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.

TERM FREQUENCY

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.

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

BAG OF WORDS

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.

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

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

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

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.

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.

	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.

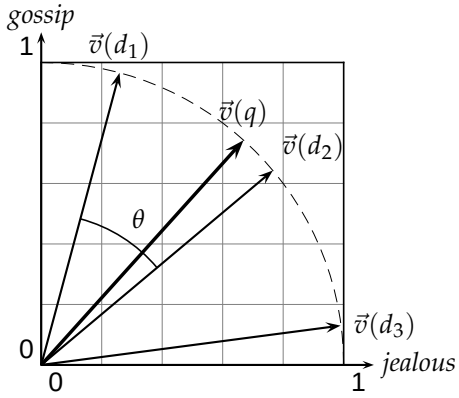
6.3 The vector space model for scoring

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



► **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

	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

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

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).

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-

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

```
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, whereas the array Scores holds the scores for each of the documents. When the scores are finally computed in Step 9, all that remains in Step 10 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 $\text{wf}_{t,d}$ 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 $\text{tf}_{t,d}$ for each postings entry. Finally, Step 12 extracts the top K scores – this requires a priority queue

TERM-AT-A-TIME

ACCUMULATOR

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