8

Evaluating Search Engines

"Evaluation, Mr. Spock"

Captain Kirk, *Star Trek: The Motion Picture*

8.1 Why Evaluate?

Evaluation is the key to making progress in building better search engines. It is also essential to understanding if a search engine is being used effectively in a specific application. Engineers don't make decisions about a new design for a commercial aircraft based on whether it *feels* better than another design. Instead, they test the performance of the design by simulations and experiments, evaluate everything again when a prototype is built, and then continue to monitor and tune the performance of the aircraft after it goes into service. Experience has shown us that ideas that we intuitively feel must improve search quality, or models that have appealing formal properties, can often have little or no impact when tested using quantitative experiments.

One of the primary distinctions made in the evaluation of search engines is between *effectiveness* and *efficiency*. Effectiveness, loosely speaking, measures the ability of the search engine to find the right information, and efficiency measures how quickly this is done. For a given query, and a specific definition of relevance, we can more precisely define effectiveness as a measure of how well the ranking produced by the search engine corresponds to a ranking based on user relevance judgments. Efficiency is defined in terms of the time and space requirements for the algorithm that produces the ranking. Viewed more generally, however, search is an interactive process involving different types of users with different information problems. In this environment, effectiveness and efficiency will be affected by many factors such as the interface used to display search results and techniques such as query suggestion and relevance feedback. Carrying out this type of holistic evaluation of effectiveness and efficiency, while important, is very difficult be-

cause of the many factors that must be controlled. For this reason, evaluation is more typically done in tightly defined experimental settings and this is the type of evaluation we focus on here.

Effectiveness and efficiency are related in that techniques that give a small boost to effectiveness may not be included in a search engine implementation if they would have a significant adverse effect on an efficiency measure such as query throughput. Generally speaking, however, information retrieval research focuses on improving the effectiveness of search, and when a technique has been established as being potentially useful, the focus shifts to finding efficient implementations. This is not to say that research on system architecture and efficiency is not important. The techniques described in Chapter 5 are a critical part of building a scalable and usable search engine and were primarily developed by research groups. The focus on effectiveness is based on the underlying goal of a search engine which is to find the relevant information. A search engine that is extremely fast is of no use unless it produces good results.

So is there a trade-off between efficiency and effectiveness? Some search engine designers discuss having "knobs" or parameters on their system that can be turned to favor either high-quality results or improved efficiency. The current situation, however, is that there is no reliable technique that significantly improves effectiveness that cannot be incorporated into a search engine due to efficiency considerations. This may change in the future.

In addition to efficiency and effectiveness, the other significant consideration in search engine design is cost. We may know how to implement a particular search technique efficiently, but to do so may require a huge investment in processors, memory, disk, and networking. In general, if we pick targets for any two of these three factors, the third will be determined. For example, if we want a particular level of effectiveness and efficiency, this will determine the cost of the system configuration. Alternatively, if we decide on efficiency and cost targets, it may have an impact on effectiveness. Two extreme cases of choices for these factors are searching using a pattern matching utility such as `grep`, or searching using an organization like the Library of Congress. Searching a large text database using `grep` will have poor effectiveness and poor efficiency, but will be very cheap. Searching using the staff analysts at the Library of Congress will produce excellent results (high effectiveness) due to the manual effort involved, will be efficient in terms of the user's time although it will involve a delay waiting for a response from the analysts, and will be very expensive. Searching directly using an effective search engine is designed to be a reasonable compromise between these extremes.

An important point about terminology is that "optimization" is frequently discussed in the context of evaluation. The retrieval and indexing techniques in a search engine have many parameters that can be adjusted to optimize performance, both in terms of effectiveness and efficiency. Typically the best values for these parameters are determined using training data and a cost function. Training data is a sample of the real data, and the cost function is the quantity based on the data that is being maximized (or minimized). For example, the training data could be samples of queries with relevance judgments, and the cost function for a ranking algorithm would be a particular effectiveness measure. The optimization process would use the training data to learn parameter settings for the ranking algorithm that maximized the effectiveness measure. This use of optimization is very different from "search engine optimization" which is the process of tailoring Web pages to ensure high rankings from search engines.

In the remainder of this chapter, we will discuss the most important evaluation measures, both for effectiveness and efficiency. We will also describe how experiments are carried out in controlled environments to ensure that the results are meaningful.

8.2 The Evaluation Corpus

One of the basic requirements for evaluation is that the results from different techniques can be compared. To do this comparison fairly and to ensure that experiments are repeatable, the experimental settings and data used must be fixed. Starting with the earliest large-scale evaluations of search performance in the 1960s, generally referred to as the Cranfield[1] experiments (Cleverdon, 1970), researchers assembled *test collections* consisting of documents, queries, and relevance judgments to address this requirement. In other language-related research fields, such as linguistics, machine translation, or speech recognition, a *text corpus* is a large amount of text, usually in the form of many documents, that is used for statistical analysis of various kinds. The test collection, or *evaluation corpus*, in information retrieval is unique in that the queries and relevance judgments for a particular search task are gathered in addition to the documents.

Test collections have changed over the years to the reflect the changes in data and user communities for typical search applications. As an example of these

[1] Named after the place in the United Kingdom where the experiments were done.

changes, the following three test collections were created at intervals of about 10 years, starting in the 1980s:

- CACM: Titles and abstracts from the Communications of the ACM from 1958-1979. Queries and relevance judgments generated by computer scientists.

- AP: Associated Press newswire documents from 1988-1990 (from TREC disks 1-3). Queries are the title fields from TREC topics 51-150. Topics and relevance judgments generated by government information analysts.

- GOV2: Web pages crawled from Web sites in the .gov domain during early 2004. Queries are the title fields from TREC topics 701-850. Topics and relevance judgments generated by government analysts.

The CACM collection was created when most search applications focused on *bibliographic records* containing titles and abstracts, rather than the full text of documents. Table 8.1 shows that the number of documents in the collection (3,204) and the average number of words per document (64) are both quite small. The total size of the document collection is only 2.2 megabytes, which is considerably less than the size of a single typical music file for an MP3 player. The queries for this collection of abstracts of computer science papers were generated by students and faculty of a computer science department, and are supposed to represent actual information needs. An example of a CACM query is

> Security considerations in local networks, network operating systems, and distributed systems.

Relevance judgments for each query were done by the same people, and were relatively *exhaustive* in the sense that most relevant documents were identified. This was possible since the collection is small and the people who generated the questions were very familiar with the documents. Table 8.2 shows that the CACM queries are quite long (13 words on average) and that there are an average of 16 relevant documents per query.

The AP and GOV2 collections were created as part of the TREC conference series sponsored by the National Institute of Standards and Technology (NIST). The AP collection is typical of the full text collections that were first used in the early 1990s. The availability of cheap magnetic disk technology and online text entry led to a number of search applications involving databases such as news stories, legal documents, and encyclopedia articles. The AP collection is much bigger

Collection	Number of documents	Size	Average number of words/doc.
CACM	3,204	2.2 Mb	64
AP	242,918	0.7 Gb	474
GOV2	25,205,179	426 Gb	1073

Table 8.1. Statistics for three example text collections. The average number of words per document is calculated without stemming.

(by two orders of magnitude) than the CACM collection, both in terms of number of documents and the total size. The average document is also considerably longer (474 versus 64 words) since they contain the full text of a news story. The GOV2 collection, which is another two orders of magnitude larger, was designed to be a testbed for Web search applications and was created by a crawl of the .gov domain. Many of these government Web pages contain lengthy policy descriptions or tables and consequently the average document length is the largest of the three collections.

Collection	Number of queries	Average number of words/query	Average number of relevant docs/query
CACM	64	13.0	16
AP	100	4.3	220
GOV2	150	3.1	180

Table 8.2. Statistics for queries from example text collections.

The queries for the AP and GOV2 collections are based on TREC topics. The topics were created by government information analysts employed by NIST. The early TREC topics were designed to reflect the needs of professional analysts in government and industry and were quite complex. Later TREC topics were supposed to represent more general information needs but they retained the TREC topic format. An example is shown in Figure 8.1. TREC topics contain three fields indicated by the tags. The title field is supposed to be a short query, more typical of a Web application. The description field is a longer version of the query, which as this example shows, can sometimes be more precise than the short query. The narrative field describes the criteria for relevance, which is used by the people doing relevance judgments to increase consistency, and should not be considered as

a query. Most recent TREC evaluations have focused on using the title field of the topic as the query, and our statistics in Table 8.2 are based on that field.

```
<top>
<num> Number: 794

<title> pet therapy

<desc> Description:
How are pets or animals used in therapy for humans and what are the
benefits?

<narr> Narrative:
Relevant documents must include details of how pet- or animal-assisted
therapy is or has been used.  Relevant details include information
about pet therapy programs, descriptions of the circumstances in which
pet therapy is used, the benefits of this type of therapy, the degree
of success of this therapy, and any laws or regulations governing it.

</top>
```

Fig. 8.1. Example of a TREC topic.

The relevance judgments in TREC depend on the task that is being evaluated. For the queries in these tables, the task emphasized high recall, where it is important not to miss information. Given the context of that task, TREC analysts judged a document as relevant if it contained information that could be used to help write a report on the query topic. In chapter 7, we discussed the difference between *user relevance* and *topical relevance*. Although this definition does refer to the usefulness of the information found, because analysts are instructed to include documents with duplicate information, it is primarily focused on topical relevance. All relevance judgments for the TREC and CACM collections are binary, meaning that a document is either relevant or it is not. For some tasks, multiple levels of relevance may be appropriate, and we discuss effectiveness measures for both binary and graded relevance in section 8.4. Different retrieval tasks can affect the number of relevance judgments required, as well as the type of judgments and the effectiveness measure. For example, in Chapter 7 we described *navigational* searches, where the user is looking for a particular page. In this case, there is only one relevant document for the query.

