**UNIT4**

**Computing Scores in a Complete Search System**

Computing Scores in a Complete Search System- Efficient scoring and ranking, Inexact retrieval, Champion lists, Impact ordering, Cluster pruning, Tiered indexes, Query term proximity, Evaluation in Information Retrieval: Standard test collections, unranked retrieval sets, Ranked retrieval results, Assessing relevance, Relevance feedback.

## Computing Scores in a Complete Search System

# 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:

1. The unit vector $\vec{v}(q)$ has only two non-zero components.
2. 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 $\vec{v}(d)$ to $\vec{V}(q)$ (in which all non-zero components of the query vector are set to 1), rather than to the unit vector $\vec{v}(q)$. For any two documents $d_1, d_2$

$$\vec{V}(q) \cdot \vec{v}(d_1) > \vec{V}(q) \cdot \vec{v}(d_2) \Leftrightarrow \vec{v}(q) \cdot \vec{v}(d_1) > \vec{v}(q) \cdot \vec{v}(d_2). \quad (34)$$

For any document $d$, the cosine similarity $\vec{V}(q) \cdot \vec{v}(d)$ 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 $w_{t,q}$ 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 $N$.

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

**Figure 7.1:** A faster algorithm for vector space scores.

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.

**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. The heuristics have the following two-step scheme:

1. Find a set $A$ of documents that are contenders, where $K < |A| \ll N$. $A$ does not necessarily contain the $K$ top-scoring documents for the query, but is likely to have many documents with scores near those of the top $K$.
2. Return the $K$ top-scoring documents in $A$.

From the descriptions of these ideas it will be clear that many of them require parameters to be tuned to the collection and application at hand; pointers to experience in setting these parameters may be found at the end of this chapter. It

should also be noted that most of these heuristics are well-suited to free text queries, but not for Boolean or phrase queries.

# Champion lists

The idea of *champion lists* (sometimes also called *fancy lists* or *top docs*) is to precompute, for each term $t$ in the dictionary, the set of the $r$ documents with the highest weights for $t$; the value of $r$ is chosen in advance. For tf-idf weighting, these would be the $r$ documents with the highest tf values for term $t$. We call this set of $r$ documents the *champion list* for term $t$.

Now, given a query $q$ we create a set $A$ as follows: we take the union of the champion lists for each of the terms comprising $q$. We now restrict cosine computation to only the documents in $A$. A critical parameter in this scheme is the value $r$, which is highly application dependent. Intuitively, $r$ should be large compared with $K$, especially if we use any form of the index elimination described in Section 7.1.2 . One issue here is that the value $r$ is set at the time of index construction, whereas $K$ is application dependent and may not be available until the query is received; as a result we may (as in the case of index elimination) find ourselves with a set $A$ that has fewer than $K$ documents. There is no reason to have the same value of $r$ for all terms in the dictionary; it could for instance be set to be higher for rarer terms.

### Impact ordering

In all the postings lists described thus far, we order the documents consistently by some common ordering: typically by document ID but in Section 7.1.4 by static quality scores. As noted at the end of Section 6.3.3 , such a common ordering supports the concurrent traversal of all of the query terms' postings lists, computing the score for each document as we encounter it. Computing scores in this manner is sometimes referred to as *document-at-a-time* scoring. We will now introduce a technique for inexact top-$K$ retrieval in which the postings are not all ordered by a common ordering, thereby precluding such a concurrent traversal. We will therefore require scores to be ``accumulated'' one term at a time as in the scheme of Figure 6.14 , so that we have *term-at-a-time* scoring.

### Static quality scores and ordering

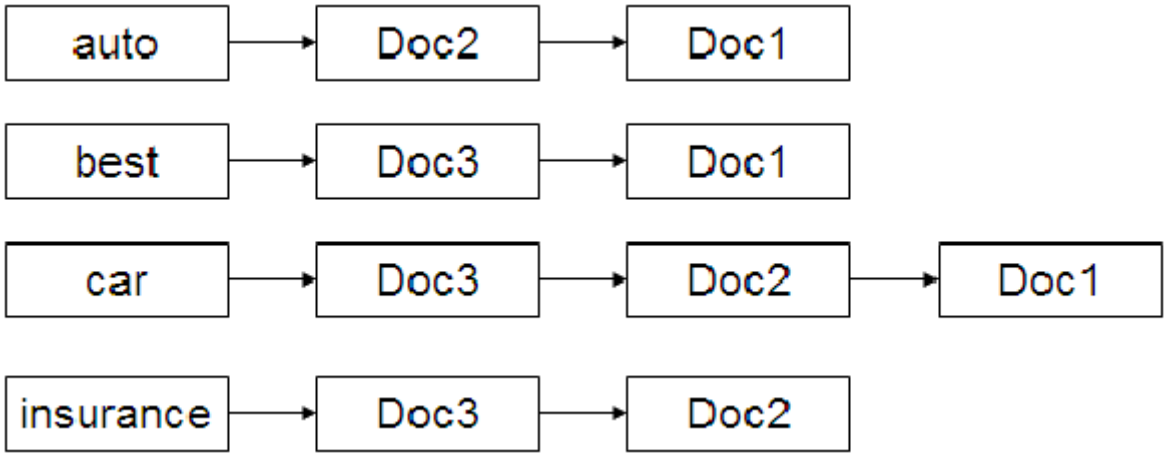
We now further develop the idea of champion lists, in the somewhat more general setting of *static quality scores* . In many search engines, we have available a measure of quality $g(d)$ for each document $d$ that is query-independent and thus *static*. This quality measure may be viewed as a number between zero and one. For instance, in the context of news stories on the web, $g(d)$ may be derived from the number of favorable reviews of the story by web surfers. otherindexing provides further discussion on this topic, as does Chapter 21 in the context of web search.

The net score for a document $d$ is some combination of $g(d)$ together with the query-dependent score induced (say) by (27). The precise combination may be determined by the learning methods of Section 6.1.2 , to be developed further in Section 15.4.1 ; but for the purposes of our exposition here, let us consider a simple sum:

$$\text{net-score}(q,d) = g(d) + \frac{\vec{V}(q)\cdot\vec{V}(d)}{|\vec{V}(q)||\vec{V}(d)|}. \tag{35}$$

In this simple form, the static quality $g(d)$ and the query-dependent score from (24) have equal contributions, assuming each is between 0 and 1. Other relative weightings are possible; the effectiveness of our heuristics will depend on the specific relative weighting.

