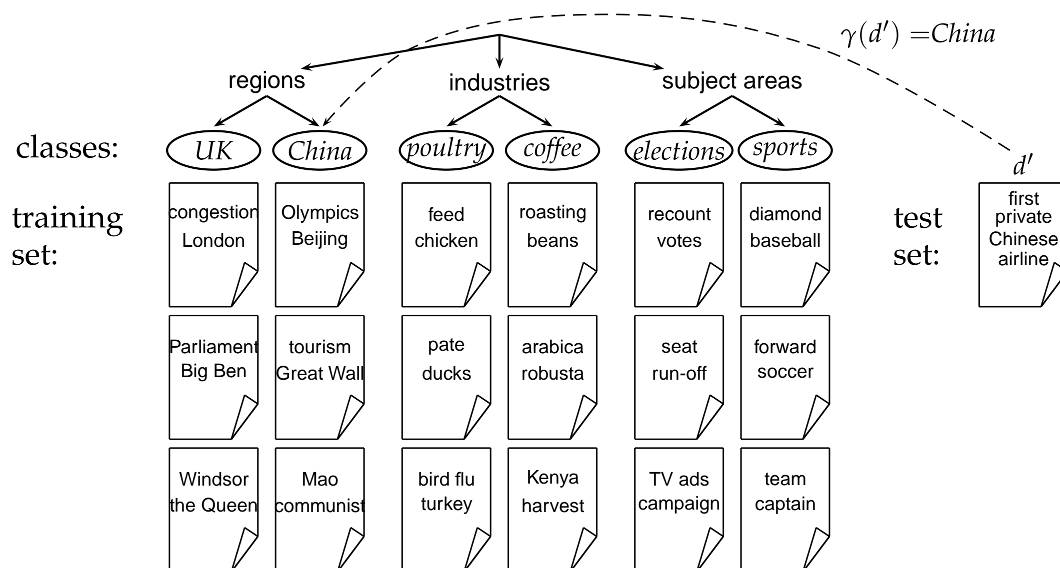
This type of learning is called *supervised learning* since a supervisor (the human who defines the classes and labels training documents) serves as a teacher directing the learning process. We denote the supervised learning method by Γ and write $\Gamma(\mathbb{D}) = \gamma$. The learning method Γ takes the training set $\mathbb{D}$ as input and returns the learned classification function γ.

Most names for learning methods Γ are also used for classifiers γ. We talk about the Naive Bayes *learning method* Γ when we say that "Naive Bayes is robust", meaning that it can be applied to many different learning problems and is unlikely to produce classifiers that fail catastrophically. But when we say that "Naive Bayes had an error rate of 20%", we are describing an experiment in which a particular Naive Bayes *classifier* γ (which was produced by the Naive Bayes learning method) had a 20% error rate in an application.

Figure 13.1 shows an example of text classification from the Reuters-RCV1 collection, introduced in Section 4.2, page 69. There are six classes (*UK*, *China*, ..., *sports*), each with three training documents. We show a few mnemonic words for each document's content. The training set provides some typical examples for each class, so that we can learn the classification function γ. Once we have learned γ, we can apply it to the *test set* (or *test data*), for example the new document *first private Chinese airline* whose class is unknown.

TEST SET

Preliminary draft (c) 2008 Cambridge UP



► **Figure 13.1** Classes, training set and test set in text classification.

In Figure 13.1, the classification function assigns the new document to class $\gamma(d') = China$, which is the correct assignment.

The classes in text classification often have some interesting structure such as the hierarchy in Figure 13.1. There are two instances each of region categories, industry categories and subject area categories. A hierarchy can be an important aid in solving a classification problem; see Section 15.3.2 for further discussion. Until then, we will make the assumption in the text classification chapters that the classes form a set with no subset relationships between them.

Definition (13.1) stipulates that a document is a member of exactly one class. This is not the most appropriate model for the hierarchy in Figure 13.1. For instance, a document about the 2008 Olympics should be a member of two classes: the *China* class and the *sports* class. This type of classification problem is referred to as an *any-of* problem and we will return to it in Section 14.5 (page 306). For the time being, we only consider *one-of* problems where a document is a member of exactly one class.

Our goal in text classification is high accuracy on test data or *new data* – for example, the newswire articles that we will encounter tomorrow morning in the multicore chip example. It is easy to achieve high accuracy on the training

set (e.g., we can simply memorize the labels). However, high accuracy on the training set in general does not mean that the classifier will work well on new data in an application. When we use the training set to learn a classifier for test data, we make the assumption that training data and test data are similar or from *the same distribution*. We defer a precise definition of this notion to Section 14.6 (page 308).

13.2 Naive Bayes text classification

MULTINOMIAL NAIVE BAYES

The first supervised learning method we introduce is the *multinomial Naive Bayes* or *multinomial NB* model, a probabilistic learning method. The probability of a document d being in class c is computed as

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{13.2}$$

where $P(t_k|c)$ is the conditional probability of term t_k occurring in a document of class c. We interpret $P(t_k|c)$ as a measure of how much evidence t_k contributes that c is the correct class. $P(c)$ is the prior probability of a document occurring in class c. If a document's terms do not provide clear evidence for one class vs. another, we choose the one that has a higher prior probability. $\langle t_1, t_2, \ldots, t_{n_d} \rangle$ are the tokens in d that are part of the vocabulary we use for classification and n_d is the number of such tokens in d. For example, $\langle t_1, t_2, \ldots, t_{n_d} \rangle$ for the one-sentence document *Beijing and Taipei join the WTO* might be $\langle$Beijing, Taipei, join, WTO$\rangle$, with $n_d = 4$, if we treat the terms and and the as stop words.

MAXIMUM A POSTERIORI CLASS

In text classification, our goal is to find the best class for the document. The best class in NB classification is the most likely or *maximum a posteriori* (MAP) class c_{map}:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c) \tag{13.3}$$

We write $\hat{P}$ for P since we do not know the true values of the parameters $P(c)$ and $P(t_k|c)$, but estimate them from the training set as we will see in a moment.

In Equation (13.3), many conditional probabilities are multiplied, one for each position $1 \leq k \leq n_d$. This can result in a floating point underflow. It is therefore better to perform the computation by adding logarithms of probabilities instead of multiplying probabilities. The class with the highest log probability score is still the most probable since $\log(xy) = \log(x) + \log(y)$ and the logarithm function is monotonic. Hence, the maximization that is

Preliminary draft (c) 2008 Cambridge UP

actually done in most implementations of Naive Bayes is:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \, [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)] \tag{13.4}$$

Equation (13.4) has a simple interpretation. Each conditional parameter $\log \hat{P}(t_k|c)$ is a weight that indicates how good an indicator t_k is for c. Similarly, the prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c. More frequent classes are more likely to be the correct class than infrequent classes. The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class, and Equation (13.4) selects the class for which we have the most evidence.

We will initially work with this intuitive interpretation of the multinomial NB model and defer a formal derivation to Section 13.4.

How do we estimate the parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$? We first try the maximum likelihood estimate (MLE, Section 11.3.2, page 226), which is simply the relative frequency and corresponds to the most likely value of each parameter given the training data. For the priors this estimate is:

$$\hat{P}(c) = \frac{N_c}{N} \tag{13.5}$$

where N_c is the number of documents in class c and N is the total number of documents.

We estimate the conditional probability $\hat{P}(t|c)$ as the relative frequency of term t in documents belonging to class c:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \tag{13.6}$$

where T_{ct} is the number of occurrences of t in training documents from class c, including multiple occurrences of a term in a document. We have made the *positional independence assumption* here, which we will discuss in more detail in the next section: T_{ct} is a count of occurrences in all positions k in the documents in the training set. Thus, we do not compute different estimates for different positions and, for example, if a word occurs twice in a document, in positions k_1 and k_2, then $\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c)$.

The problem with the MLE estimate is that it is zero for a term-class combination that did not occur in the training data. If the term WTO in the training data only occurred in *China* documents, then the MLE estimates for the other classes, for example *UK*, will be zero:

$$\hat{P}(\text{WTO}|UK) = 0$$

Now the one-sentence document *Britain is a member of the WTO* will get a conditional probability of zero for *UK* since we are multiplying the conditional

Preliminary draft (c) 2008 Cambridge UP

	docID	words in document	in c = *China*?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

► **Table 13.1** Data for parameter estimation examples.

probabilities for all terms in Equation (13.2). Clearly, the model should assign a high probability to the *UK* class since the term Britain occurs. The problem is that the zero probability for WTO cannot be "conditioned away," no matter how strong the evidence for the class *UK* from other features. The estimate is 0 because of *sparseness*: The training data are never large enough to represent the frequency of rare events adequately, for example, the frequency of WTO occurring in *UK* documents.

SPARSENESS

ADD-ONE SMOOTHING

To eliminate zeros, we use *add-one* or *Laplace* smoothing, which simply adds one to each count (cf. Section 11.3.2):

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B} \tag{13.7}$$

where $B = |V|$ is the number of terms in the vocabulary. Add-one smoothing can be interpreted as a uniform prior (each term occurs once for each class) that is then updated as evidence from the training data comes in. Note that this is a prior probability for the occurrence of a *term* as opposed to the prior probability of a *class* which we estimate in Equation (13.5) on the document level.