Creating a new test collection can be a time-consuming task. Relevance judgments, in particular, require a considerable investment of manual effort for the high-recall search task. When collections were very small, most of the documents in a collection could be evaluated for relevance. In a collection such as GOV2, however, this would clearly be impossible. Instead, a technique called *pooling* is used. In this technique, the top k results (for TREC, k varied between 50 and 200) from the rankings obtained by different search engines (or retrieval algorithms) are merged into a pool, duplicates are removed, and the documents are presented in some random order to the people doing the relevance judgments. Pooling produces a large number of relevance judgments for each query, as shown in Table 8.2. This list is, however, incomplete and, for a new retrieval algorithm that had not contributed documents to the original pool, this could potentially be a problem. Specifically, if a new algorithm found many relevant documents that were not part of the pool, they would be treated as being not relevant and the effectiveness of that algorithm could, as a consequence, be significantly underestimated. Studies with the TREC data, however, have shown that the relevance judgments are complete enough to produce accurate comparisons for new search techniques.

TREC corpora have been extremely useful for evaluating new search techniques, but they have limitations. A high-recall search task and collections of news articles are clearly not appropriate for evaluating product search on an e-commerce site, for example. New TREC "tracks" can be created to address important new applications, but this process can take months or years. On the other hand, new search applications and new data types such as blogs, forums, and annotated videos are constantly being developed. Fortunately, it is not that difficult to develop an evaluation corpus for any given application using the following basic guidelines:

1. Use a document database that is representative for the application in terms of the number, size, and type of documents. In some cases, this may be the actual database for the application, in others it will be a sample of the actual database, or even a similar database. If the target application is very general, then more than one database should be used to ensure that results are not corpus-specific. For example, in the case of the high-recall TREC task, a number of different news and government databases were used for evaluation.
2. The queries that are used for the test collection should also be representative of the queries submitted by users of the target application. These may be acquired either from a query log from a similar application or by asking poten-

tial users for examples of queries. Although it may be possible to gather tens of thousands of queries in some applications, the need for relevance judgments is a major constraint. The number of queries must be sufficient to establish that a new technique makes a significant difference. An analysis of TREC experiments has shown that with 25 queries, a difference in the effectiveness measure MAP (section 8.4.2) of .05 will result in the wrong conclusion about which system is better in about 13% of the comparisons. With 50 queries, this error rate falls below 4%. A difference of .05 in MAP is quite large. If a *significance test*, such as those discussed in section 8.6.1, is used in the evaluation, a *relative* difference of 10% in MAP is sufficient to guarantee a low error rate with 50 queries. If resources or the application makes more relevance judgments possible, it will be more productive in terms of producing reliable results to judge more queries rather than judging more documents from existing queries (i.e., increasing k). Strategies such as judging a small number (e.g., 10) of the top-ranked documents from many queries or selecting documents to judge that will make the most difference in the comparison (Carterette, Allan, & Sitaraman, 2006) have been shown to be effective. If a small number of queries are used, the results should only be considered indicative rather than conclusive. In that case, it is important that the queries should be at least representative and have good coverage in terms of the goals of the application. For example, if algorithms for local search were being tested, the queries in the test collection should include many different types of location information.

3. Relevance judgments should be done either by the people who asked the questions, or by independent judges who have been instructed in how to determine relevance for the application being evaluated. Relevance may seem to be a very subjective concept, and it is known that relevance judgments can vary depending on the person making the judgments or even for the same person at different times. Despite this variation, analysis of TREC experiments has shown that conclusions about the relative performance of systems are very stable. In other words, differences in relevance judgments do not have a significant effect on the error rate for comparisons. The number of documents that are evaluated for each query and the type of relevance judgments will depend on the effectiveness measures that are chosen. For most applications, it is generally easier for people to decide between at least three levels of relevance, which are *definitely relevant*, *definitely not relevant*, and *possibly relevant*. These can be converted into binary judgments by assigning the possibly relevant to either one of the other levels, if that is required for an effectiveness measure.

Some applications and effectiveness measures, however, may support more than 3 levels of relevance.

As a final point, it is worth emphasizing that many user actions can be considered *implicit* relevance judgments and that if these can be exploited this can substantially reduce the effort of constructing a test collection. For example, actions such as clicking on a document in a result list, moving it to a folder, or sending it to a printer may indicate that it is relevant. In previous chapters, we have described how query logs and clickthrough can be used for to support operations such as query expansion and spelling correction. In the next section, we discuss the role of query logs in search engine evaluation.

8.3 Logging

Query logs that capture user interactions with a search engine have become an extremely important resource for Web search engine development. From an evaluation perspective, these logs provide large amounts of data showing how users browse the results that a search engine provides for a query. In a general Web search application, the number of users and queries represented can number in the tens of millions. Compared to the hundreds of queries used in typical TREC collections, query log data can potentially support a much more extensive and realistic evaluation. The main drawback with this data is that it is not as precise as explicit relevance judgments.

An additional concern is maintaining the *privacy* of the users. This is particularly an issue when query logs are shared, distributed for research, or used to construct user profiles (see section 6.2.5). Various techniques can be used to *anonymize* the logged data, such as removing identifying information or queries that may contain personal data, although this can reduce the utility of the log for some purposes.

A typical query log will contain the following data for each query:

1. User identifier or user session identifier. This can be obtained in a number of ways. If a user logs on to a service, uses a search toolbar, or even allows cookies, this information allows the search engine to identify the user. A session is a series of queries submitted to a search engine over a limited amount of time. In some circumstances, it may only be possible to identify a user in the context of a session.
2. Query terms. The query is stored exactly as the user entered it.

3. List of URLs of results, their ranks on the result list, and whether they were clicked on[2].
4. Timestamp(s). The timestamp records the time that the query was submitted. Additional timestamps may also record the times that specific results were clicked on.

The clickthrough data in the log (item 3) has been shown to be highly correlated with explicit judgments of relevance when interpreted appropriately, and has been used for both training and evaluating search engines. More detailed information about user interaction can be obtained through a client-side application such as a search toolbar in a Web browser. Although this information is not always available, some user actions other than clickthrough have been shown to be good predictors of relevance. Two of the best predictors are page *dwell time* and search exit action. The page dwell time is the amount of time the user spends on a clicked result, measured from the initial click to the time when the user comes back to the results page or exits the search application. The search exit action is the way the user exits the search application, such as entering another URL, closing the browser window, or timing out. Other actions, such as printing a page, are very predictive but much less frequent.

Although clicks on result pages are highly correlated with relevance, they cannot be used directly in place of explicit relevance judgments because they are very biased toward pages that are highly ranked or have other features such as being popular or having a a good snippet on the result page. This means, for example, that pages at the top rank are clicked on much more frequently than lower ranked pages, even when the relevant pages are at the lower ranks. One approach to removing this bias is to use clickthrough data to predict user *preferences* between pairs of documents rather than relevance judgments. User preferences were first mentioned in section 7.6, where they were used to train a ranking function. A preference for document d_1 compared to document d_2 means that d_1 is more relevant or, equivalently, that it should be ranked higher. Preferences are most appropriate for search tasks where documents can have multiple levels of relevance, and are focused more on user relevance than purely topical relevance. Relevance judgments (either multi-level or binary) can be used to generate preferences, but preferences do not imply specific relevance levels.

[2] In some logs, only the clicked-on URLs are recorded. Logging all the results enables the generation of preferences, and provides a source of "negative" examples for various tasks.

The bias in clickthrough data is addressed by "strategies" or policies that generate preferences. These strategies are based on observations of user behavior and verified by experiments. One strategy that is similar to that described in section 7.6 is known as *Skip Above and Skip Next* (Agichtein, Brill, Dumais, & Ragno, 2006). This strategy assumes that, given a set of results for a query and a clicked result at rank position p, all unclicked results ranked above p are predicted to be less relevant than the result at p. In addition, unclicked results immediately following a clicked result are less relevant than the clicked result. For example, given a result list of ranked documents together with click data as follows:

d_1
d_2
d_3 (clicked)
d_4,

this strategy will generate the following preferences:

$d_3 > d_2$
$d_3 > d_1$
$d_3 > d_4$

Since preferences are only generated when higher ranked documents are ignored, a major source of bias is removed.

The "Skip" strategy uses the clickthrough patterns of individual users to generate preferences. This data can be noisy and inconsistent because of the variability in individual users' behavior. Since query logs typically contain many instances of the same query submitted by different users, clickthrough data can be aggregated to remove potential noise from individual differences. Specifically, *click distribution* information can be used to identify clicks that have a higher frequency than would be expected based on typical click patterns. These clicks have been shown to correlate well with relevance judgments. For a given query, we can use all the instances of that query in the log to compute the observed click frequency $O(d, p)$ for the result d in rank position p. We can also compute the expected click frequency $E(p)$ at rank p by averaging across all queries. The *click deviation* $CD(d, p)$ for a result d in position p is computed as

$$CD(d, p) = O(d, p) - E(p).$$

We can then use the value of $CD(d, p)$ to "filter" clicks and provide more reliable click information to the Skip strategy.