First, consider ordering the documents in the postings list for each term by decreasing value of $g(d)$. This allows us to perform the postings intersection algorithm of Figure 1.6 . In order to perform the intersection by a single pass through the postings of each query term, the algorithm of Figure 1.6 relied on the postings being ordered by document IDs. But in fact, we only required that all postings be ordered by a single common ordering; here we rely on the $g(d)$ values to provide this common ordering. This is illustrated in Figure 7.2 , where the postings are ordered in decreasing order of $g(d)$.




A static quality-ordered index.In this example we assume that Doc1, Doc2 and Doc3 respectively have static quality scores $g(1) = 0.25, g(2) = 0.5, g(3) = 1$.

The first idea is a direct extension of champion lists: for a well-chosen value $r$, we maintain for each term $t$ a *global champion list* of the $r$ documents with the highest values for $g(d) + \text{tf-idf}_{t,d}$. The list itself is, like all the postings lists considered so far, sorted by a common order (either by document IDs or by static quality). Then at query time, we only compute the net scores (35) for documents in the union of these global champion lists. Intuitively, this has the effect of focusing on documents likely to have large net scores.

We conclude the discussion of global champion lists with one further idea. We maintain for each term $t$ two postings lists consisting of disjoint sets of documents, each sorted by $g(d)$ values. The first list, which we call *high*, contains the $m$ documents with the highest tf values for $t$. The second list, which we call *low*, contains all other documents containing $t$. When processing a query, we first scan only the high lists of the query terms, computing net scores for any document on the high lists of all (or more than a certain number of) query terms. If we obtain scores for $K$ documents in the process, we terminate. If not, we continue the scanning into the low lists, scoring documents in these postings lists. This idea is developed further in Section 7.2.1 .

# Cluster pruning

In *cluster pruning* we have a preprocessing step during which we cluster the document vectors. Then at query time, we consider only documents in a small number of clusters as candidates for which we compute cosine scores. Specifically, the preprocessing step is as follows:

1. Pick $\sqrt{N}$ documents at random from the collection. Call these *leaders*.
2. For each document that is not a leader, we compute its nearest leader.

We refer to documents that are not leaders as *followers*. Intuitively, in the partition of the followers induced by the use of $\sqrt{N}$ randomly chosen leaders, the expected number of followers for each leader is $\approx N/\sqrt{N} = \sqrt{N}$. Next, query processing proceeds as follows:

1. Given a query $q$, find the leader $L$ that is closest to $q$. This entails computing cosine similarities from $q$ to each of the $\sqrt{N}$ leaders.

2. The candidate set $A$ consists of $L$ together with its followers. We compute the cosine scores for all documents in this candidate set.

The use of randomly chosen leaders for clustering is fast and likely to reflect the distribution of the document vectors in the vector space: a region of the vector space that is dense in documents is likely to produce multiple leaders and thus a finer partition into sub-regions. This illustrated in Figure 7.3 .

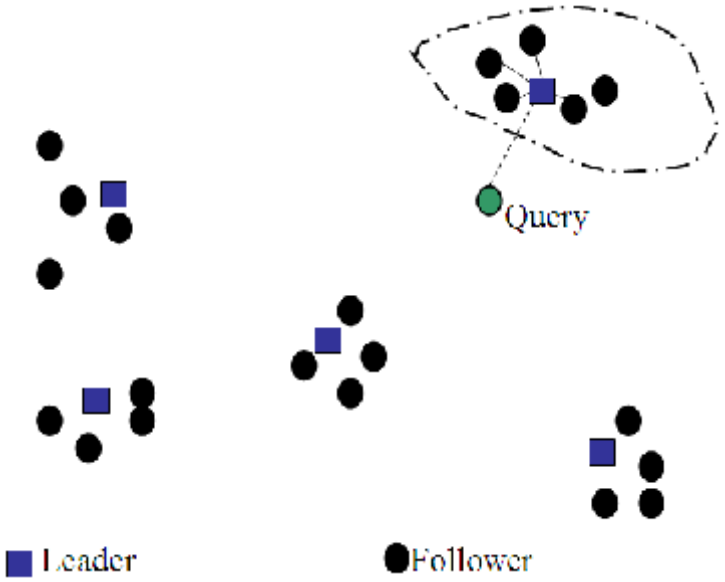
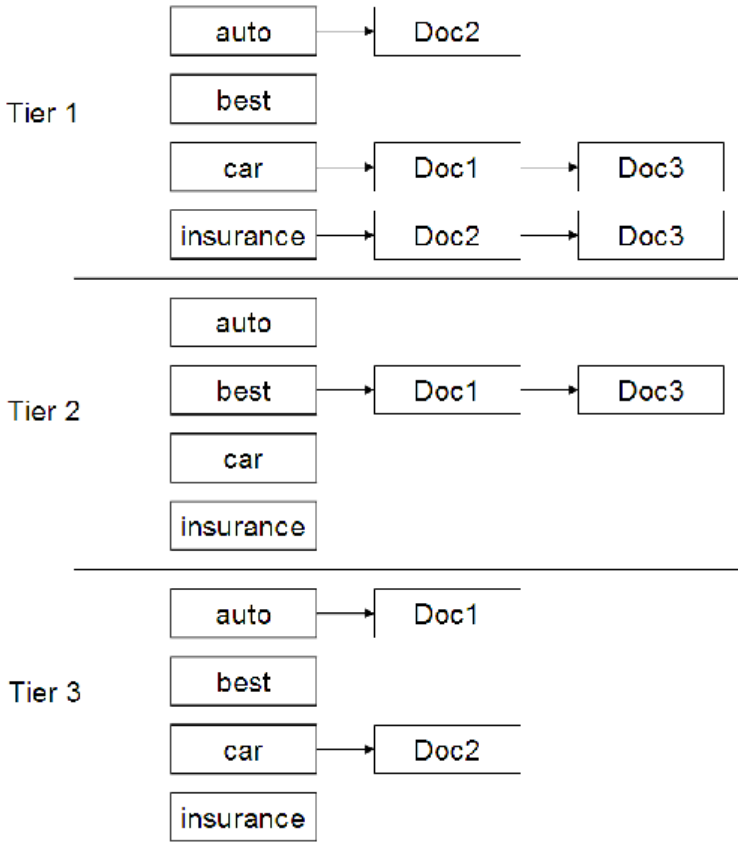



**Figure 7.3:** Cluster pruning.

Variations of cluster pruning introduce additional parameters $b_1$ and $b_2$, both of which are positive integers. In the pre-processing step we attach each follower to its $b_1$ closest leaders, rather than a single closest leader. At query time we consider the $b_2$ leaders closest to the query $q$. Clearly, the basic scheme above corresponds to the case $b_1 = b_2 = 1$. Further, increasing $b_1$ or $b_2$ increases the likelihood of finding $K$ documents that are more likely to be in the set of true top-scoring $K$ documents, at the expense of more computation. We reiterate this approach when describing clustering in Chapter 16 (page 16.1 ).