We have now introduced all the elements we need for training and applying an NB classifier. The complete algorithm is described in Figure 13.2.

Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}
\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms. We then get:

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

Preliminary draft (c) 2008 Cambridge UP

```
TRAINMULTINOMIALNB(ℂ, 𝔻)
 1  V ← EXTRACTVOCABULARY(𝔻)
 2  N ← COUNTDOCS(𝔻)
 3  for each c ∈ ℂ
 4  do N_c ← COUNTDOCSINCLASS(𝔻, c)
 5      prior[c] ← N_c/N
 6      text_c ← CONCATENATETEXTOFALLDOCSINCLASS(𝔻, c)
 7      for each t ∈ V
 8      do T_ct ← COUNTTOKENSOFTERM(text_c, t)
 9      for each t ∈ V
10      do condprob[t][c] ← (T_ct + 1) / Σ_t'(T_ct' + 1)
11  return V, prior, condprob

APPLYMULTINOMIALNB(ℂ, V, prior, condprob, d)
 1  W ← EXTRACTTOKENSFROMDOC(V, d)
 2  for each c ∈ ℂ
 3  do score[c] ← log prior[c]
 4     for each t ∈ W
 5     do score[c] += log condprob[t][c]
 6  return arg max_{c∈ℂ} score[c]
```

► **Figure 13.2** Naive Bayes algorithm (multinomial model): Training and testing.

mode	time complexity
training	$\Theta(\lvert\mathbb{D}\rvert L_{\text{ave}} + \lvert\mathbb{C}\rvert\lvert V\rvert)$
testing	$\Theta(L_{\text{a}} + \lvert\mathbb{C}\rvert M_{\text{a}}) = \Theta(\lvert\mathbb{C}\rvert M_{\text{a}})$

► **Table 13.2** Training and test times for Naive Bayes.

$$\hat{P}(\overline{c}|d_5) \quad \propto \quad 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to c = *China*. The reason for this classification decision is that the three occurrences of the positive indicator Chinese in d_5 outweigh the occurrences of the two negative indicators Japan and Tokyo.

What is the time complexity of Naive Bayes? The complexity of computing the parameters is $\Theta(|\mathbb{C}||V|)$ since the set of parameters consists of $|\mathbb{C}||V|$ conditional probabilities and $|\mathbb{C}|$ priors. The preprocessing necessary for computing the parameters (extracting the vocabulary, counting terms etc.) can be done in one pass through the training data. The time complexity of this component is therefore $\Theta(|\mathbb{D}|L_{\text{ave}})$ where $|\mathbb{D}|$ is the number of documents and L_{ave} is the average length of a document.

Preliminary draft (c) 2008 Cambridge UP

We use $\Theta(|\mathbb{D}|L_{\text{ave}})$ as a notation for $\Theta(T)$ here where T is the length of the training collection. This is non-standard since $\Theta(.)$ is not defined for an average. We prefer expressing the time complexity in terms of $\mathbb{D}$ and L_{ave} because these are the primary statistics used to characterize training collections.

The time complexity of APPLYMULTINOMIALNB in Figure 13.2 is $\Theta(|\mathbb{C}|L_{\text{a}})$. L_{a} and M_{a} are the numbers of tokens and types, respectively, in the test document. APPLYMULTINOMIALNB can be modified to be $\Theta(L_{\text{a}} + |\mathbb{C}|M_{\text{a}})$ (Exercise 13.8). Finally, assuming that the length of test documents is bounded, $\Theta(L_{\text{a}} + |\mathbb{C}|M_{\text{a}}) = \Theta(|\mathbb{C}|M_{\text{a}})$ because $L_{\text{a}} < b|C|M_{\text{a}}$ for a fixed constant b.

Table 13.2 summarizes the time complexities. In general, we have $|\mathbb{C}||V| < |\mathbb{D}|L_{\text{ave}}$, so both training and testing complexity is linear in the time it takes to scan the data. Since we have to look at the data at least once, Naive Bayes can be said to have optimal time complexity. Its efficiency is one reason why Naive Bayes is a popular text classification method.

13.2.1 Relation to multinomial unigram language model

The multinomial NB model is formally identical to the multinomial unigram language model (Section 12.2.1, page 242). In particular, Equation (13.2) is a special case of Equation (12.12) from page 243, which we repeat here for $\lambda = 1$:

$$P(d|q) \propto P(d) \prod_{t \in q} P(t|M_d) \tag{13.8}$$

The document d in text classification (Equation (13.2)) takes the role of the query in language modeling (Equation (13.8)) and the classes c in text classification take the role of the documents d in language modeling. We used Equation (13.8) to rank documents according to the probability that they are relevant to the query q. In NB classification, we are usually only interested in the top-ranked class.

We also used MLE estimates in Section 12.2.2 (page 243) and encountered the problem of zero estimates due to sparse data (page 244); but instead of add-one smoothing, we used a mixture of two distributions to address the problem there. Add-one smoothing is closely related to add-$\frac{1}{2}$ smoothing in Section 11.3.4 (page 228).

Exercise 13.1

Why is $|\mathbb{C}||V| < |\mathbb{D}|L_{\text{ave}}$ in Table 13.2 expected to hold for most text collections?

13.3 The Bernoulli model

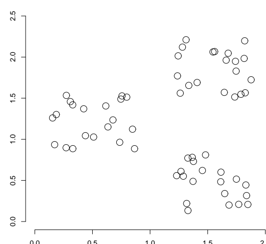
There are two different ways we can set up an NB classifier. The model we introduced in the previous section is the multinomial model. It generates one

Preliminary draft (c) 2008 Cambridge UP

16 *Flat clustering*

CLUSTER

Clustering algorithms group a set of documents into subsets or *clusters*. The algorithms' goal is to create clusters that are coherent internally, but clearly different from each other. In other words, documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters.



► **Figure 16.1** An example of a data set with a clear cluster structure.

UNSUPERVISED LEARNING

Clustering is the most common form of *unsupervised learning*. No supervision means that there is no human expert who has assigned documents to classes. In clustering, it is the distribution and makeup of the data that will determine cluster membership. A simple example is Figure 16.1. It is visually clear that there are three distinct clusters of points. This chapter and Chapter 17 introduce algorithms that find such clusters in an unsupervised fashion.

The difference between clustering and classification may not seem great at first. After all, in both cases we have a partition of a set of documents into groups. But as we will see the two problems are fundamentally different. Classification is a form of supervised learning (Chapter 13, page 256): our goal is to replicate a categorical distinction that a human supervisor im-

Preliminary draft (c) 2008 Cambridge UP

poses on the data. In unsupervised learning, of which clustering is the most important example, we have no such teacher to guide us.

The key input to a clustering algorithm is the distance measure. In Figure 16.1, the distance measure is distance in the 2D plane. This measure suggests three different clusters in the figure. In document clustering, the distance measure is often also Euclidean distance. Different distance measures give rise to different clusterings. Thus, the distance measure is an important means by which we can influence the outcome of clustering.

FLAT CLUSTERING

Flat clustering creates a flat set of clusters without any explicit structure that would relate clusters to each other. *Hierarchical clustering* creates a hierarchy of clusters and will be covered in Chapter 17. Chapter 17 also addresses the difficult problem of labeling clusters automatically.

HARD CLUSTERING
SOFT CLUSTERING

A second important distinction can be made between hard and soft clustering algorithms. *Hard clustering* computes a *hard assignment* – each document is a member of exactly one cluster. The assignment of *soft clustering algorithms* is *soft* – a document's assignment is a distribution over all clusters. In a soft assignment, a document has fractional membership in several clusters. Latent semantic indexing, a form of dimensionality reduction, is a soft clustering algorithm (Chapter 18, page 417).

This chapter motivates the use of clustering in information retrieval by introducing a number of applications (Section 16.1), defines the problem we are trying to solve in clustering (Section 16.2) and discusses measures for evaluating cluster quality (Section 16.3). It then describes two flat clustering algorithms, K-means (Section 16.4), a hard clustering algorithm, and the Expectation-Maximization (or EM) algorithm (Section 16.5), a soft clustering algorithm. K-means is perhaps the most widely used flat clustering algorithm due to its simplicity and efficiency. The EM algorithm is a generalization of K-means and can be applied to a large variety of document representations and distributions.

16.1 Clustering in information retrieval

CLUSTER HYPOTHESIS

The *cluster hypothesis* states the fundamental assumption we make when using clustering in information retrieval.

> **Cluster hypothesis.** Documents in the same cluster behave similarly with respect to relevance to information needs.

The hypothesis states that if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant. This is because clustering puts together documents that share many terms. The cluster hypothesis essentially is the contiguity

Preliminary draft (c) 2008 Cambridge UP

Application	What is clustered?	Benefit	Example
Search result clustering	search results	more effective information presentation to user	Figure 16.2
Scatter-Gather	(subsets of) collection	alternative user interface: "search without typing"	Figure 16.3
Collection clustering	collection	effective information presentation for exploratory browsing	McKeown et al. (2002), http://news.google.com
Language modeling	collection	increased precision and/or recall	Liu and Croft (2004)
Cluster-based retrieval	collection	higher efficiency: faster search	Salton (1971a)

► **Table 16.1** Some applications of clustering in information retrieval.

hypothesis in Chapter 14 (page 289). In both cases, we posit that similar documents behave similarly with respect to relevance.