A typical evaluation scenario involves the comparison of the result lists for two or more systems for a given set of queries. Preferences are an alternate method of specifying which documents should be retrieved for a given query (relevance judgments being the typical method). The quality of the result lists for each system is then summarized using an effectiveness measure that is based on either preferences or relevance judgments. The following section describes the measures that are most commonly used in research and system development.

8.4 Effectiveness Metrics

8.4.1 Recall and Precision

The two most common effectiveness measures, *recall* and *precision*, were introduced in the Cranfield studies to summarize and compare search results. Intuitively, recall measures how well the search engine is doing at finding all the relevant documents for a query, and precision measures how well it is doing at rejecting non-relevant documents.

The definition of these measures assumes that, for a given query, there is a set of documents that is *retrieved* and a set that is *not retrieved* (the rest of the documents). This obviously applies to the results of a Boolean search, but the same definition can also be used with a ranked search, as we will see later. If, in addition, relevance is assumed to be binary, then the results for a query can be summarized as shown in Table 8.3. In this table, A is the relevant set of documents for the query, $\overline{A}$ is the nonrelevant set, B is the set of retrieved documents, and $\overline{B}$ is the set of documents that are not retrieved. The operator $\cap$ gives the intersection of two sets. For example, $A \cap B$ is the set of documents that are both relevant *and* retrieved.

	Relevant	Non-Relevant
Retrieved	$A \cap B$	$\overline{A} \cap B$
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$

Table 8.3. Sets of documents defined by a simple search with binary relevance.

A number of effectiveness measures can be defined using this table. The two we are particularly interested in are:

$$Recall = \frac{|A \cap B|}{|A|}$$
$$Precision = \frac{|A \cap B|}{|B|}$$

where $|.|$ gives the size of the set. In other words, recall is the proportion of relevant documents that are retrieved, and precision is the proportion of retrieved documents that are relevant. There is an implicit assumption in using these measures that the task involves retrieving as many of the relevant documents as possible, and minimizing the number of non-relevant documents retrieved. In other words, even if there are 500 relevant documents for a query, the user is interested in finding them all.

We can also view the search results summarized in Table 8.3 as the output of a binary classifier, as was mentioned in section 7.2.1. When a document is retrieved, it is the same as making a prediction that the document is relevant. From this perspective, there are two types of errors that can be made in prediction (or retrieval). These errors are called *false positives* (a nonrelevant document is retrieved) and *false negatives* (a relevant document is not retrieved). Recall is related to one type of error (the false negatives), but precision is not related directly to the other type of error. Instead, another measure known as *fallout*[3], which is the proportion of non-relevant documents that are retrieved, is related to the false positive errors:

$$Fallout = \frac{|\overline{A} \cap B|}{|\overline{A}|}$$

Given that fallout and recall together characterize the effectiveness of a search as a classifier, why do we use precision instead? The answer is simply that preci-

[3] In the classification and signal detection literature, the errors are known as Type I and Type II errors. Recall is often called the true positive rate, or sensitivity. Fallout is called the false positive rate, or the false alarm rate. Another measure used, specificity, is 1 - fallout. Precision is known as the positive predictive value, and is often used in medical diagnostic tests where the probability that a positive test is correct is particularly important. The true positive rate and the false positive rate are used to draw ROC (receiver operating characteristic) curves that show the tradeoff between these two quantities as the discrimination threshold varies. This threshold is the value at which the classifier makes a positive prediction. In the case of search, the threshold would correspond to a position in the document ranking. In information retrieval, recall-precision graphs are generally used instead of ROC curves.

sion is more meaningful to the user of a search engine. If 20 documents were retrieved for a query, a precision value of 0.7 means that 14 out of the 20 retrieved documents will be relevant. Fallout, on the other hand, will always be very small because there are so many non-relevant documents. If there were 1,000,000 non-relevant documents for the query used in the precision example, fallout would be 6/1000000 = 0.000006. If precision fell to 0.5, which would be noticeable to the user, fallout would be 0.00001. The skewed nature of the search task, where most of the corpus is not relevant to any given query, also means that evaluating a search engine as a classifier can lead to counter-intuitive results. A search engine trained to minimize classification errors would tend to retrieve nothing, since classifying a document as non-relevant is always a good decision!

The *F measure* is an effectiveness measure based on recall and precision that is used for evaluating classification performance and also in some search applications. It has the advantage of summarizing effectiveness in a single number. It is defined as the *harmonic mean* of recall and precision, which is:

$$F = \frac{1}{\frac{1}{2}(\frac{1}{R} + \frac{1}{P})} = \frac{2RP}{(R+P)}$$

Why use the harmonic mean instead of the usual arithmetic mean or average? The harmonic mean emphasizes the importance of small values, whereas the arithmetic mean is more affected by values that are unusually large (outliers). A search result that returned nearly the entire document database, for example, would have a recall of 1.0 and a precision near 0. The arithmetic mean of these values is 0.5, but the harmonic mean will be close to 0. The harmonic mean is clearly a better summary of the effectiveness of this retrieved set[4].

Most of the retrieval models we have discussed produce ranked output. To use recall and precision measures, retrieved sets of documents must be defined based on the ranking. One possibility is to calculate recall and precision values at every rank position. Figure 8.2 shows the top ten documents of two possible rankings, together with the recall and precision values calculated at every rank position for

[4] The more general form of the F measure is the *weighted harmonic mean*, which allows weights reflecting the relative importance of recall and precision to be used. This measure is $F = RP/(\alpha R + (1-\alpha)P)$, where α is a weight. This is often transformed using $\alpha = 1/(\beta^2+1)$, which gives $F_\beta = (\beta^2+1)RP/(R+\beta^2 P)$. The common F measure is in fact F_1, where recall and precision have equal importance. In some evaluations, precision or recall is emphasized by varying the value of β. Values of $\beta > 1$ emphasize recall.

a query that has six relevant documents. These rankings might correspond to, for example, the output of different retrieval algorithms or search engines.

At rank position 10 (i.e. when ten documents are retrieved), the two rankings have the same effectiveness as measured by recall and precision. Recall is 1.0 because all the relevant documents have been retrieved, and precision is 0.6 because both rankings contain 6 relevant documents in the retrieved set of 10 documents. At higher rank positions, however, the first ranking is clearly better. For example, at rank position 4 (4 documents retrieved), the first ranking has a recall of 0.5 (3 out of 6 relevant documents retrieved) and a precision of 0.75 (3 out of 4 retrieved documents are relevant). The second ranking has a recall of 0.17 (1/6) and a precision of 0.25 (1/4).

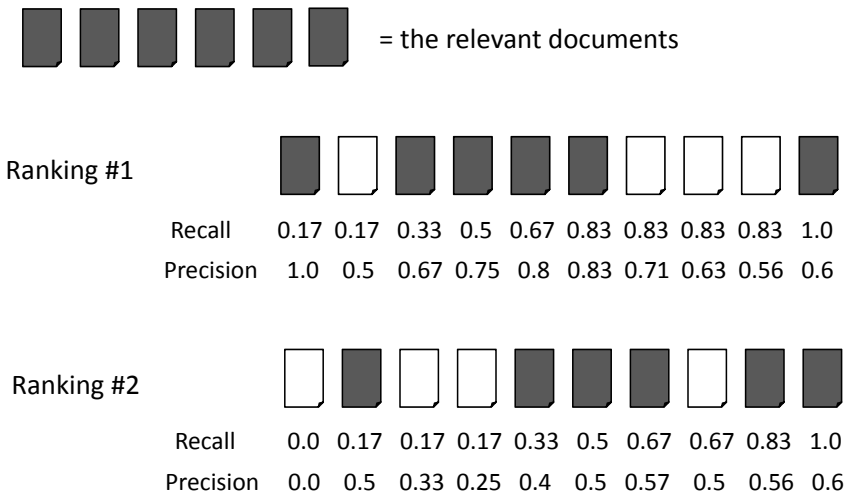

Fig. 8.2. Recall and precision values for two rankings of six relevant documents.

If there are a large number of relevant documents for a query, or if the relevant documents are widely distributed in the ranking, a list of recall-precision values for every rank position will be long and unwieldy. Instead, a number of techniques have been developed to summarize the effectiveness of a ranking. The first of these is simply to calculate recall-precision values at a small number of predefined rank positions. In fact, to compare two or more rankings for a given query, only the precision at the predefined rank positions needs to be calculated. If the precision for a ranking at rank position p is higher than the precision for another ranking,

the recall will be higher as well. This can be seen by comparing the corresponding recall-precision values in Figure 8.2. This effectiveness measure is known as *precision at rank* p. There are many possible values for the rank position p, but this measure is typically used to compare search output at the top of the ranking, since that is what many users care about. Consequently, the most common versions are precision at 10, and precision at 20. Note that if these measures are used, the implicit search task has changed to finding the most relevant documents at a given rank, rather than finding as many relevant documents as possible. Differences in search output further down the ranking than position 20 will not be considered. This measure also does not distinguish between differences in the rankings at positions 1 to p, which may be considered important for some tasks. For example, the two rankings in Figure 8.2 will be the same when measured using precision at 10.

Another method of summarizing the effectiveness of a ranking is to calculate precision at fixed or *standard* recall levels from 0.0 to 1.0 in increments of 0.1. Each ranking is then represented using eleven numbers. This method has the advantage of summarizing the effectiveness of the ranking of *all* relevant documents, rather than just those in the top ranks. Using the recall-precision values in Figure 8.2 as an example, however, it is clear that values of precision at these standard recall levels are often not available. In this example, only the precision values at the standard recall levels of 0.5 and 1.0 have been calculated. To obtain the precision values at all of the standard recall levels will require *interpolation*[5]. Since standard recall levels are used as the basis for averaging effectiveness across queries and generating *recall-precision graphs*, we will discuss interpolation in the next section.