# Tiered indexes

We mentioned in Section 7.1.2 that when using heuristics such as index elimination for inexact top-$K$ retrieval, we may occasionally find ourselves with a set $A$ of contenders that has fewer than $K$ documents. A common solution to this issue is the user of *tiered indexes* , which may be viewed as a generalization of *champion lists* . We illustrate this idea in Figure 7.4 , where we represent the documents and terms of Figure 6.9 . In this example we set a tf threshold of 20 for tier 1 and 10 for tier 2, meaning that the tier 1 index only has postings entries with tf values exceeding 20, while the tier 2 index only has postings entries with tf values exceeding 10. In this example we have chosen to order the postings entries within a tier by document ID.




Tiered indexes.If we fail to get $K$ results from tier 1, query processing ``falls back'' to tier 2, and so on. Within each tier, postings are ordered by document ID.

# Query-term proximity

Especially for free text queries on the web (Chapter 19 ), users prefer a document in which most or all of the query terms appear close to each other, because this is evidence that the document has text focused on their query intent. Consider a query with two or more query terms, $t_1, t_2, \ldots, t_k$. Let $\omega$ be the width of the smallest window in a document $d$ that contains all the query terms, measured in the number of words in the window. For instance, if the document were to simply consist of the sentence The quality of mercy is not strained, the smallest window for the query strained mercy would be 4. Intuitively, the smaller that $\omega$ is, the better that $d$ matches the query. In cases where the document does not contain all of the query terms, we can set $\omega$ to be some enormous number. We could also consider variants in which only words that are not stop words are considered in computing $\omega$. Such proximity-weighted scoring functions are a departure from pure cosine similarity and closer to the ``soft conjunctive'' semantics that Google and other web search engines evidently use.

How can we design such a *proximity-weighted* scoring function to depend on $\omega$? The simplest answer relies on a ``hand coding'' technique we introduce below in Section 7.2.3 . A more scalable approach goes back to Section 6.1.2 - we treat the integer $\omega$ as yet another feature in the scoring function, whose importance is assigned by machine learning, as will be developed further in Section 15.4.1 .

We now consider a simple case of weighted zone scoring, where each document has a *title* zone and a *body* zone. Given a query $q$ and a document $d$, we use the given Boolean match function to compute Boolean variables $s_T(d,q)$ and $s_B(d,q)$, depending on whether the title (respectively, body) zone of $d$ matches query $q$. For instance, the algorithm in Figure 6.4 uses an AND of the query terms for this Boolean function. We will compute a score between 0 and 1 for each (document, query) pair using $s_T(d,q)$ and $s_B(d,q)$ by using a constant $g \in [0,1]$, as follows:

$$score(d,q) = g \cdot s_T(d,q) + (1-g) \cdot s_B(d,q). \tag{14}$$

We now describe how to determine the constant $g$ from a set of *training examples*, each of which is a triple of the form $\Phi_j = (d_j, q_j, r(d_j, q_j))$. In each training example, a given training document $d_j$ and a given training query $q_j$ are assessed by a human editor who delivers a relevance judgment $r(d_j, q_j)$ that is either *Relevant* or *Non-relevant*. This is illustrated in Figure 6.5 , where seven training examples are shown.

| Example | DocID | Query | $s_T$ | $s_B$ | Judgment |
|---|---|---|---|---|---|
| $\Phi_1$ | 37 | linux | 1 | 1 | Relevant |
| $\Phi_2$ | 37 | penguin | 0 | 1 | Non-relevant |
| $\Phi_3$ | 238 | system | 0 | 1 | Relevant |
| $\Phi_4$ | 238 | penguin | 0 | 0 | Non-relevant |
| $\Phi_5$ | 1741 | kernel | 1 | 1 | Relevant |
| $\Phi_6$ | 2094 | driver | 0 | 1 | Relevant |
| $\Phi_7$ | 3191 | driver | 1 | 0 | Non-relevant |

**Figure 6.5:** An illustration of training examples.

For each training example $\Phi_j$ we have Boolean values $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ that we use to compute a score from (14)

$$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j). \tag{15}$$

We now compare this computed score to the human relevance judgment for the same document-query pair $(d_j, q_j)$; to this end, we will quantize each *Relevant* judgment as a 1 and each *Non-relevant* judgment as a 0. Suppose that we define the error of the scoring function with weight $g$ as

$$\varepsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2, \tag{16}$$

where we have quantized the editorial relevance judgment $r(d_j, q_j)$ to 0 or 1. Then, the total error of a set of training examples is given by

$$\sum_j \varepsilon(g, \Phi_j). \tag{17}$$

The problem of learning the constant $\underline{g}$ from the given training examples then reduces to picking the value of $\underline{g}$ that minimizes the total error in (17).

Picking the best value of $\underline{g}$ in (17) in the formulation of Section 6.1.3 reduces to the problem of minimizing a quadratic function of $\underline{g}$ over the interval $[0,1]$. This reduction is detailed in Section 6.1.3

# Information retrieval system evaluation

To measure ad hoc information retrieval effectiveness in the standard way, we need a test collection consisting of three things:

1. A document collection
2. A test suite of information needs, expressible as queries
3. A set of relevance judgments, standardly a binary assessment of either *relevant* or *nonrelevant* for each query-document pair.

The standard approach to information retrieval system evaluation revolves around the notion of *relevant* and *nonrelevant* documents. With respect to a user information need, a document in the test collection is given a binary classification as either relevant or nonrelevant. This decision is referred to as the *gold standard* or *ground truth* judgment of relevance. The test document collection and suite of information needs have to be of a reasonable size: you need to average performance over fairly large test sets, as results are highly variable over different documents and information needs. As a rule of thumb, 50 information needs has usually been found to be a sufficient minimum.

Relevance is assessed relative to an , *not* a query. For example, an information need might be:

Information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.

This might be translated into a query such as:
wine and red and white and heart and attack and effective
A document is relevant if it addresses the stated information need, not because it just happens to contain all the words in the query. This distinction is often misunderstood in practice, because the information need is not overt. But, nevertheless, an information need is present. If a user types python into a web search engine, they might be wanting to know where they can purchase a pet python. Or they might be wanting information on the programming language Python. From a one word query, it is very difficult for a system to know what the information need is. But, nevertheless, the user has one, and can judge the returned results on the basis of their relevance to it. To evaluate a system, we require an overt expression of an information need, which can be used for judging returned documents as relevant or nonrelevant. At this point, we make a simplification: relevance can reasonably be thought of as a scale, with some documents highly relevant and others marginally so. But for the moment, we will use just a binary decision of relevance. We discuss the reasons for using binary relevance judgments and alternatives in Section 8.5.1 .

