7.1 Efficient scoring and ranking

We begin by recapping the algorithm of Figure 6.14. For a query such as q = jealous gossip, two observations are immediate:

- The unit vector ~v(q) has only two non-zero components.
- In the absence of any weighting for query terms, these non-zero components are equal – in this case, both equal 0.707.

For the purpose of ranking the documents matching this query, we are really interested in the relative (rather than absolute) scores of the documents in the collection. To this end, it suffices to compute the cosine similarity from each document unit vector ~v(d) to V~ (q) (in which all non-zero components of the query vector are set to 1), rather than to the unit vector ~v(q). For any

FASTCOSINESCORE(q)

- float Scores[N] = 0
- for each d
- do Initialize Length[d] to the length of doc d
- for each query term t
- do calculate wt,q and fetch postings list for t
- for each pair(d, tft,d) in postings list
- do add wft,d to Scores[d]
- Read the array Length[d]
- for each d
- do Divide Scores[d] by Length[d]
- return Top K components of Scores[]

two documents d1, d2

For any document d, the cosine similarity V~ (q) · ~v(d) is the weighted sum, over all terms in the query q, of the weights of those terms in d. This in turn can be computed by a postings intersection exactly as in the algorithm of Figure 6.14, with line 8 altered since we take wt, q to be 1 so that the multiply-add in that step becomes just an addition; the result is shown in Figure 7.1. We walk through the postings in the inverted index for the terms in q, accumulating the total score for each document – very much as in processing a Boolean query, except we assign a positive score to each document that appears in any of the postings being traversed. As mentioned in Section 6.3.3 we maintain an idf value for each dictionary term and a tf value for each postings entry. This scheme computes a score for every document in the postings of any of the query terms; the total number of such documents may be considerably smaller than N.

Given these scores, the final step before presenting results to a user is to pick out the K highest-scoring documents. While one could sort the complete set of scores, a better approach is to use a heap to retrieve only the top K documents in order. Where J is the number of documents with

non-zero cosine scores, constructing such a heap can be performed in 2J comparison steps, following which each of the K highest scoring documents can be "read off" the heap with log J comparison steps.

7.1.1 Inexact top K document retrieval

Thus far, we have focused on retrieving precisely the K highest-scoring documents for a query. We now consider schemes by which we produce K documents that are likely to be among the K highest scoring documents for a query. In doing so, we hope to dramatically lower the cost of computing the K documents we output, without materially altering the user's perceived relevance of the top K results. Consequently, in most applications it suffices to retrieve K documents whose scores are very close to those of the K best. In the sections that follow we detail schemes that retrieve K such documents while potentially avoiding computing scores for most of the N documents in the collection.

Such inexact top-K retrieval is not necessarily, from the user's perspective, a bad thing. The top K documents by the cosine measure are in any case not necessarily the K best for the query: cosine similarity is only a proxy for the user's perceived relevance. In Sections 7.1.2–7.1.6 below, we give heuristics using which we are likely to retrieve K documents with cosine scores close to those of the top K documents. The principal cost in computing the output stems from computing cosine similarities between the query and a large number of documents. Having a large number of documents in contention also increases the selection cost in the final stage of culling the top K documents from a heap. We now consider a series of ideas designed to eliminate a large number of documents without computing their cosine scores.