The third method, and the most popular, is to summarize the ranking by averaging the precision values from the rank positions where a relevant document was retrieved (i.e. when recall increases). For the first ranking in Figure 8.2, the *average precision* is calculated as:

$$(1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

For the second ranking, it is

$$(0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

Average precision has a number of advantages. It is a single number that is based on the ranking of all the relevant documents, but the value depends heavily on the

[5] Interpolation refers to any technique for calculating a new point between two existing data points.

highly ranked relevant documents. This means it is an appropriate measure for evaluating the task of finding as many relevant documents as possible, while still reflecting the intuition that the top ranked documents are the most important.

All three of these methods summarize the effectiveness of a ranking for a single query. To provide a realistic assessment of the effectiveness of a retrieval algorithm, it must be tested on a number of queries. Given the potentially large set of results from these queries, we will need a method of summarizing the performance of the retrieval algorithm by calculating the average effectiveness for the entire set of queries. In the next section, we discuss the averaging techniques that are used in most evaluations.

8.4.2 Averaging and Interpolation

In the following discussion of averaging techniques, the two rankings shown in Figure 8.3 are used as a running example. These rankings come from using the same ranking algorithm on two *different* queries. The aim of an averaging technique is to summarize the effectiveness of a specific ranking algorithm across a collection of queries. Different queries will often have different numbers of relevant documents, as is the case in this example. Figure 8.3 also gives the recall-precision values calculated for the top ten rank positions.

Given that the average precision provides a number for each ranking, the simplest way to summarize the effectiveness of rankings from multiple queries would be to average these numbers. This effectiveness measure, *mean average precision*[6] or *MAP*, is used in most research papers and some system evaluations[7]. Since it is based on average precision, it assumes that the user is interested in finding many relevant documents for each query. Consequently, using this measure for comparison of retrieval algorithms or systems can require a considerable effort to acquire the relevance judgments, although methods for reducing the number of judgments required have been suggested (e.g., Carterette et al., 2006).

[6] This sounds a lot better than average average precision!

[7] In some evaluations the *geometric mean* of the average precision (*GMAP*) is used instead of the arithmetic mean. This measure, because it multiplies average precision values, emphasizes the impact of queries with low performance. It is defined as

$$GMAP = \exp \frac{1}{n} \sum_{n} \log AP_n$$

where n is the number of queries, and AP_i is the average precision for query i.

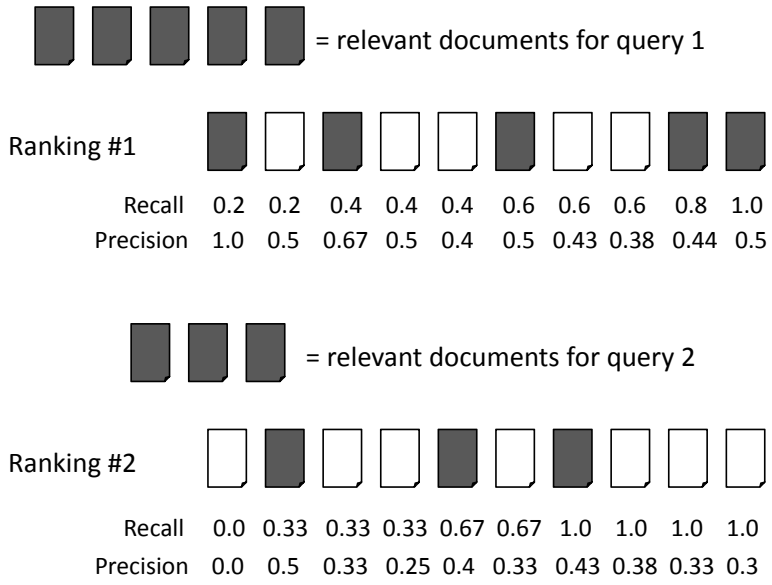

Fig. 8.3. Recall and precision values for rankings from two different queries.

For the example in Figure 8.3, the mean average precision is calculated as follows:

average precision query 1 $= (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$
average precision query 2 $= (0.5 + 0.4 + 0.43)/3 = 0.44$
mean average precision $= (0.62 + 0.44)/2 = 0.53$

The MAP measure provides a very succinct summary of the effectiveness of a ranking algorithm over many queries. Although this is often useful, sometimes too much information is lost in this process. *Recall-precision graphs*, and the tables of recall-precision values they are based on, give more detail on the effectiveness of the ranking algorithm at different recall levels. Figure 8.4 shows the recall-precision graph for the two queries in the example from Figure 8.3. Graphs for individual queries have very different shapes and are difficult to compare. To generate a recall-precision graph that summarizes effectiveness over all the queries, the recall-precision values in Figure 8.3 should be averaged. To simplify the averaging process, the recall-precision values for each query are converted to precision

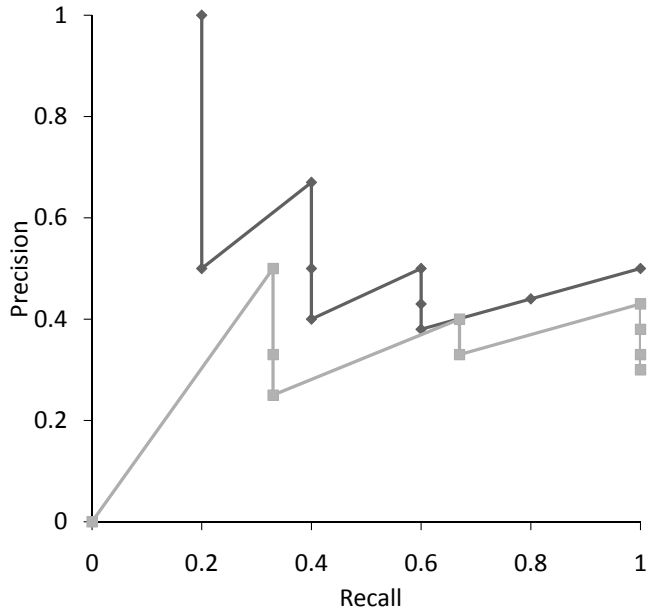

Fig. 8.4. Recall-precision graphs for two queries.

values at standard recall levels, as mentioned in the last section. The precision values for all queries at each standard recall level can then be averaged[8].

The standard recall levels are 0.0 to 1.0 in increments of 0.1. To obtain precision values for each query at these recall levels, the recall-precision data points, such as those in Figure 8.3 must be interpolated. That is, we have to define a function based on those data points that has a value at each standard recall level. There are many ways of doing interpolation, but only one method has been used in information retrieval evaluations since the 1970s. In this method, we define the precision P at any standard recall level R as:

$$P(R) = \max\{P' : R' \geq R \wedge (R', P') \in S\}$$

[8] This is called a *macroaverage* in the literature. A macroaverage computes the measure of interest for each query and then averages these measures. A *microaverage* combines all the applicable data points from every query and computes the measure from the combined data. For example, a microaverage precision at rank 5 would be calculated as $\sum_{q=1}^{n} r_q/5n$, where r_i is the number of relevant documents retrieved in the top 5 documents by query i, and n is the number of queries. Macroaveraging is used in most retrieval evaluations.

where S is the set of observed (R,P) points. This interpolation, which defines the precision at any recall level as the maximum precision observed in any recall-precision point at a higher recall level, produces a step function as shown in Figure 8.5.

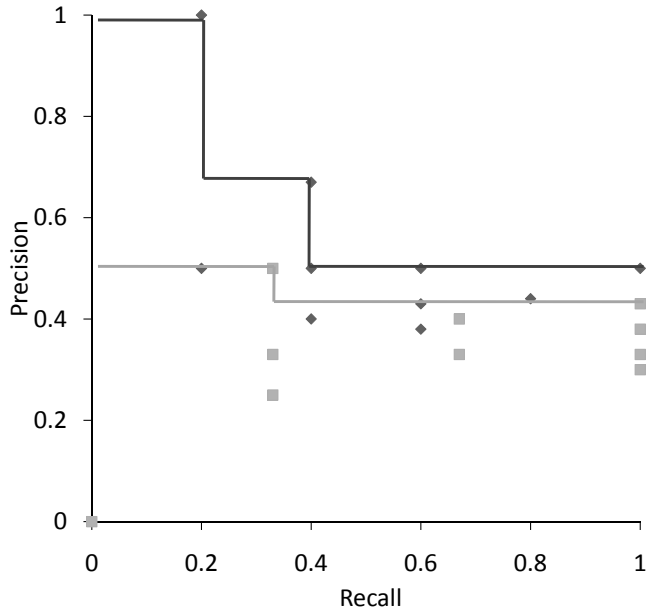

Fig. 8.5. Interpolated recall-precision graphs for two queries.

Because search engines are imperfect and nearly always retrieve some non-relevant documents, precision tends to decrease with increasing recall (although this is not always true, as is shown in Figure 8.4). This interpolation method is consistent with this observation in that it produces a function that is monotonically decreasing. This means that precision values always go down (or stay the same) with increasing recall. The interpolation also defines a precision value for the recall level of 0.0, which would not be obvious otherwise! The general intuition behind this interpolation is that the recall-precision values are defined by the sets of documents in the ranking with the best possible precision values. In query 1, for example, there are three sets of documents that would be the best possible for the user to look at in terms of finding the highest proportion of relevant documents.