Table 16.1 shows some of the main applications of clustering in information retrieval. They differ in the set of documents that they cluster – search results, collection or subsets of the collection – and the aspect of an information retrieval system they try to improve – user experience, user interface, effectiveness or efficiency of the search system. But they are all based on the basic assumption stated by the cluster hypothesis.

SEARCH RESULT CLUSTERING

The first application mentioned in Table 16.1 is *search result clustering* where by search results we mean the documents that were returned in response to a query. The default presentation of search results in information retrieval is a simple list. Users scan the list from top to bottom until they have found the information they are looking for. Instead, search result clustering clusters the search results, so that similar documents appear together. It is often easier to scan a few coherent groups than many individual documents. This is particularly useful if a search term has different word senses. The example in Figure 16.2 is jaguar. Three frequent senses on the web refer to the car, the animal and an Apple operating system. The *Clustered Results* panel returned by the Vivísimo search engine (http://vivisimo.com) can be a more effective user interface for understanding what is in the search results than a simple list of documents.

SCATTER-GATHER

A better user interface is also the goal of *Scatter-Gather*, the second application in Table 16.1. Scatter-Gather clusters the whole collection to get groups of documents that the user can select or *gather*. The selected groups are merged and the resulting set is again clustered. This process is repeated until a cluster of interest is found. An example is shown in Figure 16.3.

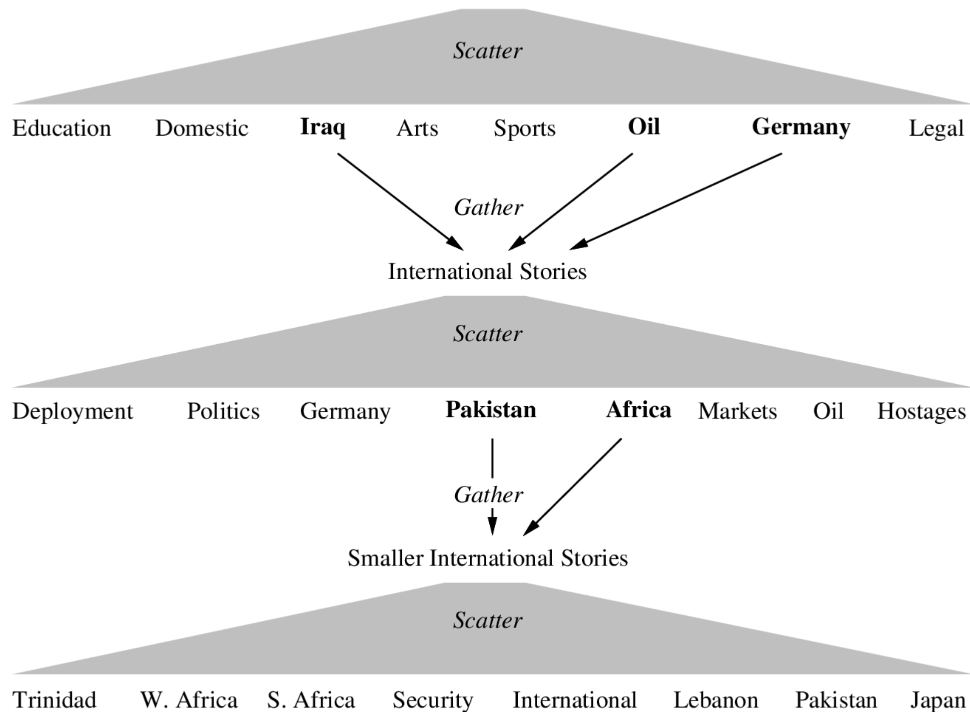
Preliminary draft (c) 2008 Cambridge UP

► **Figure 16.2** Clustering of search results to improve recall. None of the top hits cover the animal sense of jaguar, but users can easily access it by clicking on the *cat* cluster in the *Clustered Results* panel on the left (third arrow from the top).

Automatically generated clusters like those in Figure 16.3 are not as neatly organized as a manually constructed hierarchical tree like the Open Directory at http://dmoz.org. Also, finding descriptive labels for clusters automatically is a difficult problem (Section 17.7, page 396). But cluster-based navigation is an interesting alternative to keyword searching, the standard information retrieval paradigm. This is especially true in scenarios where users prefer browsing over searching because they are unsure about which search terms to use.

As an alternative to the user-mediated iterative clustering in Scatter-Gather, we can also compute a static hierarchical clustering of a collection that is not influenced by user interactions ("Collection clustering" in Table 16.1). Google News and its precursor, the Columbia NewsBlaster system, are examples of this approach. In the case of news, we need to frequently recompute the clustering to make sure that users can access the latest breaking stories. Clustering is well suited for access to a collection of news stories since news reading is not really search, but rather a process of selecting a subset of stories about recent events.

Preliminary draft (c) 2008 Cambridge UP



► **Figure 16.3** An example of a user session in Scatter-Gather. A collection of New York Times news stories is clustered ("scattered") into eight clusters (top row). The user manually *gathers* three of these into a smaller collection *International Stories* and performs another scattering operation. This process repeats until a small cluster with relevant documents is found (e.g., *Trinidad*).

The fourth application of clustering exploits the cluster hypothesis directly for improving search results, based on a clustering of the entire collection. We use a standard inverted index to identify an initial set of documents that match the query, but we then add other documents from the same clusters even if they have low similarity to the query. For example, if the query is car and several car documents are taken from a cluster of automobile documents, then we can add documents from this cluster that use terms other than car (automobile, vehicle etc). This can increase recall since a group of documents with high mutual similarity is often relevant as a whole.

More recently this idea has been used for language modeling. Equation (12.10), page 245, showed that to avoid sparse data problems in the language modeling approach to IR, the model of document d can be interpolated with a

Preliminary draft (c) 2008 Cambridge UP

collection model. But the collection contains many documents with terms untypical of d. By replacing the collection model with a model derived from d's cluster, we get more accurate estimates of the occurrence probabilities of terms in d.

Clustering can also speed up search. As we saw in Section 6.3.2 (page 123) search in the vector space model amounts to finding the nearest neighbors to the query. The inverted index supports fast nearest-neighbor search for the standard IR setting. However, sometimes we may not be able to use an inverted index efficiently, e.g., in latent semantic indexing (Chapter 18). In such cases, we could compute the similarity of the query to every document, but this is slow. The cluster hypothesis offers an alternative: Find the clusters that are closest to the query and only consider documents from these clusters. Within this much smaller set, we can compute similarities exhaustively and rank documents in the usual way. Since there are many fewer clusters than documents, finding the closest cluster is fast; and since the documents matching a query are all similar to each other, they tend to be in the same clusters. While this algorithm is inexact, the expected decrease in search quality is small. This is essentially the application of clustering that was covered in Section 7.1.6 (page 141).

?

Exercise 16.1

Define two documents as similar if they have at least two proper names like Clinton or Sarkozy in common. Give an example of an information need and two documents, for which the cluster hypothesis does *not* hold for this notion of similarity.

Exercise 16.2

Make up a simple one-dimensional example (i.e. points on a line) with two clusters where the inexactness of cluster-based retrieval shows up. In your example, retrieving clusters close to the query should do worse than direct nearest neighbor search.

16.2 Problem statement

We can define the goal in hard flat clustering as follows. Given (i) a set of documents $D = \{d_1, \ldots, d_N\}$, (ii) a desired number of clusters K, and (iii) an *objective function* that evaluates the quality of a clustering, we want to compute an assignment $\gamma : D \rightarrow \{1, \ldots, K\}$ that minimizes (or, in other cases, maximizes) the objective function. In most cases, we also demand that γ is surjective, i.e., that none of the K clusters is empty.

OBJECTIVE FUNCTION

The objective function is often defined in terms of similarity or distance between documents. Below, we will see that the objective in K-means clustering is to minimize the average distance between documents and their centroids or, equivalently, to maximize the similarity between documents and their centroids. The discussion of similarity measures and distance metrics

Preliminary draft (c) 2008 Cambridge UP

in Chapter 14 (page 291) also applies to this chapter. As in Chapter 14, we use both similarity and distance to talk about relatedness between documents.

For documents, the type of similarity we want is usually topic similarity or high values on the same dimensions in the vector space model. For example, documents about China have high values on dimensions like Chinese, Beijing, and Mao whereas documents about the UK tend to have high values for London, Britain and Queen. We approximate topic similarity with cosine similarity or Euclidean distance in vector space (Chapter 6). If we intend to capture similarity of a type other than topic, for example, similarity of language, then a different representation may be appropriate. When computing topic similarity, stop words can be safely ignored, but they are important cues for separating clusters of English (in which the occurs frequently and la infrequently) and French documents (in which the occurs infrequently and la frequently).

PARTITIONAL CLUSTERING

A note on terminology. An alternative definition of hard clustering is that a document can be a full member of more than one cluster. *Partitional clustering* always refers to a clustering where each document belongs to exactly one cluster. (But in a partitional hierarchical clustering (Chapter 17) all members of a cluster are of course also members of its parent.) On the definition of hard clustering that permits multiple membership, the difference between soft clustering and hard clustering is that membership values in hard clustering are either 0 or 1, whereas they can take on any non-negative value in soft clustering.