Many systems contain various weights (often known as parameters) that can be adjusted to tune system performance. It is wrong to report results on a test collection which were obtained by tuning these parameters to maximize performance on that collection. That is because such tuning overstates the expected performance of the system, because the weights will be set to maximize performance on one particular set of queries rather than for a random sample of queries. In such cases, the correct procedure is to have one or more *development test collections* , and to tune the parameters on the development test collection. The tester then runs the system with those weights on the test collection and reports the results on that collection as an unbiased estimate of performance.

# Standard test collections

Here is a list of the most standard test collections and evaluation series. We focus particularly on test collections for ad hoc information retrieval system evaluation, but also mention a couple of similar test collections for text classification.

The *Cranfield* collection. This was the pioneering test collection in allowing precise quantitative measures of information retrieval effectiveness, but is nowadays too small for anything but the most elementary pilot experiments. Collected in the United Kingdom starting in the late 1950s, it contains 1398 abstracts of aerodynamics journal articles, a set of 225 queries, and exhaustive relevance judgments of all (query, document) pairs.

*Text Retrieval Conference (TREC)* . The U.S. *National Institute of Standards and Technology* (NIST) has run a large IR test bed evaluation series since 1992. Within this framework, there have been many tracks over a range of different test collections, but the best known test collections are the ones used for the TREC Ad Hoc track during the first 8 TREC evaluations between 1992 and 1999. In total, these test collections comprise 6 CDs containing 1.89 million documents (mainly, but not exclusively, newswire articles) and relevance judgments for 450 information needs, which are called *topics* and specified in detailed text passages. Individual test collections are defined over different subsets of this data. The early TRECs each consisted of 50 information needs, evaluated over different but overlapping sets of documents. TRECs 6-8 provide 150 information needs over about 528,000 newswire and Foreign Broadcast Information Service articles. This is probably the best subcollection to use in future work, because it is the largest and the topics are more consistent. Because the test document collections are so large, there are no exhaustive relevance judgments. Rather, NIST assessors' relevance judgments are available only for the documents that were among the

top $k$ returned for some system which was entered in the TREC evaluation for which the information need was developed.

In more recent years, NIST has done evaluations on larger document collections, including the 25 million page *GOV2* web page collection. From the beginning, the NIST test document collections were orders of magnitude larger than anything available to researchers previously and GOV2 is now the largest Web collection easily available for research purposes. Nevertheless, the size of GOV2 is still more than 2 orders of magnitude smaller than the current size of the document collections indexed by the large web search companies.

NII Test Collections for IR Systems ( *NTCIR* ). The NTCIR project has built various test collections of similar sizes to the TREC collections, focusing on East Asian language and *cross-language information retrieval* , where queries are made in one language over a document collection containing documents in one or more other languages. See: http://research.nii.ac.jp/ntcir/data/data-en.html

Cross Language Evaluation Forum ( *CLEF* ). This evaluation series has concentrated on European languages and cross-language information retrieval. See: http://www.clef-campaign.org/
and Reuters-RCV1. For text classification, the most used test collection has been the Reuters-21578 collection of 21578 newswire articles; see Chapter 13 , page 13.6 . More recently, Reuters released the much larger Reuters Corpus Volume 1 (RCV1), consisting of 806,791 documents; see Chapter 4 , page 4.2 . Its scale and rich annotation makes it a better basis for future research.
*20 Newsgroups* . This is another widely used text classification collection, collected by Ken Lang. It consists of 1000 articles from each of 20 Usenet newsgroups (the newsgroup name being regarded as the category). After the removal of duplicate articles, as it is usually used, it contains 18941 articles.

# Evaluation of unranked retrieval sets

Given these ingredients, how is system effectiveness measured? The two most frequent and basic measures for information retrieval effectiveness are precision and recall. These are first defined for the simple case where an IR system returns a set of documents for a query. We will see later how to extend these notions to ranked retrieval situations.

*Precision* ($P$) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved}) \tag{36}$$

*Recall* ($R$) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant}) \tag{37}$$

These notions can be made clear by examining the following contingency table: (G)

| | Relevant | Nonrelevant |
|---|---|---|
| Retrieved | true positives (tp) | false positives (fp) |
| Not retrieved | false negatives (fn) | true negatives (tn) |

Then:

$$P = tp/(tp+fp) \tag{38}$$

$$R = tp/(tp+fn) \tag{39}$$

An obvious alternative that may occur to the reader is to judge an information retrieval system by its *accuracy* , that is, the fraction of its classifications that are correct. In terms of the contingency table above, $\text{accuracy} = (tp+tn)/(tp+fp+fn+tn)$. This seems plausible, since there are two actual classes, relevant and nonrelevant, and an information retrieval system can be thought of as a two-class classifier which attempts to label them as such (it retrieves the subset of documents which it believes to be relevant). This is precisely the effectiveness measure often used for evaluating machine learning classification problems.

There is a good reason why accuracy is not an appropriate measure for information retrieval problems. In almost all circumstances, the data is extremely skewed: normally over 99.9% of the documents are in the nonrelevant category. A system tuned to maximize accuracy can appear to perform well by simply deeming all

documents nonrelevant to all queries. Even if the system is quite good, trying to label some documents as relevant will almost always lead to a high rate of false positives. However, labeling all documents as nonrelevant is completely unsatisfying to an information retrieval system user. Users are always going to want to see some documents, and can be assumed to have a certain tolerance for seeing some false positives providing that they get some useful information. The measures of precision and recall concentrate the evaluation on the return of true positives, asking what percentage of the relevant documents have been found and how many false positives have also been returned.

The advantage of having the two numbers for precision and recall is that one is more important than the other in many circumstances. Typical web surfers would like every result on the first page to be relevant (high precision) but have not the slightest interest in knowing let alone looking at every document that is relevant. In contrast, various professional searchers such as paralegals and intelligence analysts are very concerned with trying to get as high recall as possible, and will tolerate fairly low precision results in order to get it. Individuals searching their hard disks are also often interested in high recall searches. Nevertheless, the two quantities clearly trade off against one another: you can always get a recall of 1 (but very low precision) by retrieving all documents for all queries! Recall is a non-decreasing function of the number of documents retrieved. On the other hand, in a good system, precision usually decreases as the number of documents retrieved is increased. In general we want to get some amount of recall while tolerating only a certain percentage of false positives.