The average precision values at the standard recall levels are calculated by simply averaging the precision values for each query. Table 8.4 shows the interpolated precision values for the two example queries, along with the average precision values. The resulting average recall-precision graph is shown in Figure 8.6.

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Ranking 1	1.0	1.0	1.0	0.67	0.67	0.5	0.5	0.5	0.5	0.5	0.5
Ranking 2	0.5	0.5	0.5	0.5	0.43	0.43	0.43	0.43	0.43	0.43	0.43
Average	0.75	0.75	0.75	0.59	0.47	0.47	0.47	0.47	0.47	0.47	0.47

Table 8.4. Precision values at standard recall levels calculated using interpolation.

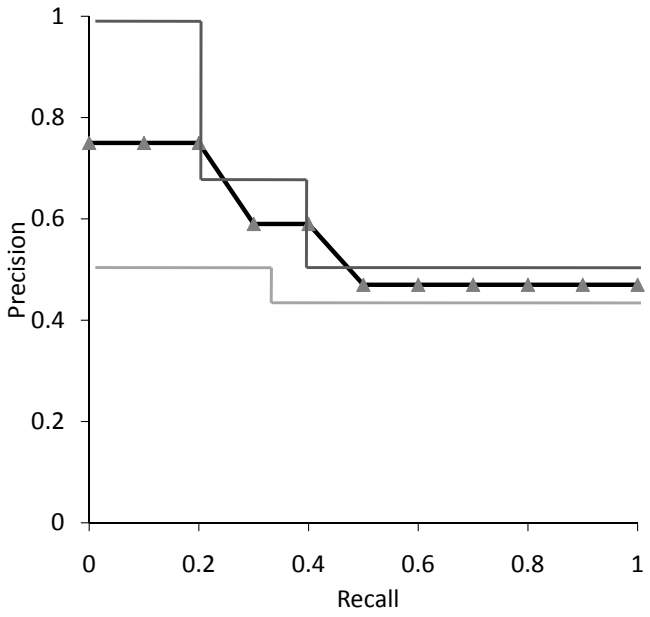

Fig. 8.6. Average recall-precision graph using standard recall levels.

The average recall-precision graph is plotted by simply joining the average precision points at the standard recall levels, rather than using another step function. Although this is somewhat inconsistent with the interpolation method, the intermediate recall levels are never used in evaluation. When graphs are averaged over

many queries, they tend to become smoother. Figure 8.7 shows a typical recall-precision graph from a TREC evaluation using 50 queries.

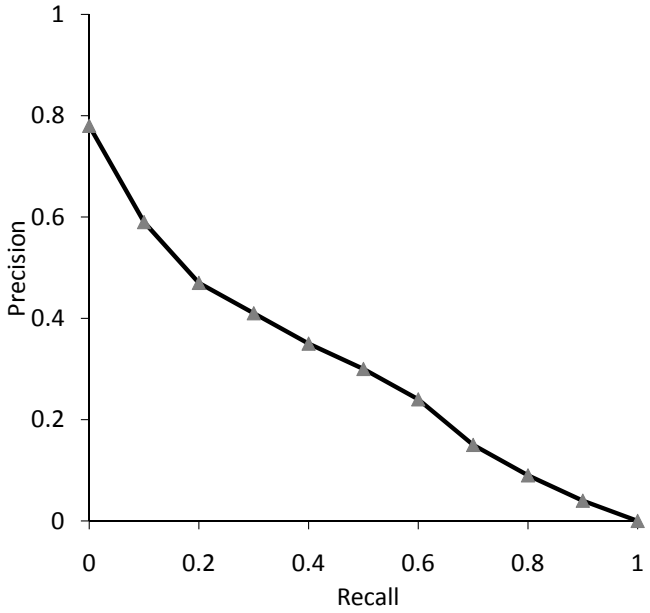

Fig. 8.7. Typical recall-precision graph for 50 queries from TREC.

8.4.3 Focusing On The Top Documents

In many search applications, users tend to look at only the top part of the ranked result list to find relevant documents. In the case of Web search, this means that many users look at just the first page or two of results. In addition, tasks such as *navigational* search (Chapter 7) or *question answering* (Chapter 1) have just a single relevant document. In these situations, recall is not an appropriate measure. Instead, the focus of an effectiveness measure should be on how well the search engine does at retrieving relevant documents at very high ranks (i.e. close to the top of the ranking).

One measure with this property that has already been mentioned is *precision at rank p*, where *p* in this case will typically be 10. This measure is easy to compute, can be averaged over queries to produce a single summary number, and is readily understandable. The major disadvantage is that it does not distinguish between

different rankings of a given number of relevant documents. For example, if only one relevant document was retrieved in the top 10, a ranking where that document is in the top position would be the same as a ranking where it was at rank 10, according to the precision measure. Other measures have been proposed that are more sensitive to the rank position.

The *reciprocal rank* measure has been used for applications where there is typically a single relevant document. It is defined as the reciprocal of the rank at which the first relevant document is retrieved. The *mean reciprocal rank (MRR)* is the average of the reciprocal ranks over a set of queries. For example, if the top five documents retrieved for a query were d_n, d_r, d_n, d_n, d_n, where d_n is a non-relevant document and d_r is a relevant document, the reciprocal rank would be $1/2 = 0.5$. Even if more relevant documents had been retrieved, as in the ranking d_n, d_r, d_n, d_r, d_n, the reciprocal rank would still be 0.5. The reciprocal rank is very sensitive to the rank position. It falls from 1.0 to 0.5 from rank 1 to 2, and the ranking d_n, d_n, d_n, d_n, d_r would have a reciprocal rank of $1/5 = 0.2$. The MRR for these two rankings would be $(0.5 + 0.2)/2 = 0.35$.

The *discounted cumulative gain (DCG)* has become a popular measure for evaluating Web search and related applications (Järvelin & Kekäläinen, 2002). It is based on two assumptions:

1. Highly relevant documents are more useful than marginally relevant documents.
2. The lower the ranked position of a relevant document (i.e. further down the ranked list), the less useful it is for the user, since it is less likely to be examined.

These two assumptions lead to an evaluation that uses graded relevance as a measure of the usefulness or *gain* from examining a document. The gain is accumulated starting at the top of the ranking and may be reduced or *discounted* at lower ranks. The DCG is the total gain accumulated at a particular rank p. Specifically, it is defined as:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

where rel_i is the graded relevance level of the document retrieved at rank i. For example, Web search evaluations have been reported that used manual relevance judgments on a six point scale ranging from "Bad" to "Perfect" (i.e. $0 \leq rel_i \leq 5$). Binary relevance judgments can also be used, in which case rel_i would be either 0 or 1.

The denominator $\log_2 i$ is the discount or reduction factor that is applied to the gain. There is no theoretical justification for using this particular discount factor, although it does provide a relatively smooth (gradual) reduction[9]. By varying the base of the logarithm, the discount can be made sharper or smoother. With base 2, the discount at rank 4 is 1/2 and at rank 8, it is 1/3. As an example, consider the following ranking where each number is a relevance level on the scale 0-3 (not relevant-highly relevant):

3, 2, 3, 0, 0, 1, 2, 2, 3, 0

These numbers represent the gain at each rank. The discounted gain would be:

3, 2/1, 3/1.59, 0, 0, 1/2.59, 2/2.81, 2/3, 3/3.17, 0 =
3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0

The DCG at each rank is formed by accumulating these numbers, giving:

3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61

Similar to precision at rank p, specific values of p are chosen for the evaluation and the DCG numbers are averaged across a set of queries. Since the focus of this measure is on the top ranks, these values are typically small, such as 5 and 10. For this example, DCG at rank 5 is 6.89 and at rank 10 is 9.61. To facilitate averaging across queries with different numbers of relevant documents, these numbers can be normalized by comparing the DCG at each rank with the DCG value for the *perfect* ranking for that query. For example, if the ranking above contained all the relevant documents for that query, the perfect ranking would have gain values at each rank of:

3, 3, 3, 2, 2, 2, 1, 0, 0, 0

which would give *ideal* DCG values of

3, 6, 7.89, 8.89, 9.75, 10.52, 10.88, 10.88, 10.88, 10.88

[9] In some publications, DCG is defined as

$$DCG_p = \sum_{i=1}^{p} (2^{rel_i} - 1)/log(1+i)$$

For binary relevance judgments, the two definitions are the same, but for graded relevance this definition puts a strong emphasis on retrieving highly relevant documents. This version of the measure is used by some search engine companies and, because of this, may become the standard.

Normalizing the actual DCG values by dividing by the ideal values gives us the *normalized discounted cumulative gain (NDCG)* values:

1, 0.83, 0.87, 0.76, 0.71, 0.69, 0.73, 0.8, 0.88, 0.88

Note that the NDCG measure is ≤ 1 at any rank position. To summarize, the NDCG for a given query can be defined as:

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

where IDCG is the ideal DCG value for that query.

8.4.4 Using Preferences

In section 8.3, we discussed how user preferences can be inferred from query logs. Preferences have been used for training ranking algorithms, and have been suggested as an alternative to relevance judgments for evaluation. Currently, however, there is no standard effectiveness measure based on preferences.