EXHAUSTIVE

Some researchers distinguish between *exhaustive* clusterings that assign each document to a cluster and non-exhaustive clusterings, in which some documents will be assigned to no cluster. Non-exhaustive clusterings in which each document is a member of either no cluster or one cluster are called *exclusive*. We define clustering to be exhaustive in this book.

EXCLUSIVE

16.2.1 Cardinality – the number of clusters

CARDINALITY

A difficult issue in clustering is determining the number of clusters or *cardinality* of a clustering, which we denote by K. Often K is nothing more than a good guess based on experience or domain knowledge. But for K-means, we will also introduce a heuristic method for choosing K and an attempt to incorporate the selection of K into the objective function. Sometimes the application puts constraints on the range of K. For example, the Scatter-Gather interface in Figure 16.3 could not display more than about $K = 10$ clusters per layer because of the size and resolution of computer monitors in the early 1990s.

Since our goal is to optimize an objective function, clustering is essentially

Preliminary draft (c) 2008 Cambridge UP

a search problem. The brute force solution would be to enumerate all possible clusterings and pick the best. However, there are exponentially many partitions, so this approach is not feasible.[1] For this reason, most flat clustering algorithms refine an initial partitioning iteratively. If the search starts at an unfavorable initial point, we may miss the global optimum. Finding a good starting point is therefore another important problem we have to solve in flat clustering.

16.3 Evaluation of clustering

Typical objective functions in clustering formalize the goal of attaining high intra-cluster similarity (documents within a cluster are similar) and low inter-cluster similarity (documents from different clusters are dissimilar). This is an *internal criterion* for the quality of a clustering. But good scores on an internal criterion do not necessarily translate into good effectiveness in an application. An alternative to internal criteria is direct evaluation in the application of interest. For search result clustering, we may want to measure the time it takes users to find an answer with different clustering algorithms. This is the most direct evaluation, but it is expensive, especially if large user studies are necessary.

INTERNAL CRITERION OF QUALITY

As a surrogate for user judgments, we can use a set of classes in an evaluation benchmark or gold standard (see Section 8.5, page 164, and Section 13.6, page 279). The gold standard is ideally produced by human judges with a good level of inter-judge agreement (see Chapter 8, page 152). We can then compute an *external criterion* that evaluates how well the clustering matches the gold standard classes. For example, we may want to say that the optimal clustering of the search results for jaguar in Figure 16.2 consists of three classes corresponding to the three senses *car*, *animal*, and *operating system*. In this type of evaluation, we only use the partition provided by the gold standard, not the class labels.

EXTERNAL CRITERION OF QUALITY

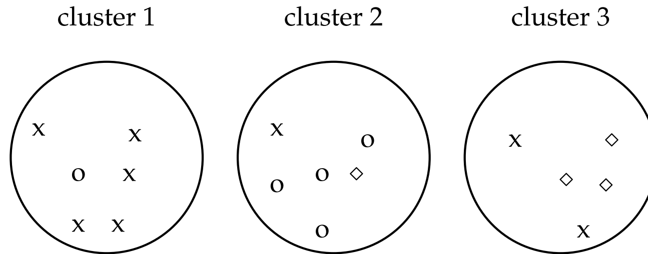
This section introduces four external criteria of clustering quality. *Purity* is a simple and transparent evaluation measure. *Normalized mutual information* can be information-theoretically interpreted. The *Rand index* penalizes both false positive and false negative decisions during clustering. The *F measure* in addition supports differential weighting of these two types of errors.

PURITY

To compute *purity*, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned documents and dividing by N.

1. An upper bound on the number of clusterings is $K^N/K!$. The exact number of different partitions of N documents into K clusters is the Stirling number of the second kind. See http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html or Comtet (1974).

Preliminary draft (c) 2008 Cambridge UP



► **Figure 16.4** Purity as an external evaluation criterion for cluster quality. Majority class and number of members of the majority class for the three clusters are: x, 5 (cluster 1); o, 4 (cluster 2); and ⋄, 3 (cluster 3). Purity is $(1/17) \times (5+4+3) \approx 0.71$.

	purity	NMI	RI	F_5
lower bound	0.0	0.0	0.0	0.0
maximum	1	1	1	1
value for Figure 16.4	0.71	0.36	0.68	0.46

► **Table 16.2** The four external evaluation measures applied to the clustering in Figure 16.4.

Formally:

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j| \tag{16.1}$$

where $\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ is the set of clusters and $\mathbb{C} = \{c_1, c_2, \ldots, c_J\}$ is the set of classes. We interpret ω_k as the set of documents in ω_k and c_j as the set of documents in c_j in Equation (16.1).

We present an example of how to compute purity in Figure 16.4.[2] Bad clusterings have purity values close to 0, a perfect clustering has a purity of 1. Purity is compared with the other three measures discussed in this chapter in Table 16.2.

High purity is easy to achieve when the number of clusters is large – in particular, purity is 1 if each document gets its own cluster. Thus, we cannot use purity to trade off the quality of the clustering against the number of clusters.

NORMALIZED MUTUAL INFORMATION

A measure that allows us to make this tradeoff is *normalized mutual infor-*

2. Recall our note of caution from Figure 14.2 (page 291) when looking at this and other 2D figures in this and the following chapter: these illustrations can be misleading because 2D projections of length-normalized vectors distort similarities and distances between points.

Preliminary draft (c) 2008 Cambridge UP

mation or *NMI*:

$$\text{NMI}(\Omega, \mathbb{C}) = \frac{I(\Omega; \mathbb{C})}{[H(\Omega) + H(\mathbb{C})]/2} \quad (16.2)$$

I is mutual information (cf. Chapter 13, page 272):

$$I(\Omega; \mathbb{C}) = \sum_k \sum_j P(\omega_k \cap c_j) \log \frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j)} \quad (16.3)$$

$$= \sum_k \sum_j \frac{|\omega_k \cap c_j|}{N} \log \frac{N|\omega_k \cap c_j|}{|\omega_k||c_j|} \quad (16.4)$$

where $P(\omega_k)$, $P(c_j)$, and $P(\omega_k \cap c_j)$ are the probabilities of a document being in cluster ω_k, class c_j, and in the intersection of ω_k and c_j, respectively. Equation (16.4) is equivalent to Equation (16.3) for maximum likelihood estimates of the probabilities (i.e., the estimate of each probability is the corresponding relative frequency).

H is entropy as defined in Chapter 5 (page 99):

$$H(\Omega) = -\sum_k P(\omega_k) \log P(\omega_k) \quad (16.5)$$

$$= -\sum_k \frac{|\omega_k|}{N} \log \frac{|\omega_k|}{N} \quad (16.6)$$

where, again, the second equation is based on maximum likelihood estimates of the probabilities.

$I(\Omega; \mathbb{C})$ in Equation (16.3) measures the amount of information by which our knowledge about the classes increases when we are told what the clusters are. The minimum of $I(\Omega; \mathbb{C})$ is 0 if the clustering is random with respect to class membership. In that case, knowing that a document is in a particular cluster does not give us any new information about what its class might be. Maximum mutual information is reached for a clustering Ω_{exact} that perfectly recreates the classes – but also if clusters in Ω_{exact} are further subdivided into smaller clusters (Exercise 16.7). In particular, a clustering with $K = N$ one-document clusters has maximum MI. So MI has the same problem as purity: it does not penalize large cardinalities and thus does not formalize our bias that, other things being equal, fewer clusters are better.

The normalization by the denominator $[H(\Omega) + H(\mathbb{C})]/2$ in Equation (16.2) fixes this problem since entropy tends to increase with the number of clusters. For example, $H(\Omega)$ reaches its maximum $\log N$ for $K = N$, which ensures that NMI is low for $K = N$. Because NMI is normalized, we can use it to compare clusterings with different numbers of clusters. The particular form of the denominator is chosen because $[H(\Omega) + H(\mathbb{C})]/2$ is a tight upper bound on $I(\Omega; \mathbb{C})$ (Exercise 16.8). Thus, NMI is always a number between 0 and 1.

Preliminary draft (c) 2008 Cambridge UP

An alternative to this information-theoretic interpretation of clustering is to view it as a series of decisions, one for each of the $N(N-1)/2$ pairs of documents in the collection. We want to assign two documents to the same cluster if and only if they are similar. A true positive (TP) decision assigns two similar documents to the same cluster, a true negative (TN) decision assigns two dissimilar documents to different clusters. There are two types of errors we can commit. A false positive (FP) decision assigns two dissimilar documents to the same cluster. A false negative (FN) decision assigns two similar documents to different clusters. The *Rand index* (RI) measures the percentage of decisions that are correct. That is, it is simply accuracy (Section 8.3, page 155).

RAND INDEX
RI

$$\text{RI} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

As an example, we compute RI for Figure 16.4. We first compute TP + FP. The three clusters contain 6, 6, and 5 points, respectively, so the total number of "positives" or pairs of documents that are in the same cluster is:

$$\text{TP} + \text{FP} = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the $\diamond$ pairs in cluster 3, and the x pair in cluster 3 are true positives:

$$\text{TP} = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

Thus, FP = 40 − 20 = 20.