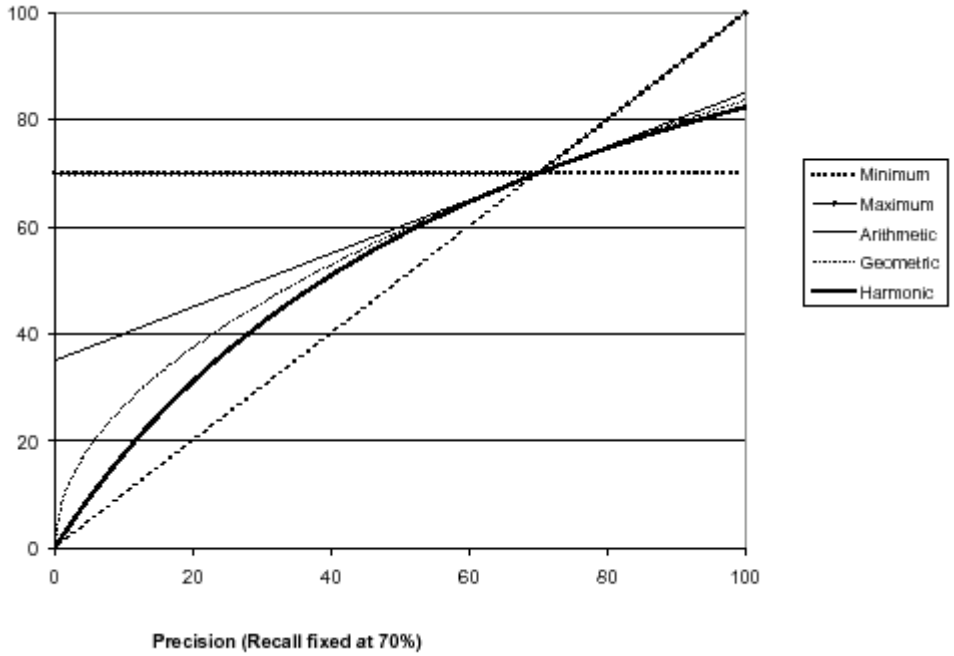
A single measure that trades off precision versus recall is the *F measure* , which is the weighted harmonic mean of precision and recall:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha} \tag{40}$$

where $\alpha \in [0,1]$ and thus $\beta^2 \in [0,\infty]$. The default *balanced F measure* equally weights precision and recall, which means making $\alpha = 1/2$ or $\beta = 1$. It is commonly written as $F_1$, which is short for $F_{\beta=1}$, even though the formulation in terms of $\alpha$ more transparently exhibits the F measure as a weighted harmonic mean. When using $\beta = 1$, the formula on the right simplifies to:

$$F_{\beta=1} = \frac{2PR}{P+R} \tag{41}$$

However, using an even weighting is not the only choice. Values of $\beta < 1$ emphasize precision, while values of $\beta > 1$ emphasize recall. For example, a value of $\beta = 3$ or $\beta = 5$ might be used if recall is to be emphasized. Recall, precision, and the F measure are inherently measures between 0 and 1, but they are also very commonly written as percentages, on a scale between 0 and 100.




Graph comparing the harmonic mean to other means.The graph shows a slice through the calculation of various means of precision and recall for the fixed recall value of 70%. The harmonic mean is always less than either the arithmetic or geometric mean, and often quite close to the minimum of the two numbers. When the precision is also 70%, all the measures coincide.

Why do we use a harmonic mean rather than the simpler average (arithmetic mean)? Recall that we can always get 100% recall by just returning all documents, and therefore we can always get a 50% arithmetic mean by the same process. This strongly suggests that the arithmetic mean is an unsuitable measure to use. In contrast, if we assume that 1 document in 10,000 is relevant to the query, the harmonic mean score of this strategy is 0.02%. The harmonic mean is always less than or equal to the arithmetic mean and the geometric mean. When the values of two numbers differ greatly, the harmonic mean is closer to their minimum than to their arithmetic mean; see Figure 8.1 .

# Evaluation of ranked retrieval results

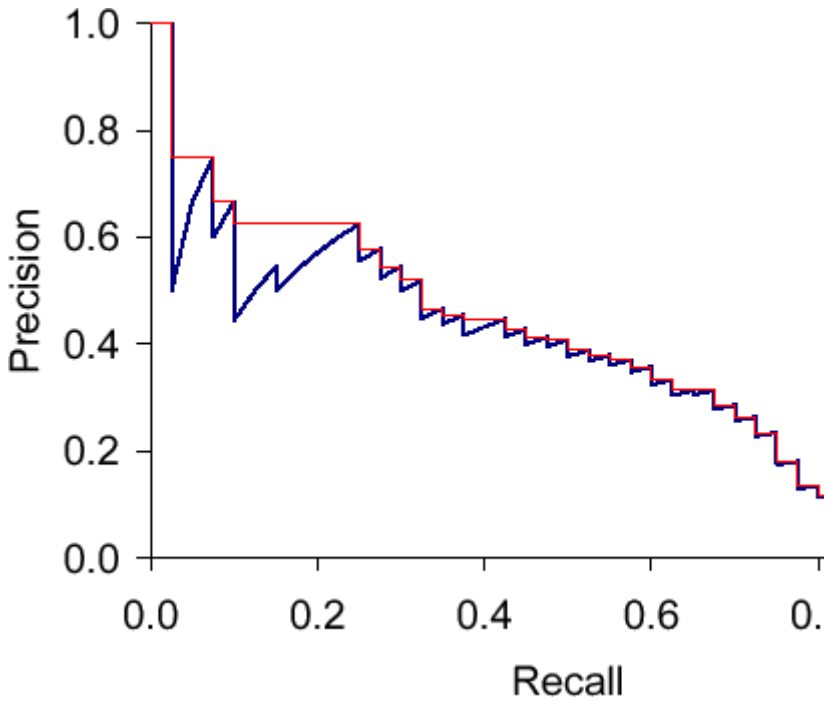



**Figure 8.2:** Precision/recall graph.

Precision, recall, and the F measure are set-based measures. They are computed using unordered sets of documents. We need to extend these measures (or to define new measures) if we are to evaluate the ranked retrieval results that are now standard with search engines. In a ranked retrieval context, appropriate sets of retrieved documents are naturally given by the top $k$ retrieved documents. For each such set, precision and recall values can be plotted to give a *precision-recall curve* , such as the one shown in Figure 8.2 . Precision-recall curves have a distinctive saw-tooth shape: if the $(k+1)^{th}$ document retrieved is nonrelevant then recall is the same as for the top $k$ documents, but precision has dropped. If it is relevant, then both precision and recall increase, and the curve jags up and to the right. It is often useful to remove these jiggles and the standard way to do this is with an interpolated precision: the *interpolated precision* $p_{interp}$ at a certain recall level $r$ is defined as the highest precision found for any recall level $r' \geq r$:

$$p_{interp}(r) = \max_{r' \geq r} p(r') \tag{42}$$

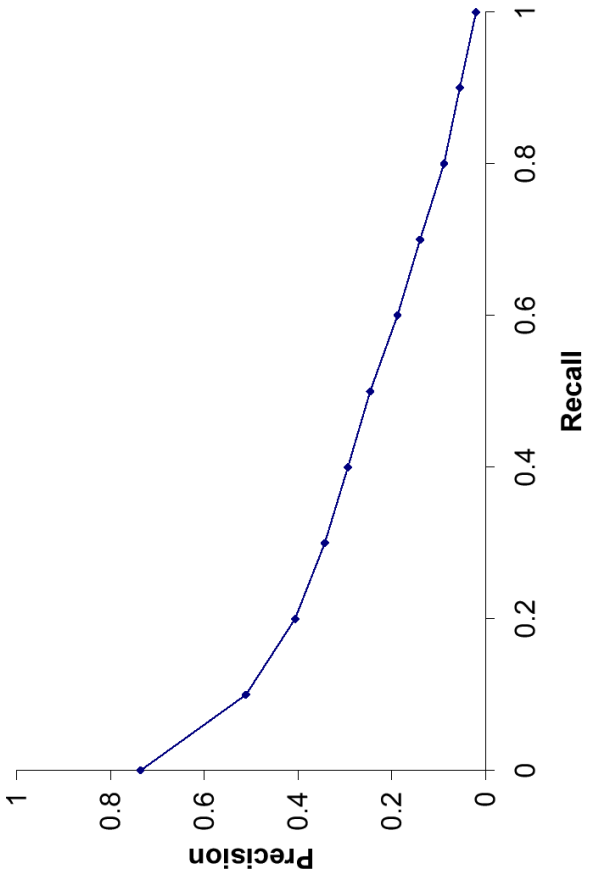
The justification is that almost anyone would be prepared to look at a few more documents if it would increase the percentage of the viewed set that were relevant (that is, if the precision of the larger set is higher). Interpolated precision is shown

by a thinner line in Figure 8.2 . With this definition, the interpolated precision at a recall of 0 is well-defined (Exercise 8.4 ).

| Recall | Interp. Precision |
|---|---|
| 0.0 | 1.00 |
| 0.1 | 0.67 |
| 0.2 | 0.63 |
| 0.3 | 0.55 |
| 0.4 | 0.45 |
| 0.5 | 0.41 |
| 0.6 | 0.36 |
| 0.7 | 0.29 |
| 0.8 | 0.13 |
| 0.9 | 0.10 |
| 1.0 | 0.08 |

Calculation of 11-point Interpolated Average Precision.This is for the precision-recall curve shown in Figure 8.2 .

Examining the entire precision-recall curve is very informative, but there is often a desire to boil this information down to a few numbers, or perhaps even a single number. The traditional way of doing this (used for instance in the first 8 TREC Ad Hoc evaluations) is the *11-point interpolated average precision* . For each information need, the interpolated precision is measured at the 11 recall levels of 0.0, 0.1, 0.2, ..., 1.0. For the precision-recall curve in Figure 8.2 , these 11 values are shown in Table 8.1 . For each recall level, we then calculate the arithmetic mean of the interpolated precision at that recall level for each information need in the test collection. A composite precision-recall curve showing 11 points can then be graphed. Figure 8.3 shows an example graph of such results from a representative good system at TREC 8.




Averaged 11-point precision/recall graph across 50 queries for a representative TREC system.The Mean Average Precision for this system is 0.2553.

In recent years, other measures have become more common. Most standard among the TREC community is *Mean Average Precision* (MAP), which provides a single-figure measure of quality across recall levels. Among evaluation measures, MAP has been shown to have especially good discrimination and stability. For a single information need, Average Precision is the average of the precision value obtained for the set of top $k$ documents existing after each relevant document is retrieved, and this value is then averaged over information needs. That is, if the set of relevant documents for an information need $q_j \in Q$ is $\{d_1, \ldots d_{m_j}\}$ and $R_{jk}$ is the set of ranked retrieval results from the top result until you get to document $d_k$, then

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \tag{43}$$

When a relevant document is not retrieved at all, the precision value in the above equation is taken to be 0. For a single information need, the average precision approximates the area under the uninterpolated precision-recall curve, and so the

MAP is roughly the average area under the precision-recall curve for a set of queries.

Using MAP, fixed recall levels are not chosen, and there is no interpolation. The MAP value for a test collection is the arithmetic mean of average precision values for individual information needs. (This has the effect of weighting each information need equally in the final reported number, even if many documents are relevant to some queries whereas very few are relevant to other queries.) Calculated MAP scores normally vary widely across information needs when measured within a single system, for instance, between 0.1 and 0.7. Indeed, there is normally more agreement in MAP for an individual information need across systems than for MAP scores for different information needs for the same system. This means that a set of test information needs must be large and diverse enough to be representative of system effectiveness across different queries.

The above measures factor in precision at all recall levels. For many prominent applications, particularly web search, this may not be germane to users. What matters is rather how many good results there are on the first page or the first three pages. This leads to measuring precision at fixed low levels of retrieved results, such as 10 or 30 documents. This is referred to as ``Precision at $k$'', for example ``Precision at 10''. It has the advantage of not requiring any estimate of the size of the set of relevant documents but the disadvantages that it is the least stable of the commonly used evaluation measures and that it does not average well, since the total number of relevant documents for a query has a strong influence on precision at $k$.