In general, two rankings described using preferences can be compared using the *Kendall tau coefficient* (τ). If P is the number of preferences that agree and Q is the number that disagree, Kendall's τ is defined as:

$$\tau = \frac{P - Q}{P + Q}$$

This measure varies between 1 when all preferences agree, to -1 if they all disagree. If preferences are derived from clickthrough data, however, only a partial ranking is available. Experimental evidence shows that this partial information can be used to learn effective ranking algorithms, which suggests that effectiveness can be measured this way. Instead of using the complete set of preferences to calculate P and Q, a new ranking would be evaluated by comparing it to the *known* set of preferences. For example, if there were 15 preferences learned from clickthrough data, and a ranking agreed with 10 of these, the τ measure would be $(10 - 5)/15 = .33$. Although this seems reasonable, no studies are available that show that this effectiveness measure is useful for comparing systems.

In the case of preferences derived from binary relevance judgments, the BPREF[10] measure has been shown to be robust with partial information and to give similar results (in terms of system comparisons) to recall-precision measures such as

[10] Binary Preference

MAP. In this measure, the number of relevant and non-relevant documents are balanced to facilitate averaging across queries. For a query with R relevant documents, only the first R non-relevant documents are considered. This is equivalent to using $R \times R$ preferences (all relevant documents are preferred to all non-relevant documents). Given this, the measure is defined as:

$$BPREF = \frac{1}{R} \sum_{d_r} (1 - \frac{N_{d_r}}{R})$$

where d_r is a relevant document, and N_{d_r} gives the number of non-relevant documents (from the set of R non-relevant that are considered) that are ranked higher than d_r. If this is expressed in terms of preferences, N_{d_r} is actually a method for counting the number of preferences that disagree (for binary relevance judgments). Since $R \times R$ is the number of preferences being considered, an alternative definition of BPREF is:

$$BPREF = \frac{P}{P + Q}$$

which means it is very similar to Kendall's τ. The main difference is that BPREF varies between 0 and 1. Given that BPREF is a useful effectiveness measure, this suggests that the same measure or τ could be used with preferences associated with graded relevance.

8.5 Efficiency Metrics

Compared to effectiveness, the efficiency of a search system seems like it should be easier to quantify. Most of what we care about can be measured automatically with a timer instead of with costly relevance judgments. However, like effectiveness, it is important to determine exactly what aspects of efficiency we want to measure.

The most commonly quoted efficiency metric is query throughput, measured in queries processed per second. Throughput numbers are only comparable for the same collection and queries processed on the same hardware, although rough comparisons can be made between runs on similar hardware. As a single-number metric of efficiency, throughput is good because it is intuitive, and mirrors the common problems we want to solve with efficiency numbers. A real system user will want to use throughput numbers for capacity planning, to help determine if more hardware is necessary to handle a particular query load. Since it is simple to measure the number of queries per second currently being issued to a service, it is

Metric name	Description
Elapsed indexing time	Measures the amount of time necessary to build a document index on a particular system.
Indexing processor time	Measures the CPU seconds used in building a document index. This is similar to elapsed time, but does not count time waiting for I/O or speed gains from parallelism.
Query throughput	Number of queries processed per second.
Query latency	The amount of time a user must wait after issuing a query before receiving a response, measured in milliseconds. This can be measured using the mean, but is often more instructive when used with the median or a percentile bound.
Indexing temporary space	Amount of temporary disk space used while creating an index.
Index size	Amount of storage necessary to store the index files.

Table 8.5. Definitions of some important efficiency metrics

easy to determine if a system's query throughput is adequate to handle the needs of an existing service.

The trouble with using throughput alone is that it does not capture latency. Latency measures the elapsed time the system takes between when the user issues a query and when the system delivers its response. Psychology research suggests that users consider any operation that takes less than about 150 milliseconds to be instantaneous. Above that level, users react very negatively to the delay they perceive.

This brings us back to throughput, because latency and throughput are not orthogonal: generally we can improve throughput by increasing latency, and reducing latency leads to poorer throughput. To see why this is so, think of the difference between having a personal chef and ordering food at a restaurant. The personal chef prepares your food with the lowest possible latency, since she has no other demands on her time and focuses completely on preparing your food. Unfortunately, the personal chef is low throughput, since her focus only on you leads to idle time when she is not completely occupied. The restaurant is a high throughput operation with lots of chefs working on many different orders simultaneously. Having many orders and many chefs leads to certain economies of scale, for instance when a single chef prepares many identical orders at the same time. Note that the chef is able to process these orders simultaneously precisely because

some latency has been added to some orders: instead of starting to cook immediately upon receiving an order, the chef may decide to wait a few minutes to see if anyone else orders the same thing. The result is that the chefs are able to cook food with high throughput but at some cost in latency.

Query processing works the same way. It is possible to build a system that handles just one query at a time, devoting all resources to the current query, just like the personal chef devotes all her time to a single customer. This kind of system is low throughput, because only one query is processed at a time, which leads to idle resources. The radical opposite approach is to process queries in large batches. The system can then reorder the incoming queries so that queries that use common subexpressions are evaluated at the same time, saving valuable execution time. However, interactive users will hate waiting for their query batch to complete.

Like recall and precision in effectiveness, low latency and high throughput are both desirable properties of a retrieval system, but they are in conflict with each other and cannot both be maximized at the same time. In a real system, query throughput is not a variable but a requirement: the system needs to handle every query the users submit. The two remaining variables are latency (how long the users will have to wait for a response) and hardware cost (how many processors will be applied to the search problem). A common way to talk about latency is with percentile bounds, such as "99% of all queries will complete in under 100 milliseconds." System designers can then add hardware until this requirement is met.

Query throughput and latency are the most visible system efficiency metrics, but we should also consider the costs of indexing. For instance, given enough time and space, it is possible to cache every possible query of a particular length. A system that did this would have excellent query throughput and query latency, but at the cost of enormous storage and indexing costs. Therefore, we need to also measure the size of the index structures and the time necessary to create them. Because indexing is often a distributed process, we need to know both the total amount of processor time used during indexing and the elapsed time. Since the process of inversion often requires temporary storage, it is interesting to measure the amount of temporary storage used.

8.6 Training, Testing, and Statistics

8.6.1 Significance Tests

Retrieval experiments generate data, such as average precision values or NDCG values. In order to decide whether this data shows that there is a meaningful difference between two retrieval algorithms or search engines, *significance tests* are needed. Every significance test is based on a *null hypothesis*. In the case of a typical retrieval experiment, we are comparing the value of an effectiveness measure for rankings produced by two retrieval algorithms. The null hypothesis is that there is no difference in effectiveness between the two retrieval algorithms. The *alternative hypothesis* is that there is a difference. In fact, given two retrieval algorithms A and B, where A is a *baseline* algorithm and B is a new algorithm, we are usually trying to show that the effectiveness of B is better than A, rather than simply finding a difference. Since the rankings that are compared are based on the same set of queries for both retrieval algorithms, this is known as a *matched pair* experiment.

A significance test enables us to reject the null hypothesis in favor of the alternative hypothesis (i.e. show that B is better than A) on the basis of the data from the retrieval experiments. Otherwise, we say that the null hypothesis cannot be rejected (i.e. B might not be better than A). As with any binary decision process, a significance test can make two types of error. A Type I error is when the null hypothesis is rejected when it is true. A Type II error is when the null hypothesis is accepted when it is false[11]. Significance tests are often described by their *power*, which is the probability that the test will reject the null hypothesis correctly (i.e. decide that B is better than A). In other words, a test with high power will reduce the chance of a Type II error. The power of a test can also be increased by increasing the sample size, which in this case is the number of queries in the experiment.

The procedure for comparing two retrieval algorithms using a particular set of queries and a significance test is as follows:

1. Compute the effectiveness measure for every query for both rankings.
2. Compute a *test statistic* based on a comparison of the effectiveness measures for each query. The test statistic depends on the significance test, and is simply a quantity calculated from the sample data that is used to decide whether or not the null hypothesis should be rejected.

[11] Compare to the discussion of errors in section 8.3.1.

3. The test statistic is used to compute a *P-value*, which is the probability that a test statistic value at least that extreme could be observed if the null hypothesis were true. Small P-values suggest that the null hypothesis may be false.
4. The null hypothesis (no difference) is rejected in favor of the alternate hypothesis (i.e. B is more effective than A) if the P-value is $\leq \alpha$, the *significance level*. Values for α are small, typically .05 and .1, to minimize Type I errors.

In other words, if the probability of getting a specific test statistic value is very small assuming the null hypothesis is true, we reject that hypothesis and conclude that ranking algorithm B is more effective than the baseline algorithm A.

The computation of the test statistic and the corresponding P-value is usually done using tables or standard statistical software. The significance tests discussed here are also provided in Galago.

The procedure described above is known as a *one-sided* or *one-tailed* test since we want to establish that B is better than A. If we were just trying to establish that there is a difference between B and A, it would be a *two-sided* or *two-tailed* test, and the P-value would be doubled. The "side" and "tail" referred to is the tail of a probability distribution. For example, Figure 8.8 shows a distribution for the possible values of a test statistic assuming the null hypothesis. The shaded part of the distribution is the *region of rejection* for a one-sided test. If a test yielded the test statistic value x, the null hypothesis would be rejected since the probability of getting that value or higher (the P-value) is less than the significance level of 0.05.