FN and TN are computed similarly, resulting in the following contingency table:

	Same cluster	Different clusters
Same class	TP = 20	FN = 24
Different classes	FP = 20	TN = 72

RI is then $(20+72)/(20+20+24+72) \approx 0.68$.

The Rand index gives equal weight to false positives and false negatives. Separating similar documents is sometimes worse than putting pairs of dissimilar documents in the same cluster. We can use the *F measure* (Section 8.3, page 154) to penalize false negatives more strongly than false positives by selecting a value $\beta > 1$, thus giving more weight to recall.

F MEASURE

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad R = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Preliminary draft (c) 2008 Cambridge UP

Based on the numbers in the contingency table, $P = 20/40 = 0.5$ and $R = 20/44 \approx 0.455$. This gives us $F_1 \approx 0.48$ for $\beta = 1$ and $F_5 \approx 0.456$ for $\beta = 5$. In information retrieval, evaluating clustering with F has the advantage that the measure is already familiar to the research community.

?

Exercise 16.3

Replace every point d in Figure 16.4 with two identical copies of d in the same class. (i) Is it less difficult, equally difficult or more difficult to cluster this set of 34 points as opposed to the 17 points in Figure 16.4? (ii) Compute purity, NMI, RI, and F_5 for the clustering with 34 points. Which measures increase and which stay the same after doubling the number of points? (iii) Given your assessment in (i) and the results in (ii), which measures are best suited to compare the quality of the two clusterings?

16.4 *K*-means

CENTROID

K-means is the most important flat clustering algorithm. Its objective is to minimize the average squared Euclidean distance (Chapter 6, page 131) of documents from their cluster centers where a cluster center is defined as the mean or *centroid* $\vec{\mu}$ of the documents in a cluster ω:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

The definition assumes that documents are represented as length-normalized vectors in a real-valued space in the familiar way. We used centroids for Rocchio classification in Chapter 14 (page 292). They play a similar role here. The ideal cluster in K-means is a sphere with the centroid as its center of gravity. Ideally, the clusters should not overlap. Our desiderata for classes in Rocchio classification were the same. The difference is that we have no labeled training set in clustering for which we know which documents should be in the same cluster.

RESIDUAL SUM OF SQUARES

A measure of how well the centroids represent the members of their clusters is the *residual sum of squares* or *RSS*, the squared distance of each vector from its centroid summed over all vectors:

$$\text{RSS}_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

$$\text{RSS} = \sum_{k=1}^{K} \text{RSS}_k \tag{16.7}$$

RSS is the objective function in K-means and our goal is to minimize it. Since N is fixed, minimizing RSS is equivalent to minimizing the average squared distance, a measure of how well centroids represent their documents.

Preliminary draft (c) 2008 Cambridge UP

```
K-MEANS({x⃗_1,...,x⃗_N}, K)
 1  (s⃗_1, s⃗_2, ..., s⃗_K) ← SELECTRANDOMSEEDS({x⃗_1,...,x⃗_N}, K)
 2  for k ← 1 to K
 3  do μ⃗_k ← s⃗_k
 4  while stopping criterion has not been met
 5  do for k ← 1 to K
 6       do ω_k ← {}
 7       for n ← 1 to N
 8       do j ← arg min_j' |μ⃗_j' − x⃗_n|
 9          ω_j ← ω_j ∪ {x⃗_n}  (reassignment of vectors)
10       for k ← 1 to K
11       do μ⃗_k ← (1/|ω_k|) Σ_{x⃗∈ω_k} x⃗  (recomputation of centroids)
12  return {μ⃗_1,...,μ⃗_K}
```

► **Figure 16.5** The K-means algorithm. For most IR applications, the vectors $\vec{x}_n \in \mathbb{R}^M$ should be length-normalized. Alternative methods of seed selection and initialization are discussed on page 364.

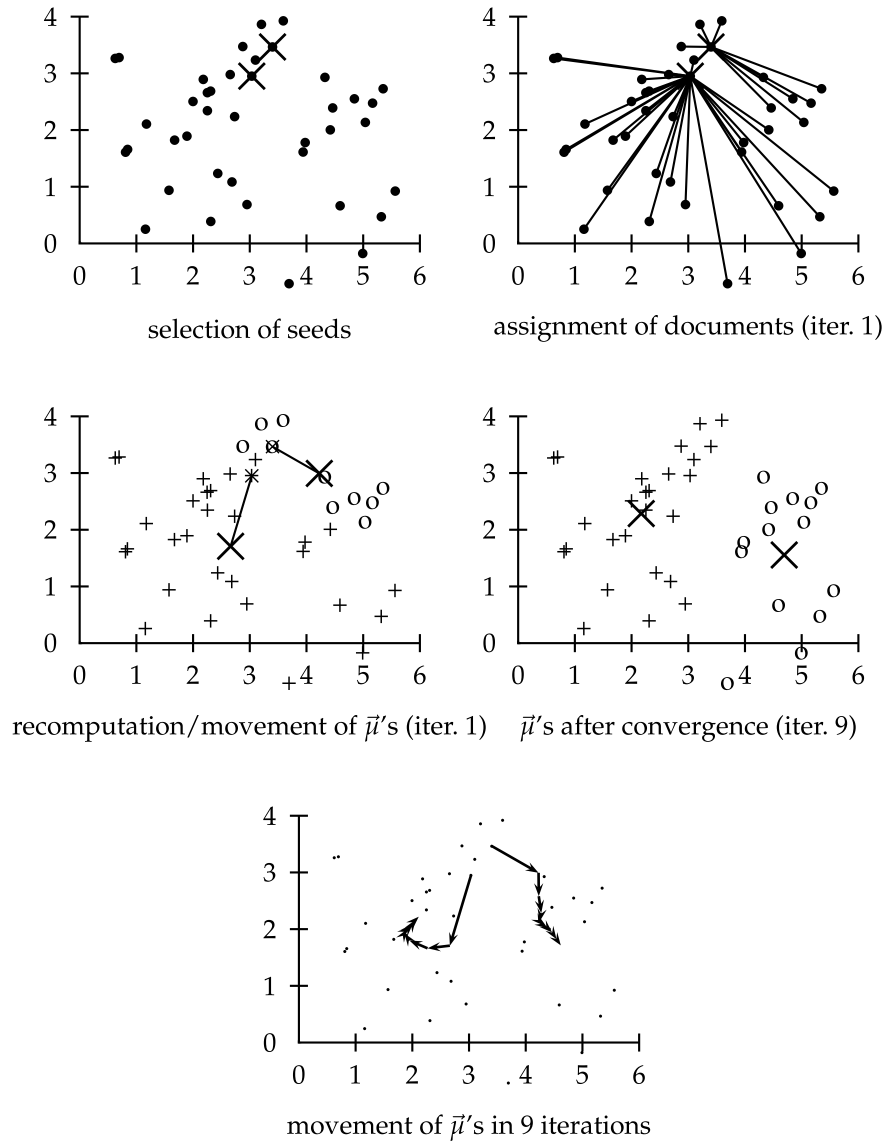
SEED

The first step of K-means is to select as initial cluster centers K randomly selected documents, the *seeds*. The algorithm then moves the cluster centers around in space in order to minimize RSS. As shown in Figure 16.5, this is done iteratively by repeating two steps until a stopping criterion is met: reassigning documents to the cluster with the closest centroid; and recomputing each centroid based on the current members of its cluster. Figure 16.6 shows snapshots from nine iterations of the K-means algorithm for a set of points. The "centroid" column of Table 17.2 (page 397) shows examples of centroids.

We can apply one of the following termination conditions.

- A fixed number of iterations I has been completed. This condition limits the runtime of the clustering algorithm, but in some cases the quality of the clustering will be poor because of an insufficient number of iterations.
- Assignment of documents to clusters (the partitioning function γ) does not change between iterations. Except for cases with a bad local minimum, this produces a good clustering, but runtime may be unacceptably long.
- Centroids $\vec{\mu}_k$ do not change between iterations. This is equivalent to γ not changing (Exercise 16.5).
- Terminate when RSS falls below a threshold. This criterion ensures that the clustering is of a desired quality after termination. In practice, we

Preliminary draft (c) 2008 Cambridge UP



► **Figure 16.6** A K-means example for $K = 2$ in $\mathbb{R}^2$. The position of the two centroids ($\vec{\mu}$'s shown as X's in the top four panels) converges after nine iterations.

Preliminary draft (c) 2008 Cambridge UP

need to combine it with a bound on the number of iterations to guarantee termination.

- Terminate when the decrease in RSS falls below a threshold θ. For small θ, this indicates that we are close to convergence. Again, we need to combine it with a bound on the number of iterations to prevent very long runtimes.