An alternative, which alleviates this problem, is *R-precision* . It requires having a set of known relevant documents $Rel$, from which we calculate the precision of the top $Rel$ documents returned. (The set $Rel$ may be incomplete, such as when $Rel$ is formed by creating relevance judgments for the pooled top $k$ results of particular systems in a set of experiments.) R-precision adjusts for the size of the set of relevant documents: A perfect system could score 1 on this metric for each query, whereas, even a perfect system could only achieve a precision at 20 of 0.4 if there were only 8 documents in the collection relevant to an information need. Averaging this measure across queries thus makes more sense. This measure is harder to explain to naive users than Precision at $k$ but easier to explain than MAP. If there are $|Rel|$ relevant documents for a query, we examine the top $|Rel|$ results of a system, and find that $r$ are relevant, then by definition, not only is the

precision (and hence R-precision) $r/|Rel|$, but the recall of this result set is also $r/|Rel|$. Thus, R-precision turns out to be identical to the *break-even point*, another measure which is sometimes used, defined in terms of this equality relationship holding. Like Precision at $k$, R-precision describes only one point on the precision-recall curve, rather than attempting to summarize effectiveness across the curve, and it is somewhat unclear why you should be interested in the break-even point rather than either the best point on the curve (the point with maximal F-measure) or a retrieval level of interest to a particular application (Precision at $k$). Nevertheless, R-precision turns out to be highly correlated with MAP empirically, despite measuring only a single point on the curve.

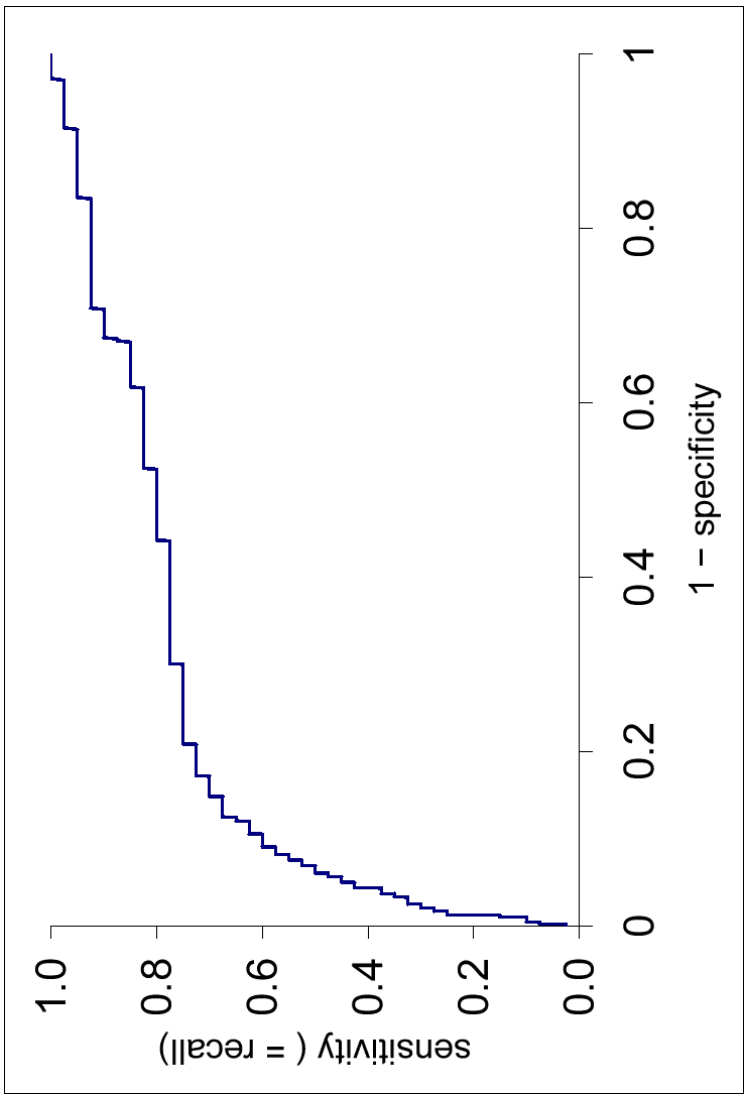



**Figure 8.4:** The ROC curve corresponding to the precision-recall curve in Figure 8.2 .

Another concept sometimes used in evaluation is an *ROC curve* . (``ROC" stands for ``Receiver Operating Characteristics", but knowing that doesn't help most

people.) An ROC curve plots the true positive rate or sensitivity against the false positive rate or ($1 - \text{specificity}$). Here, *sensitivity* is just another term for recall. The false positive rate is given by $fp/(fp+tn)$. Figure 8.4 shows the ROC curve corresponding to the precision-recall curve in Figure 8.2 . An ROC curve always goes from the bottom left to the top right of the graph. For a good system, the graph climbs steeply on the left side. For unranked result sets, *specificity* , given by $tn/(fp+tn)$, was not seen as a very useful notion. Because the set of true negatives is always so large, its value would be almost 1 for all information needs (and, correspondingly, the value of the false positive rate would be almost 0). That is, the ``interesting" part of Figure 8.2 is $0 < \text{recall} < 0.4$, a part which is compressed to a small corner of Figure 8.4 . But an ROC curve could make sense when looking over the full retrieval spectrum, and it provides another way of looking at the data. In many fields, a common aggregate measure is to report the area under the ROC curve, which is the ROC analog of MAP. Precision-recall curves are sometimes loosely referred to as ROC curves. This is understandable, but not accurate.

A final approach that has seen increasing adoption, especially when employed with machine learning approaches to ranking svm-ranking is measures of *cumulative gain* , and in particular *normalized discounted cumulative gain* ( *NDCG* ). NDCG is designed for situations of non-binary notions of relevance (cf. Section 8.5.1 ). Like precision at $k$, it is evaluated over some number $k$ of top search results. For a set of queries $Q$, let $R(j,d)$ be the relevance score assessors gave to document $d$ for query $j$. Then,

$$\text{NDCG}(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_{kj} \sum_{m=1}^{k} \frac{2^{R(j,m)}-1}{\log_2(1+m)}, \tag{44}$$

where $Z_{kj}$ is a normalization factor calculated to make it so that a perfect ranking's NDCG at $k$ for query $j$ is 1. For queries for which $k' < k$ documents are retrieved, the last summation is done up to $k'$.

# Assessing relevance

To properly evaluate a system, your test information needs must be germane to the documents in the test document collection, and appropriate for predicted usage of the system. These information needs are best designed by domain experts. Using random combinations of query terms as an information need is generally not a good idea because typically they will not resemble the actual distribution of information needs.

Given information needs and documents, you need to collect relevance assessments. This is a time-consuming and expensive process involving human beings. For tiny collections like Cranfield, exhaustive judgments of relevance for each query and document pair were obtained. For large modern collections, it is usual for relevance to be assessed only for a subset of the documents for each query. The most standard approach is *pooling* , where relevance is assessed over a subset of the collection that is formed from the top $k$ documents returned by a number of different IR systems (usually the ones to be evaluated), and perhaps other sources such as the results of Boolean keyword searches or documents found by expert searchers in an interactive process.