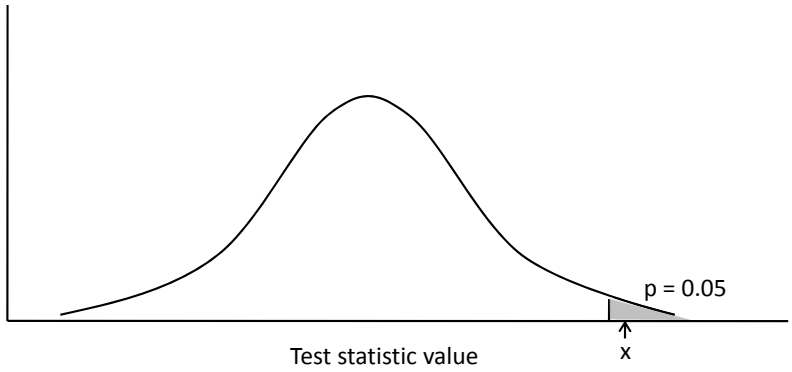

Fig. 8.8. Probability distribution for test statistic values assuming the null hypothesis. The shaded area is the region of rejection for a one-sided test.

The significance tests most commonly used in the evaluation of search engines are the *t-test*[12], the *Wilcoxon signed-rank test*, and the *sign test*. To explain these tests, we will use the data shown in Table 8.6, which shows the effectiveness values of the rankings produced by two retrieval algorithms for 10 queries. The values in the table are artificial and could be average precision or NDCG, for example, on a scale of 0-100 (instead of 0-1). The table also shows the difference in the effectiveness measure between algorithm B and the baseline algorithm A. The small number of queries in this example data is not typical of a retrieval experiment.

Query	A	B	B-A
1	25	35	10
2	43	84	41
3	39	15	-24
4	75	75	0
5	43	68	25
6	15	85	70
7	20	80	60
8	52	50	-2
9	49	58	9
10	50	75	25

Table 8.6. Artificial effectiveness data for two retrieval algorithms (A and B) over 10 queries. The column B-A gives the difference in effectiveness.

In general, the t-test assumes that data values are sampled from normal distributions. In the case of a matched pair experiment, the assumption is that the difference between the effectiveness values is a sample from a normal distribution. The null hypothesis in this case is that the mean of the distribution of differences is zero. The test statistic for the paired t-test is:

$$t = \frac{\overline{B-A}}{\sigma_{B-A}}.\sqrt{N}$$

[12] Also known as Student's t-test, where "student" is the pen name of the inventor, not the type of person who should use it.

where $\overline{B-A}$ is the mean of the differences, σ_{B-A} is the standard deviation[13] of the differences, and N is the size of the sample (the number of queries). For the data in Table 8.6, $\overline{B-A} = 21.4$, $\sigma_{B-A} = 29.1$, and $t = 2.33$. For a one-tailed test, this gives a P-value of 0.02, which would be significant at a level of $\alpha = 0.05$. Therefore for this data, the t-test enables us to reject the null hypothesis and conclude that ranking algorithm B is more effective than A.

There are two objections that could be made to using the t-test in search evaluations. The first is that the assumption that the data is sampled from normal distributions is generally not appropriate for effectiveness measures, although the distribution of differences can resemble a normal distribution for large N. Recent experimental results have supported the validity of the t-test by showing that it produces very similar results to the *randomization test* on TREC data (Smucker, Allan, & Carterette, 2007). The randomization test does not assume the data comes from normal distributions, and is the most powerful of the *nonparametric* tests[14]. The randomization test is, however, much more expensive to compute than the t-test.

The second objection that could be made is concerned with the level of *measurement* associated with effectiveness measures. The t-test (and the randomization test) assume the the evaluation data is measured on an *interval scale*. This means that the values can be ordered (e.g., an effectiveness of 54 is greater than an effectiveness of 53), and that differences between values are meaningful (e.g., the difference between 80 and 70 is the same as the difference between 20 and 10). Some people have argued that effectiveness measures are an *ordinal scale*, which means that the magnitude of the differences are not significant. The Wilcoxon signed-rank test and the sign test, which are both nonparametric, make less assumptions about the effectiveness measure. As a consequence, they do not use all the information in the data and it can be more difficult to show a significant difference. In other words, if the effectiveness measure did satisfy the conditions for using the t-test, the Wilcoxon and sign tests have less power.

The Wilcoxon signed-rank test assumes that the differences between the effectiveness values for algorithms A and B can be ranked, but the magnitude is not important. This means, for example, that the difference for query 8 in Table

[13] For a set of data values x_i, the standard deviation can be calculated by $\sigma = \sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2/N}$, where $\overline{x}$ is the mean.

[14] A nonparametric test makes less assumptions about the data and the underlying distribution than parametric tests.

8.6 will be ranked first because it is the smallest non-zero absolute value, but the magnitude of 2 is not used directly in the test. The test statistic is:

$$w = \sum_{i=1}^{N} R_i$$

where R_i is a signed-rank, and N is the number of differences $\neq 0$. To compute the signed-ranks, the differences are ordered by their absolute values (increasing), then assigned rank values (ties are assigned the average rank). The rank values are then given the sign of the original difference. The null hypothesis for this test is that the sum of the positive ranks will be the same as the sum of the negative ranks.

For example, the 9 non-zero differences from Table 8.6, in rank order of absolute value, are:

2, 9, 10, 24, 25, 25, 41, 60, 70

The corresponding signed-ranks are:

-1, +2, +3, -4, +5.5, +5.5, +7, +8, +9

Summing these signed-ranks gives a value of $w = 35$. For a one-tailed test, this gives a P-value of approximately .025, which means the null hypothesis can be rejected at a significance level of $\alpha = 0.05$.

The sign test goes further than the Wilcoxon sign-ranks test, and completely ignores the magnitude of the differences. The null hypothesis for this test is that $P(B > A) = P(A > B) = \frac{1}{2}$. In other words, we would expect, over a large sample, that the number of pairs where B is "better" than A would be the same as the number of pairs where A is "better" than B. The test statistic is simply the number of pairs where B > A. The issue for a search evaluation is deciding what difference in the effectiveness measure is "better". We could assume that even small differences in average precision or NDCG, such as .51 compared to .5, are significant. This has the risk of leading to a decision that algorithm B is more effective than A when the difference is, in fact, not noticeable to the users. Instead, an appropriate threshold for the effectiveness measure should be chosen. For example, an old IR rule of thumb is that there has to be at least 5% difference in average precision to be noticeable (10% for the conservative). This would mean that a difference of .51-.5 = .01 would be considered a tie for the sign test. If the effectiveness measure was precision at rank 10, on the other hand, any difference might be considered significant since it would correspond directly to additional relevant documents in the top 10.

For the data in Table 8.6, we will consider any difference to be significant. This means there are 7 pairs out of 10 where B is better than A. The corresponding P-value is .17, which is the chance of observing 7 "successes" in 10 trials where the probability of success is 0.5 (just like flipping a coin). Using the sign test, we cannot reject the null hypothesis. Because so much information from the effectiveness measure is discarded in the sign test, it is more difficult to show a difference and more queries are needed to increase the power of the test. On the other hand, it can be used in addition to the t-test to provide a more user-focused perspective. An algorithm that is significantly more effective according to both the t-test and the sign test, perhaps using different effectiveness measures, is more likely to be noticeably better.

8.6.2 Setting Parameter Values

Nearly every ranking algorithm has *parameters* that can be tuned to improve the effectiveness of the results. For example, BM25 has the parameters k_1, k_2, and b used in term weighting, and query likelihood with Dirichlet smoothing has the parameter μ. Ranking algorithms for Web search can have hundreds of parameters that give the weights for the associated features. The values of these parameters can have a major impact on retrieval effectiveness, and values that give the best effectiveness for one application may not be appropriate for another application, or even for a different database. Not only is choosing the right parameter values important for the performance of a search engine when it is deployed, it is an important part of comparing the effectiveness of two retrieval algorithms. An algorithm which has had the parameters tuned for optimal performance for the test collection may appear to be much more effective than it really is when compared to a baseline algorithm with poor parameter values.

The appropriate method of setting parameters for both maximizing effectiveness and making fair comparisons of algorithms is to use a *training set* and a *test set* of data. The training set is used to learn the best parameter values, and the test set is used for validating these parameter values and comparing ranking algorithms. The training and test sets are two separate test collections of documents, queries and relevance judgments, although they may be created by splitting a single collection. In TREC experiments, for example, the training set is usually documents, queries and relevance judgments from previous years. When there is not a large amount of data available, *cross-validation* can be done by partitioning the data into K subsets. One subset is used for testing and $K - 1$ are used for training. This is

repeated using each of the subsets as a test set, and the best parameter values are averaged across the K runs.

Using training and test sets helps to avoid the problem of *overfitting* (mentioned in Chapter 7), which occurs when the parameter values are tuned to fit a particular set of data too well. If this was the only data that needed to be searched in an application, that would be appropriate, but a much more common situation is that the training data is only a sample of the data that will be encountered when the search engine is deployed. Overfitting will result in a choice of parameter values that do not generalize well to this other data. A symptom of overfitting is that effectiveness on the training set improves but the effectiveness on the test set gets worse.

A fair comparison of two retrieval algorithms would involve getting the best parameter values for both algorithms using the training set, and then using those values with the test set. The effectiveness measures are used to tune the parameter values in multiple retrieval runs on the training data, and for the final comparison, which is a single retrieval run, on the test data. The "cardinal sin" of retrieval experiments, which should be avoided in nearly all situations, is testing on the training data. This typically will artificially boost the measured effectiveness of a retrieval algorithm. It is particularly problematic when one algorithm has been trained in some way using the testing data and the other has not. Although it sounds like an easy problem to avoid, it can sometimes occur in subtle ways in more complex experiments.