We now show that K-means converges by proving that RSS monotonically decreases in each iteration. We will use *decrease* in the meaning *decrease or does not change* in this section. First, RSS decreases in the reassignment step since each vector is assigned to the closest centroid, so the distance it contributes to RSS decreases. Second, it decreases in the recomputation step because the new centroid is the vector $\vec{v}$ for which RSS_k reaches its minimum.

$$\text{RSS}_k(\vec{v}) = \sum_{\vec{x}\in\omega_k} |\vec{v}-\vec{x}|^2 = \sum_{\vec{x}\in\omega_k}\sum_{m=1}^{M}(v_m - x_m)^2 \tag{16.8}$$

$$\frac{\partial \text{RSS}_k(\vec{v})}{\partial v_m} = \sum_{\vec{x}\in\omega_k} 2(v_m - x_m) \tag{16.9}$$

where x_m and v_m are the mth components of their respective vectors. Setting the partial derivative to zero, we get:

$$v_m = \frac{1}{|\omega_k|}\sum_{\vec{x}\in\omega_k} x_m \tag{16.10}$$

which is the componentwise definition of the centroid. Thus, we minimize RSS_k when the old centroid is replaced with the new centroid. RSS, the sum of the RSS_k, must then also decrease during recomputation.

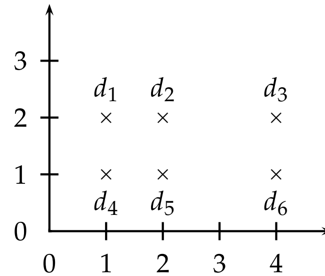
Since there is only a finite set of possible clusterings, a monotonically decreasing algorithm will eventually arrive at a (local) minimum. Take care, however, to break ties consistently, e.g., by assigning a document to the cluster with the lowest index if there are several equidistant centroids. Otherwise, the algorithm can cycle forever in a loop of clusterings that have the same cost.

While this proves the convergence of K-means, there is unfortunately no guarantee that a *global minimum* in the objective function will be reached. This is a particular problem if a document set contains many *outliers*, documents that are far from any other documents and therefore do not fit well into any cluster. Frequently, if an outlier is chosen as an initial seed, then no other vector is assigned to it during subsequent iterations. Thus, we end up with a *singleton cluster* (a cluster with only one document) even though there is probably a clustering with lower RSS. Figure 16.7 shows an example of a suboptimal clustering resulting from a bad choice of initial seeds.

OUTLIER

SINGLETON CLUSTER

Preliminary draft (c) 2008 Cambridge UP



► **Figure 16.7** The outcome of clustering in K-means depends on the initial seeds. For seeds d_2 and d_5, K-means converges to $\{\{d_1, d_2, d_3\}, \{d_4, d_5, d_6\}\}$, a suboptimal clustering. For seeds d_2 and d_3, it converges to $\{\{d_1, d_2, d_4, d_5\}, \{d_3, d_6\}\}$, the global optimum for $K = 2$.

Another type of suboptimal clustering that frequently occurs is one with empty clusters (Exercise 16.11).

Effective heuristics for seed selection include (i) excluding outliers from the seed set; (ii) trying out multiple starting points and choosing the clustering with lowest cost; and (iii) obtaining seeds from another method such as hierarchical clustering. Since deterministic hierarchical clustering methods are more predictable than K-means, a hierarchical clustering of a small random sample of size iK (e.g., for $i = 5$ or $i = 10$) often provides good seeds (see the description of the Buckshot algorithm, Chapter 17, page 399).

Other initialization methods compute seeds that are not selected from the vectors to be clustered. A robust method that works well for a large variety of document distributions is to select i (e.g., $i = 10$) random vectors for each cluster and use their centroid as the seed for this cluster. See Section 16.6 for more sophisticated initializations.

What is the time complexity of K-means? Most of the time is spent on computing vector distances. One such operation costs $\Theta(M)$. The reassignment step computes KN distances, so its overall complexity is $\Theta(KNM)$. In the recomputation step, each vector gets added to a centroid once, so the complexity of this step is $\Theta(NM)$. For a fixed number of iterations I, the overall complexity is therefore $\Theta(IKNM)$. Thus, K-means is linear in all relevant factors: iterations, number of clusters, number of vectors and dimensionality of the space. This means that K-means is more efficient than the hierarchical algorithms in Chapter 17. We had to fix the number of iterations I, which can be tricky in practice. But in most cases, K-means quickly reaches either complete convergence or a clustering that is close to convergence. In the latter case, a few documents would switch membership if further iterations were computed, but this has a small effect on the overall quality of the clustering.

Preliminary draft (c) 2008 Cambridge UP

There is one subtlety in the preceding argument. Even a linear algorithm can be quite slow if one of the arguments of $\Theta(\ldots)$ is large, and M usually is large. High dimensionality is not a problem for computing the distance of two documents. Their vectors are sparse, so that only a small fraction of the theoretically possible M componentwise differences need to be computed. Centroids, however, are dense since they pool all terms that occur in any of the documents of their clusters. As a result, distance computations are time consuming in a naive implementation of K-means. But there are simple and effective heuristics for making centroid-document similarities as fast to compute as document-document similarities. Truncating centroids to the most significant k terms (e.g., $k = 1000$) hardly decreases cluster quality while achieving a significant speedup of the reassignment step (see references in Section 16.6).

K-MEDOIDS

MEDOID

The same efficiency problem is addressed by *K-medoids*, a variant of K-means that computes medoids instead of centroids as cluster centers. We define the *medoid* of a cluster as the document vector that is closest to the centroid. Since medoids are sparse document vectors, distance computations are fast.

16.4.1 Cluster cardinality in K-means

We stated in Section 16.2 that the number of clusters K is an input to most flat clustering algorithms. What do we do if we cannot come up with a plausible guess for K?

A naive approach would be to select the optimal value of K according to the objective function, namely the value of K that minimizes RSS. Defining $\text{RSS}_{\text{min}}(K)$ as the minimal RSS of all clusterings with K clusters, we observe that $\text{RSS}_{\text{min}}(K)$ is a monotonically decreasing function in K (Exercise 16.13), which reaches its minimum 0 for $K = N$ where N is the number of documents. We would end up with each document being in its own cluster. Clearly, this is not an optimal clustering.

A heuristic method that gets around this problem is to estimate $\text{RSS}_{\text{min}}(K)$ as follows. We first perform i (e.g., $i = 10$) clusterings with K clusters (each with a different initialization) and compute the RSS of each. Then we take the minimum of the i RSS values. We denote this minimum by $\widehat{\text{RSS}}_{\text{min}}(K)$. Now we can inspect the values $\widehat{\text{RSS}}_{\text{min}}(K)$ as K increases and find the "knee" in the curve – the point where successive decreases in $\widehat{\text{RSS}}_{\text{min}}$ become noticeably smaller. There are two such points in Figure 16.8, one at $K = 4$, where the gradient flattens slightly, and a clearer flattening at $K = 9$. This is typical: there is seldom a single best number of clusters. We still need to employ an external constraint to choose from a number of possible values of K (4 and 9 in this case).

Preliminary draft (c) 2008 Cambridge UP

21 *Link analysis*

The analysis of hyperlinks and the graph structure of the Web has been instrumental in the development of web search. In this chapter we focus on the use of hyperlinks for ranking web search results. Such link analysis is one of many factors considered by web search engines in computing a composite score for a web page on any given query. We begin by reviewing some basics of the Web as a graph in Section 21.1, then proceed to the technical development of the elements of link analysis for ranking.

Link analysis for web search has intellectual antecedents in the field of citation analysis, aspects of which overlap with an area known as bibliometrics. These disciplines seek to quantify the influence of scholarly articles by analyzing the pattern of citations amongst them. Much as citations represent the conferral of authority from a scholarly article to others, link analysis on the Web treats hyperlinks from a web page to another as a conferral of authority. Clearly, not every citation or hyperlink implies such authority conferral; for this reason, simply measuring the quality of a web page by the number of in-links (citations from other pages) is not robust enough. For instance, one may contrive to set up multiple web pages pointing to a target web page, with the intent of artificially boosting the latter's tally of in-links. This phenomenon is referred to as link spam. Nevertheless, the phenomenon of citation is prevalent and dependable enough that it is feasible for web search engines to derive useful signals for ranking from more sophisticated link analysis. Link analysis also proves to be a useful indicator of what page(s) to crawl next while crawling the web; this is done by using link analysis to guide the priority assignment in the front queues of Chapter 20.

Section 21.1 develops the basic ideas underlying the use of the web graph in link analysis. Sections 21.2 and 21.3 then develop two distinct methods for link analysis, PageRank and HITS.

Preliminary draft (c) 2008 Cambridge UP

21.1 The Web as a graph

Recall the notion of the web graph from Section 19.2.1 and particularly Figure 19.2. Our study of link analysis builds on two intuitions:

1. The anchor text pointing to page B is a good description of page B.

2. The hyperlink from A to B represents an endorsement of page B, by the creator of page A. This is not always the case; for instance, many links amongst pages within a single website stem from the user of a common template. For instance, most corporate websites have a pointer from every page to a page containing a copyright notice – this is clearly not an endorsement. Accordingly, implementations of link analysis algorithms will typical discount such "internal" links.

21.1.1 Anchor text and the web graph

The following fragment of HTML code from a web page shows a hyperlink pointing to the home page of the Journal of the ACM:

```
<a href="http://www.acm.org/jacm/">Journal of the ACM.</a>
```

In this case, the link points to the page `http://www.acm.org/jacm/` and the anchor text is *Journal of the ACM.* Clearly, in this example the anchor is descriptive of the target page. But then the target page (B = `http://www.acm.org/jacm/`) itself contains the same description as well as considerable additional information on the journal. So what use is the anchor text?