**Table 8.2:** Calculating the kappa statistic.

| | | Judge 2 Relevance | | | |
|---|---|---|---|---|---|
| | | Yes | | No | Total |
| Judge 1 | Yes | 300 | | 20 | 320 |
| Relevance | No | 10 | | 70 | 80 |
| | Total | 310 | | 90 | 400 |

Observed proportion of the times the judges agreed

$$P(A) = (300 + 70)/400 = 370/400 = 0.925$$

Pooled marginals

$$P(nonrelevant) = (80 + 90)/(400 + 400) = 170/800 = 0.2125$$

$$P(relevant) = (320 + 310)/(400 + 400) = 630/800 = 0.7878$$

Probability that the two judges agreed by chance

$$P(E) = P(nonrelevant)^2 + P(relevant)^2 = 0.2125^2 + 0.7878^2 = 0.665$$

Kappa statistic

$$\kappa = (P(A) - P(E))/(1 - P(E)) = (0.925 - 0.665)/(1 - 0.665) = 0.776$$

A human is not a device that reliably reports a gold standard judgment of relevance of a document to a query. Rather, humans and their relevance judgments are quite idiosyncratic and variable. But this is not a problem to be solved: in the final analysis, the success of an IR system depends on how good it is at satisfying the needs of these idiosyncratic humans, one information need at a time.

Nevertheless, it is interesting to consider and measure how much agreement between judges there is on relevance judgments. In the social sciences, a common measure for agreement between judges is the *kappa statistic* . It is designed for categorical judgments and corrects a simple agreement rate for the rate of chance agreement.

$$kappa = \frac{P(A) - P(E)}{1 - P(E)} \tag{46}$$

where $P(A)$ is the proportion of the times the judges agreed, and $P(E)$ is the proportion of the times they would be expected to agree by chance. There are choices in how the latter is estimated: if we simply say we are making a two-class decision and assume nothing more, then the expected chance agreement rate is 0.5. However, normally the class distribution assigned is skewed, and it is usual to use *marginal* statistics to calculate expected agreement. There are still two ways to do it depending on whether one pools the marginal distribution across judges or uses the marginals for each judge separately; both forms have been used, but we present the pooled version because it is more conservative in the presence of systematic differences in assessments across judges. The calculations are shown in Table 8.2 . The kappa value will be 1 if two judges always agree, 0 if they agree only at the rate given by chance, and negative if they are worse than random. If there are more than two judges, it is normal to calculate an average pairwise kappa value. As a rule of thumb, a kappa value above 0.8 is taken as good agreement, a kappa value between 0.67 and 0.8 is taken as fair agreement, and agreement below 0.67 is seen as data providing a dubious basis for an evaluation, though the precise cutoffs depend on the purposes for which the data will be used.

Interjudge agreement of relevance has been measured within the TREC evaluations and for medical IR collections. Using the above rules of thumb, the level of agreement normally falls in the range of ``fair'' (0.67-0.8). The fact that human agreement on a binary relevance judgment is quite modest is one reason for not requiring more fine-grained relevance labeling from the test set creator. To answer the question of whether IR evaluation results are valid despite the variation of individual assessors' judgments, people have experimented with evaluations taking one or the other of two judges' opinions as the gold standard. The choice can make a considerable *absolute* difference to reported scores, but has in general been found

to have little impact on the *relative* effectiveness ranking of either different systems or variants of a single system which are being compared for effectiveness.

# Relevance feedback and query expansion

In most collections, the same concept may be referred to using different words. This issue, known as *synonymy* , has an impact on the recall of most information retrieval systems. For example, you would want a search for aircraft to match plane (but only for references to an *airplane*, not a woodworking plane), and for a search on thermodynamics to match references to heat in appropriate discussions. Users often attempt to address this problem themselves by manually refining a query, as was discussed in Section 1.4 ; in this chapter we discuss ways in which a system can help with query refinement, either fully automatically or with the user in the loop.

The methods for tackling this problem split into two major classes: global methods and local methods. Global methods are techniques for expanding or reformulating query terms independent of the query and results returned from it, so that changes in the query wording will cause the new query to match other semantically similar terms. Global methods include:

- Query expansion/reformulation with a thesaurus or WordNet (Section 9.2.2 )
- Query expansion via automatic thesaurus generation (Section 9.2.3 )
- Techniques like spelling correction (discussed in Chapter 3 )

Local methods adjust a query relative to the documents that initially appear to match the query. The basic methods here are:

- Relevance feedback (Section 9.1 )
- Pseudo relevance feedback, also known as Blind relevance feedback (Section 9.1.6 )
- (Global) indirect relevance feedback (Section 9.1.7 )

### The Rocchio algorithm for relevance feedback

The Rocchio Algorithm is the classic algorithm for implementing relevance feedback. It models a way of incorporating relevance feedback information into the vector space model of Section 6.3 .

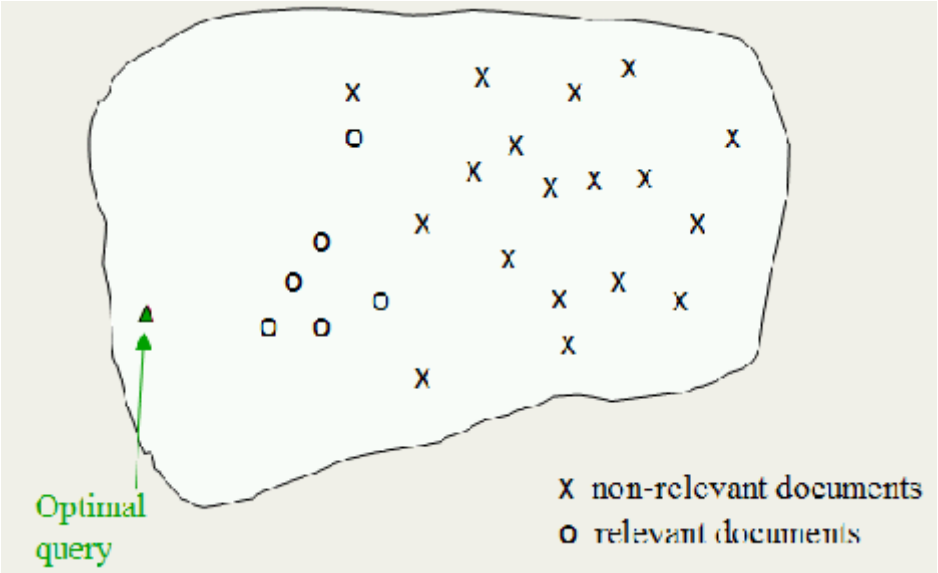



**Figure 9.3:** The Rocchio optimal query for separating relevant and nonrelevant documents.