Given a training set of data, there a number of techniques for finding the best parameter settings for a particular effectiveness measure. The most common method is simply to explore the space of possible parameter values by *brute force*. This requires a large number of retrieval runs with small variations in parameter values (a *parameter sweep*). Although this could be computationally infeasible for large numbers of parameters, it is guaranteed to find the parameter settings that give the best effectiveness for any given effectiveness measure. The Ranking SVM method described in section 7.6 is an example of a more sophisticated procedure for learning good parameter values efficiently with large numbers of parameters. This method, and similar optimization techniques, will find the best possible parameter values if the function being optimized meets certain conditions[15]. Because many of the effectiveness measures we have described do not meet these conditions, different functions are used for the optimization and the parameter

[15] Specifically, the function should be *convex* (or *concave*). A convex function is a continuous function that satisfies the following constraint: $f(\lambda x_1 + (1 - \lambda)x_2) \leq$

values are not guaranteed to be optimal. This is, however, a very active area of research and new methods for learning parameters are constantly becoming available.

8.7 The Bottom Line

In this chapter, we have presented a number of effectiveness and efficiency measures. At this point, it would be reasonable to ask which of them is the right measure to use. The answer, especially with regard to effectiveness, is that no single measure is the correct one for any search application. Instead, a search engine should be evaluated by a combination of measures that show different aspects of the system's performance. In many settings, *all* of the following measures and tests could be carried out with little additional effort:

- Mean average precision - single number summary, popular measure, pooled relevance judgments.
- Average NDCG - single number summary for each rank level, emphasizes top ranked documents, relevance judgments only needed to a specific rank depth (typically to 10).
- Recall-precision graph - conveys more information than a single number measure, pooled relevance judgments.
- Average precision at rank 10 - emphasizes top ranked documents, easy to understand, relevance judgments limited to top 10.

Using MAP and a recall-precision graph could require more effort in relevance judgments, but this analysis could also be limited to the relevant documents found in the top 10 for the NDCG and precision at 10 measures.

All these evaluations should be done relative to one or more baseline searches. It generally does not make sense to do an effectiveness evaluation without a good baseline, since the effectiveness numbers depend strongly on the particular mix of queries and the database that is used for the test collection. The t-test can be used as the significance test for average precision, NDCG, and precision at 10.

All of the standard evaluation measures and significance tests are available using the NIST program trec_eval[16]. They are also provided as part of Galago.

$\lambda f(x_1) + (1 - \lambda) f(x_2)$, for all λ in [0,1]. A function $f(x)$ is concave if and only if $-f(x)$ is convex.

[16] http://trec.nist.gov/trec_eval

In addition to these evaluations, it is also very useful to present a summary of the number of queries that were improved and the number that were degraded, relative to a baseline. Figure 8.9 gives an example of this summary for a TREC run, where the query numbers are shown as a distribution over various percentage levels of improvement for a specific evaluation measure (usually MAP). Each bar represents the number of queries that were better (or worse) than the baseline by the given percentage. This provides a simple visual summary showing that many more queries were improved than were degraded, and that the improvements were sometimes quite substantial. By setting a threshold on the level of improvement that constitutes "noticeable", the sign test can be used with this data to establish significance.

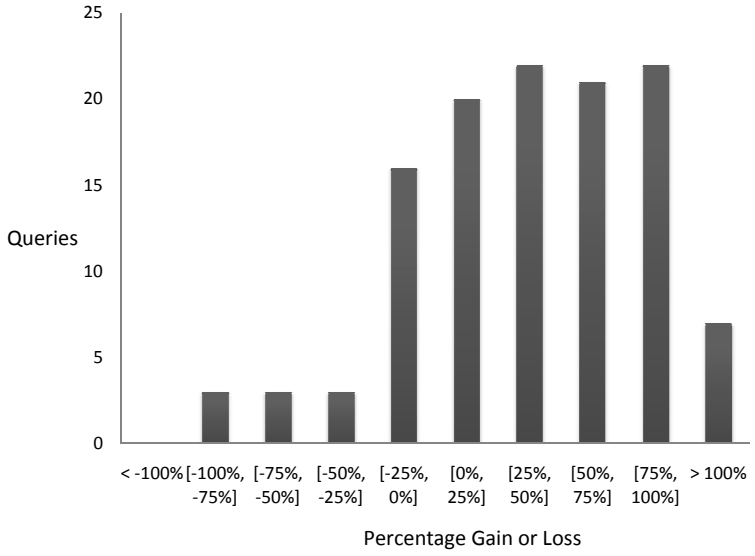

Fig. 8.9. Example distribution of query effectiveness improvements.

Given this range of measures, both developers and users will get a better picture of where the search engine is performing well, and where it may need improvement. It is often necessary to look at individual queries to get a better understanding of what is causing the ranking behavior of a particular algorithm. Query data such as Figure 8.9 can be helpful in identifying interesting queries.

References and Further Reading

Despite being discussed for more than forty years, the measurement of effectiveness in search engines is still a hot topic, with many papers being published in the major conferences every year. The chapter on evaluation in van Rijsbergen (1979) gives a good historical perspective on effectiveness measurement in information retrieval. Another useful general source is the TREC book (Voorhees & Harman, 2005), which describes the test collections and evaluation procedures used and how they evolved.

Saracevic (1975) and Mizzaro (1997) are the best papers for general reviews of the critical topic of relevance. The process of obtaining relevance judgments and the reliability of retrieval experiments are discussed in the TREC book. Zobel (1998) shows that some incompleteness of relevance judgments does not affect experiments, although Buckley and Voorhees (2004) suggest that substantial incompleteness can be a problem. Voorhees and Buckley (2002) discuss the error rates associated with different numbers of queries. Sanderson and Zobel (2005) show how using a significance test can effect the reliability of comparisons and also compare shallow versus in-depth relevance judgments. Carterette et al. (2006) describe a technique for reducing the number of relevance judgments required for reliable comparisons of search engines. Kelly and Teevan (2003) review approaches to acquiring and using implicit relevance information. Fox, Karnawat, Mydland, Dumais, and White (2005) studied implicit measures of relevance in the context of Web search, and Joachims, Granka, Pan, Hembrooke, and Gay (2005) introduced strategies for deriving preferences based on clickthrough data. Agichtein, Brill, Dumais, and Ragno (2006) extended this approach and carried out more experiments introducing click distributions and deviation, and showing that a numbers of features related to user behavior are useful for predicting relevance.

The F measure was originally proposed by van Rijsbergen (1979) in the form of $E = 1-F$. He also provided a justification for the E measure in terms of measurement theory, raised the issue of whether effectiveness measures were interval or ordinal measures, and suggested that the sign and Wilcoxon tests would be appropriate for significance. Cooper (1968) wrote an important early paper that introduced the expected search length (ESL) measure, which was the expected number of documents that a user would have to look at to find a specified number of relevant documents. Although this measure has not been widely used, it was the ancestor of measures such as NDCG (Järvelin & Kekäläinen, 2002) that fo-

cus on the top-ranked documents. Another measure of this type that has recently been introduced is rank-biased precision (Moffat, Webber, & Zobel, 2007).

Yao (1995) provides one of the first discussions of preferences and how they could be used to evaluate a search engine. The paper by Joachims (2002b) that showed how to train a linear feature-based retrieval model using preferences also used Kendall's τ as the effectiveness measure for defining the best ranking. The recent paper by Carterette and Jones (2007) shows how search engines can be evaluated using relevance information directly derived from clickthrough data, rather than converting clickthrough to preferences.

An interesting topic related to effectiveness evaluation is the *prediction* of query effectiveness. Cronen-Townsend, Zhou, and Croft (2006) describe the Clarity measure that is used to predict whether a ranked list for a query has good or bad precision. Other measures have been suggested that have even better correlations with average precision.

There are very few papers that discuss guidelines for efficiency evaluations of search engines. Zobel, Moffat, and Ramamohanarao (1996) is an example from the database literature.

Exercises

8.1. Find 3 other examples of test collections in the information retrieval literature. Describe them and compare their statistics in a table.

8.2. Imagine that you were going to study the effectiveness of a search engine for blogs. Specify the retrieval task(s) for this application, then describe the test collection you would construct, and how you would evaluate your ranking algorithms.

8.3. For one query in the CACM collection, generate a ranking using Galago, then calculate average precision, NDCG at 5 and 10, precision at 10, and the reciprocal rank by hand.

8.4. For two queries in the CACM collection, generate two uninterpolated recall-precision graphs, a table of interpolated precision values at standard recall levels, and the average interpolated recall-precision graph.

8.5. Generate the mean average precision, recall-precision graph, average NDCG at 5 and 10, and precision at 10 for the entire CACM query set.

8.6. Compare the MAP value calculated in the previous problem to the GMAP value. Which queries have the most impact on this value?

8.7. Another measure which has been used in a number of evaluations is *R-precision*. This is defined as the precision at R documents, where R is the number of relevant documents for a query. It is used in situations where there is a large variation in the number of relevant documents per query. Calculate the average R-precision for the CACM query set and compare it to the other measures.

8.8. Generate another set of rankings for for 10 CACM queries by adding structure to the queries manually. Compare the effectiveness of these queries to the simple queries using MAP, NDCG, and precision at 10. Check for significance using the t-test, Wilcoxon test, and the sign test.

8.9. For one query in the CACM collection, generate a ranking and calculate BPREF. Show that the two formulations of BPREF give the same value.