The Web is full of instances where the page B does not provide an accurate description of itself. In many cases this is a matter of how the publishers of page B choose to present themselves; this is especially common with corporate web pages, where a web presence is a marketing statement. For example, at the time of the writing of this book the home page of the IBM corporation (`http://www.ibm.com`) did not contain the term computer anywhere in its HTML code, despite the fact that IBM is widely viewed as the world's largest computer maker. Similarly, the HTML code for the home page of Yahoo! (`http://www.yahoo.com`) does not at this time contain the word portal.

Thus, there is often a gap between the terms in a web page, and how web users would describe that web page. Consequently, web searchers need not use the terms in a page to query for it. In addition, many web pages are rich in graphics and images, and/or embed their text in these images; in such cases, the HTML parsing performed when crawling will not extract text that is useful for indexing these pages. The "standard IR" approach to this would be to use the methods outlined in Chapter 9 and Section 12.4. The insight

Preliminary draft (c) 2008 Cambridge UP

behind anchor text is that such methods can be supplanted by anchor text, thereby tapping the power of the community of web page authors.

The fact that the anchors of many hyperlinks pointing to `http://www.ibm.com` include the word computer can be exploited by web search engines. For instance, the anchor text terms can be included as terms under which to index the target web page. Thus, the postings for the term computer would include the document `http://www.ibm.com` and that for the term portal would include the document `http://www.yahoo.com`, using a special indicator to show that these terms occur as anchor (rather than in-page) text. As with in-page terms, anchor text terms are generally weighted based on frequency, with a penalty for terms that occur very often (the most common terms in anchor text across the Web are Click and here, using methods very similar to idf). The actual weighting of terms is determined by machine-learned scoring, as in Section 15.4.1; current web search engines appear to assign a substantial weighting to anchor text terms.

The use of anchor text has some interesting side-effects. Searching for big blue on most web search engines returns the home page of the IBM corporation as the top hit; this is consistent with the popular nickname that many people use to refer to IBM. On the other hand, there have been (and continue to be) many instances where derogatory anchor text such as evil empire leads to somewhat unexpected results on querying for these terms on web search engines. This phenomenon has been exploited in orchestrated campaigns against specific sites. Such orchestrated anchor text may be a form of spamming, since a website can create misleading anchor text pointing to itself, to boost its ranking on selected query terms. Detecting and combating such systematic abuse of anchor text is another form of spam detection that web search engines perform.

The window of text surrounding anchor text (sometimes referred to as *extended anchor text*) is often usable in the same manner as anchor text itself; consider for instance the fragment of web text `there is good discussion of vedic scripture <a>here</a>`. This has been considered in a number of settings and the useful width of this window has been studied; see Section 21.4 for references.

Exercise 21.1

Is it always possible to follow directed edges (hyperlinks) in the web graph from any node (web page) to any other? Why or why not?

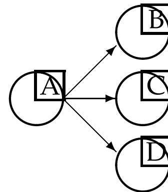
Exercise 21.2

Find an instance of misleading anchor-text on the Web.

Exercise 21.3

Given the collection of anchor-text phrases for a web page x, suggest a heuristic for choosing one term or phrase from this collection that is most descriptive of x.

Preliminary draft (c) 2008 Cambridge UP



► **Figure 21.1** The random surfer at node A proceeds with probability 1/3 to each of B, C and D.

Exercise 21.4

Does your heuristic in the previous exercise take into account a single domain D repeating anchor text for x from multiple pages in D?

21.2 PageRank

We now focus on scoring and ranking measures derived from the link structure alone. Our first technique for link analysis assigns to every node in the web graph a numerical score between 0 and 1, known as its *PageRank*. The PageRank of a node will depend on the link structure of the web graph. Given a query, a web search engine computes a composite score for each web page that combines hundreds of features such as cosine similarity (Section 6.3) and term proximity (Section 7.2.2), together with the PageRank score. This composite score, developed using the methods of Section 15.4.1, is used to provide a ranked list of results for the query.

PAGERANK

Consider a random surfer who begins at a web page (a node of the web graph) and executes a random walk on the Web as follows. At each time step, the surfer proceeds from his current page A to a randomly chosen web page that A hyperlinks to. Figure 21.1 shows the surfer at a node A, out of which there are three hyperlinks to nodes B, C and D; the surfer proceeds at the next time step to one of these three nodes, with equal probabilities 1/3.

As the surfer proceeds in this random walk from node to node, he visits some nodes more often than others; intuitively, these are nodes with many links coming in from other frequently visited nodes. The idea behind PageRank is that pages visited more often in this walk are more important.

What if the current location of the surfer, the node A, has no out-links? To address this we introduce an additional operation for our random surfer: the *teleport* operation. In the teleport operation the surfer jumps from a node to any other node in the web graph. This could happen because he types

TELEPORT

Preliminary draft (c) 2008 Cambridge UP

an address into the URL bar of his browser. The destination of a teleport operation is modeled as being chosen uniformly at random from all web pages. In other words, if N is the total number of nodes in the web graph[1], the teleport operation takes the surfer to each node with probability $1/N$. The surfer would also teleport to his present position with probability $1/N$.

In assigning a PageRank score to each node of the web graph, we use the teleport operation in two ways: (1) When at a node with no out-links, the surfer invokes the teleport operation. (2) At any node that has outgoing links, the surfer invokes the teleport operation with probability $0 < \alpha < 1$ and the standard random walk (follow an out-link chosen uniformly at random as in Figure 21.1) with probability $1 - \alpha$, where α is a fixed parameter chosen in advance. Typically, α might be 0.1.

In Section 21.2.1, we will use the theory of Markov chains to argue that when the surfer follows this combined process (random walk plus teleport) he visits each node v of the web graph a fixed fraction of the time $\pi(v)$ that depends on (1) the structure of the web graph and (2) the value of α. We call this value $\pi(v)$ the PageRank of v and will show how to compute this value in Section 21.2.2.

21.2.1 Markov chains

A Markov chain is a *discrete-time stochastic process:* a process that occurs in a series of time-steps in each of which a random choice is made. A Markov chain consists of N *states*. Each web page will correspond to a state in the Markov chain we will formulate.

A Markov chain is characterized by an $N \times N$ *transition probability matrix* P each of whose entries is in the interval $[0,1]$; the entries in each row of P add up to 1. The Markov chain can be in one of the N states at any given time-step; then, the entry P_{ij} tells us the probability that the state at the next time-step is j, conditioned on the current state being i. Each entry P_{ij} is known as a transition probability and depends only on the current state i; this is known as the Markov property. Thus, by the Markov property,

$$\forall i, j, P_{ij} \in [0,1]$$

and

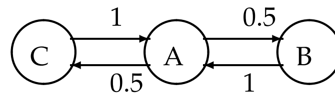
$$\forall i, \sum_{j=1}^{N} P_{ij} = 1. \tag{21.1}$$

STOCHASTIC MATRIX
PRINCIPAL LEFT EIGENVECTOR

A matrix with non-negative entries that satisfies Equation (21.1) is known as a *stochastic matrix*. A key property of a stochastic matrix is that it has a *principal left eigenvector* corresponding to its largest eigenvalue, which is 1.

1. This is consistent with our usage of N for the number of documents in the collection.

Preliminary draft (c) 2008 Cambridge UP



► **Figure 21.2** A simple Markov chain with three states; the numbers on the links indicate the transition probabilities.

In a Markov chain, the probability distribution of next states for a Markov chain depends only on the current state, and not on how the Markov chain arrived at the current state. Figure 21.2 shows a simple Markov chain with three states. From the middle state A, we proceed with (equal) probabilities of 0.5 to either B or C. From either B or C, we proceed with probability 1 to A. The transition probability matrix of this Markov chain is then

$$\begin{pmatrix} 0 & 0.5 & 0.5 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

PROBABILITY VECTOR

A Markov chain's probability distribution over its states may be viewed as a *probability vector*: a vector all of whose entries are in the interval $[0, 1]$, and the entries add up to 1. An N-dimensional probability vector each of whose components corresponds to one of the N states of a Markov chain can be viewed as a probability distribution over its states. For our simple Markov chain of Figure 21.2, the probability vector would have 3 components that sum to 1.

We can view a random surfer on the web graph as a Markov chain, with one state for each web page, and each transition probability representing the probability of moving from one web page to another. The teleport operation contributes to these transition probabilities. The adjacency matrix A of the web graph is defined as follows: if there is a hyperlink from page i to page j, then $A_{ij} = 1$, otherwise $A_{ij} = 0$. We can readily derive the transition probability matrix P for our Markov chain from the $N \times N$ matrix A:

1. If a row of A has no 1's, then replace each element by 1/N. For all other rows proceed as follows.

2. Divide each 1 in A by the number of 1's in its row. Thus, if there is a row with three 1's, then each of them is replaced by 1/3.

3. Multiply the resulting matrix by $1 - \alpha$.

Preliminary draft (c) 2008 Cambridge UP

4. Add α/N to every entry of the resulting matrix, to obtain P.

We can depict the probability distribution of the surfer's position at any time by a probability vector $\vec{x}$. At $t = 0$ the surfer may begin at a state whose corresponding entry in $\vec{x}$ is 1 while all others are zero. By definition, the surfer's distribution at $t = 1$ is given by the probability vector $\vec{x}P$; at $t = 2$ by $(\vec{x}P)P = \vec{x}P^2$, and so on. We will detail this process in Section 21.2.2. We can thus compute the surfer's distribution over the states at any time, given only the initial distribution and the transition probability matrix P.

If a Markov chain is allowed to run for many time steps, each state is visited at a (different) frequency that depends on the structure of the Markov chain. In our running analogy, the surfer visits certain web pages (say, popular news home pages) more often than other pages. We now make this intuition precise, establishing conditions under which such the visit frequency converges to fixed, steady-state quantity. Following this, we set the PageRank of each node v to this steady-state visit frequency and show how it can be computed.

ERGODIC MARKOV CHAIN

Definition: A Markov chain is said to be *ergodic* if there exists a positive integer T_0 such that for all pairs of states i, j in the Markov chain, if it is started at time 0 in state i then for all $t > T_0$, the probability of being in state j at time t is greater than 0.

For a Markov chain to be ergodic, two technical conditions are required of the its states and the non-zero transition probabilities; these conditions are known as *irreducibility* and *aperiodicity*. Informally, the first ensures that there is a sequence of transitions of non-zero probability from any state to any other, while the latter ensures that the states are not partitioned into sets such that all state transitions occur cyclically from one set to another.

STEADY-STATE

Theorem 21.1. *For any ergodic Markov chain, there is a unique* steady-state *probability vector $\vec{\pi}$ that is the principal left eigenvector of P, such that if $\eta(i,t)$ is the number of visits to state i in t steps, then*

$$\lim_{t \to \infty} \frac{\eta(i,t)}{t} = \pi(i),$$

where $\pi(i) > 0$ is the steady-state probability for state i.

It follows from Theorem 21.1 that the random walk with teleporting results in a unique distribution of steady-state probabilities over the states of the induced Markov chain. This steady-state probability for a state is the PageRank of the corresponding web page.

Preliminary draft (c) 2008 Cambridge UP

21.2.2 The PageRank computation

How do we compute PageRank values? Recall the definition of a left eigenvector from Equation 18.2; the left eigenvectors of the transition probability matrix P are N-vectors $\vec{\pi}$ such that

$$\vec{\pi}\, P = \lambda \vec{\pi}. \tag{21.2}$$

The N entries in the principal eigenvector $\vec{\pi}$ are the steady-state probabilities of the random walk with teleporting, and thus the PageRank values for the corresponding web pages. We may interpret Equation (21.2) as follows: if $\vec{\pi}$ is the probability distribution of the surfer across the web pages, he remains in the steady-state distribution $\vec{\pi}$. Given that $\vec{\pi}$ is the steady-state distribution, we have that $\pi P = 1\pi$, so 1 is an eigenvalue of P. Thus if we were to compute the principal left eigenvector of the matrix P — the one with eigenvalue 1 — we would have computed the PageRank values.

There are many algorithms available for computing left eigenvectors; the references at the end of Chapter 18 and the present chapter are a guide to these. We give here a rather elementary method, sometimes known as *power iteration*. If $\vec{x}$ is the initial distribution over the states, then the distribution at time t is $\vec{x}P^t$. As t grows large, we would expect that the distribution $\vec{x}P^t$[2] is very similar to the distribution $\vec{x}P^{t+1}$, since for large t we would expect the Markov chain to attain its steady state. By Theorem 21.1 this is independent of the initial distribution $\vec{x}$. The power iteration method simulates the surfer's walk: begin at a state and run the walk for a large number of steps t, keeping track of the visit frequencies for each of the states. After a large number of steps t, these frequencies "settle down" so that the variation in the computed frequencies is below some predetermined threshold. We declare these tabulated frequencies to be the PageRank values.

We consider the web graph in Exercise 21.6 with $\alpha = 0.5$. The transition probability matrix of the surfer's walk with teleportation is then

$$P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix}. \tag{21.3}$$

Imagine that the surfer starts in state 1, corresponding to the initial probability distribution vector $\vec{x_0} = (1\ 0\ 0)$. Then, after one step the distribution is

$$\vec{x_0}P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} = \vec{x_1}. \tag{21.4}$$

2. Note that P^t represents P raised to the tth power, not the transpose of P which is denoted P^T.

Preliminary draft (c) 2008 Cambridge UP

$\vec{x}_0$	1	0	0
$\vec{x}_1$	1/6	2/3	1/6
$\vec{x}_2$	1/3	1/3	1/3
$\vec{x}_3$	1/4	1/2	1/4
$\vec{x}_4$	7/24	5/12	7/24
...	...	...	...
$\vec{x}$	5/18	4/9	5/18

► **Figure 21.3** The sequence of probability vectors.

After two steps it is

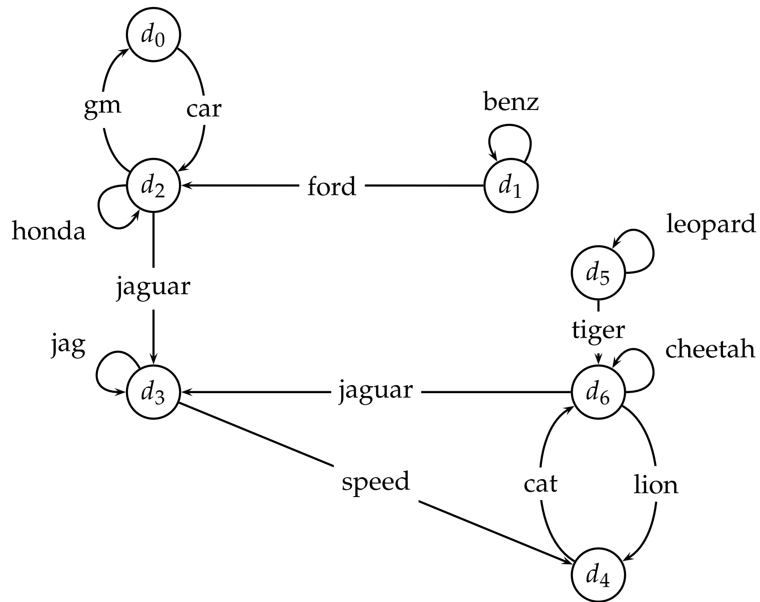
$$\vec{x}_1 P = \begin{pmatrix} 1/6 & 2/3 & 1/6 \end{pmatrix} \begin{pmatrix} 1/6 & 2/3 & 1/6 \\ 5/12 & 1/6 & 5/12 \\ 1/6 & 2/3 & 1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 1/3 & 1/3 \end{pmatrix} = \vec{x}_2. \tag{21.5}$$

Continuing in this fashion gives a sequence of probability vectors as shown in Figure 21.3.

Continuing for several steps, we see that the distribution converges to the steady state of $\vec{x} = (5/18 \quad 4/9 \quad 5/18)$. In this simple example, we may directly calculate this steady-state probability distribution by observing the symmetry of the Markov chain: states 1 and 3 are symmetric, as evident from the fact that the first and third rows of the transition probability matrix in Equation (21.3) are identical. Postulating, then, that they both have the same steady-state probability and denoting this probability by p, we know that the steady-state distribution is of the form $\vec{\pi} = (p \quad 1-2p \quad p)$. Now, using the identity $\vec{\pi} = \vec{\pi}P$, we solve a simple linear equation to obtain $p = 5/18$ and consequently, $\vec{\pi} = (5/18\; 4/9\; 5/18)$.

The PageRank values of pages (and the implicit ordering amongst them) are independent of any query a user might pose; PageRank is thus a query-independent measure of the static quality of each web page (recall such static quality measures from Section 7.1.4). On the other hand, the relative ordering of pages should, intuitively, depend on the query being served. For this reason, search engines use static quality measures such as PageRank as just one of many factors in scoring a web page on a query. Indeed, the relative contribution of PageRank to the overall score may again be determined by machine-learned scoring as in Section 15.4.1.

Preliminary draft (c) 2008 Cambridge UP



► **Figure 21.4** A small web graph. Arcs are annotated with the word that occurs in the anchor text of the corresponding link.

Example 21.1: Consider the graph in Figure 21.4. For a teleportation rate of 0.14 its (stochastic) transition probability matrix is:

0.02	0.02	0.88	0.02	0.02	0.02	0.02
0.02	0.45	0.45	0.02	0.02	0.02	0.02
0.31	0.02	0.31	0.31	0.02	0.02	0.02
0.02	0.02	0.02	0.45	0.45	0.02	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.88
0.02	0.02	0.02	0.02	0.02	0.45	0.45
0.02	0.02	0.02	0.31	0.31	0.02	0.31

The PageRank vector of this matrix is:

$$\vec{x} = (0.05 \quad 0.04 \quad 0.11 \quad 0.25 \quad 0.21 \quad 0.04 \quad 0.31) \tag{21.6}$$

Observe that in Figure 21.4, q_2, q_3, q_4 and q_6 are the nodes with at least two in-links. Of these, q_2 has the lowest PageRank since the random walk tends to drift out of the top part of the graph – the walker can only return there through teleportation.

Preliminary draft (c) 2008 Cambridge UP