Index Ordering by Query-Independent Measures

Paul Ferguson

A dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University
School of Computing

Supervisor: Prof. Alan F. Smeaton

July 2006

Abstract

There is an ever-increasing amount of data that is being produced from various data sources – this data must then be organised effectively if we hope to search though it. Traditional information retrieval approaches search through all available data in a particular collection in order to find the most suitable results, however, for particularly large collections this may be extremely time consuming.

Our purposed solution to this problem is to only search a limited amount of the collection at query-time, in order to speed this retrieval process up. Although, in doing this we aim to limit the loss in retrieval efficacy (in terms of accuracy of results). The way we aim to do this is to firstly identify the most "important" documents within the collection, and then sort the documents within the collection in order of their "importance" in the collection. In this way we can choose to limit the amount of information to search through by eliminating the documents of lesser importance, which should not only make the search more efficient, but should also limit any loss in retrieval accuracy.

In this thesis we investigate various different query-independent methods that may indicate the importance of a document in a collection. The more accurate the measure is at determining an important document, the more effectively we can eliminate documents from the retrieval process – improving the query-throughput of the system, as well as providing a high level of accuracy in the returned results. The effectiveness of these approaches are evaluated using the datasets provided by the terabyte track at the Text REtreival Conference (TREC).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in technology have made it possible to acquire and store an ever increasing amount of data from a number of heterogeneous sources, such as wireless sensor networks, text, and video. Having amassed these large collections of data, if we then wish to search through these *effectively* and *efficiently* we must take into account the usefulness of the data both at the time of storage as well as the time of retrieval – with a view to the type of retrieval that is required on this data, so that the search system can provide a more effective and efficient service.

Perhaps the most well known of these rapidly expanding collections is the World Wide Web, which has grown dramatically in recent years. However, with the existence of such a huge collection of documents, it also becomes more difficult for a user to find the information that they need, and so users require better searching facilities in order to find useful information. In tandem with the rapid growth of the Web, there has also been rapid growth in search engines such as Google (Google Inc., 2006), which have become tremendously successful due to their ability to effectively search the Web. However, even with the large size of the Web, users still demand rapid responses to their search requests, as well as highly accurate answers – most users do not look at more that the top 5 - 10 results returned by a Web search engine, even though the users themselves generally issue quite vague queries consisting of two words on average (Silverstein et al., 1999).

All this makes rather large demands on the search engine, both to be *effective* and *efficient*. An effective search engine is one that can present relevant documents to a user in response to their search query, while an efficient search engine is one that can make the most out of the system resources available to it, i.e. memory, hard-disk space and CPU cycles. Although the effectiveness and efficiency of a system are often in conflict, as it is difficult to make an increase in one without degrading the other.

Apart from dealing with the sheer size of the Web (estimated at over 11.5 billion pages by Gulli and Signorini (2005)), search engines also have to deal with information coming from a number of diverse sources and due to the lack of constraints and controls on the publishing of information on the Web, understandably the quality of the documents varies considerably. Much of the success of the Google search engine has been attributed to the incorporation of their PageRank formula, which identifies "important" documents in the Web (based on the Web's linkage structure) and so gives these a prioritised ranking. We may view this importance (or popularity) estimation of a document as one indicator of the overall quality of a document, and if we can somehow identify and discriminate between documents of high and low quality with a certain degree of confidence we may be able to leverage this information in order to make considerable gains, by promoting the high quality documents while demoting documents of lower quality. Also if we can identify these high quality documents and then only search these, then not only should we be able to improve the effectiveness of the search, but we should also be able to reduce the amount of processing that is required to answer a user's information need. This would allow potential for faster query-throughput, as well as reducing the hard-disk space necessary to store the collection, if we take the decision to eliminate these lower-quality documents from the search entirely.

In this thesis we investigate a number of different measures (such as PageRank) which may indicate the quality of a document in a query-independent manner, and use these not only to help filter out documents that may not be of particularly high

quality (and so making the search process more efficient), but also promoting the higher quality documents, so that the search process may be made more effective also.

1.1 Thesis Organisation

The thesis is organised as follows:

In Chapter 2 we give a general introduction to an information retrieval system, and describe its main components: document gathering, indexing and retrieval. We then focus on the retrieval component of the system, describing the main retrieval strategies that are used to retrieve documents in response to a user's query, as well as describing the use of linkage analysis in the retrieval process. We then describe how to evaluate the usefulness of an IR system.

Chapter 3 again focuses on the retrieval process of an IR system – firstly giving a detailed example of this process. We then outline related work in the area of "reducing the search space", not only to show what has previously been done in this area but also to highlight the specific area that the work contained within this thesis fits into.

In Chapter 4 we discuss the use of search quality in information retrieval, and introduce a number of measures that may indicate quality in a query-independent manner. We then discuss how we may use these measures to filter out documents by using a sorted inverted index. This chapter also describes the steps necessary for creating such a sorted inverted index.

In Chapter 5 we describe some of the traditional methods for combining sources of evidence together: linear combination, rank fusion, Dempster-Shafer, as well as using support vector machines. We describe these methods as we use them to combine different query-independent sources of evidence and in this chapter we describe specifically how to combine these types of sources.

In Chapter 6 we firstly evaluate a number of early termination methods as a

means of reducing the amount of processing that is done at retrieval time. We then evaluate a number of query-independent measures for their effectiveness in sorting an inverted index. We also assess how best to evaluate the usefulness of a sorted inverted index. We then evaluate the effectiveness of using various types of combination on our query-independent measures. Then finally we take a look at the trade-offs that are being made with our proposed approach, when compared with a conventional retrieval approach.

In Chapter 7 we summarise our results, as well as proposing some extensions and future directions for this work.

Chapter 2

Background

In this chapter we provide the necessary background in the area of information retrieval, that we feel is needed to understand the content of the work in the chapters that are to follow. Firstly we describe an information retrieval system, and its main components. We then concentrate specifically on the area of retrieval – looking at various ways in which the system can choose which documents to return to a user, in order to satisfy their information need. Then finally we look at how an information retrieval system may be evaluated.

2.1 Information Retrieval System

> "*The ultimate search engine would basically understand everything in the world, and it would always give you the right thing. And we're a long, long ways from that.*", Larry Page (2004)

With the huge amount of information currently available to us, as well as the new information that is being created daily, this information should be stored effectively, so that it is capable of being retrieved when required. For centuries humans have communicated using written documents, and today with the emergence of the World Wide Web and digital libraries, the need for effective storage and retrieval of documents has become ever more apparent.

In order to manage the storage and retrieval of these *documents*, an *Information Retrieval System* is needed. This Information Retrieval (IR) system consists of three main components: a *Document Gatherer*, an *Indexer*, and a *Retrieval* component, each of which are shown in Figure 2.1 and are discussed in this section.

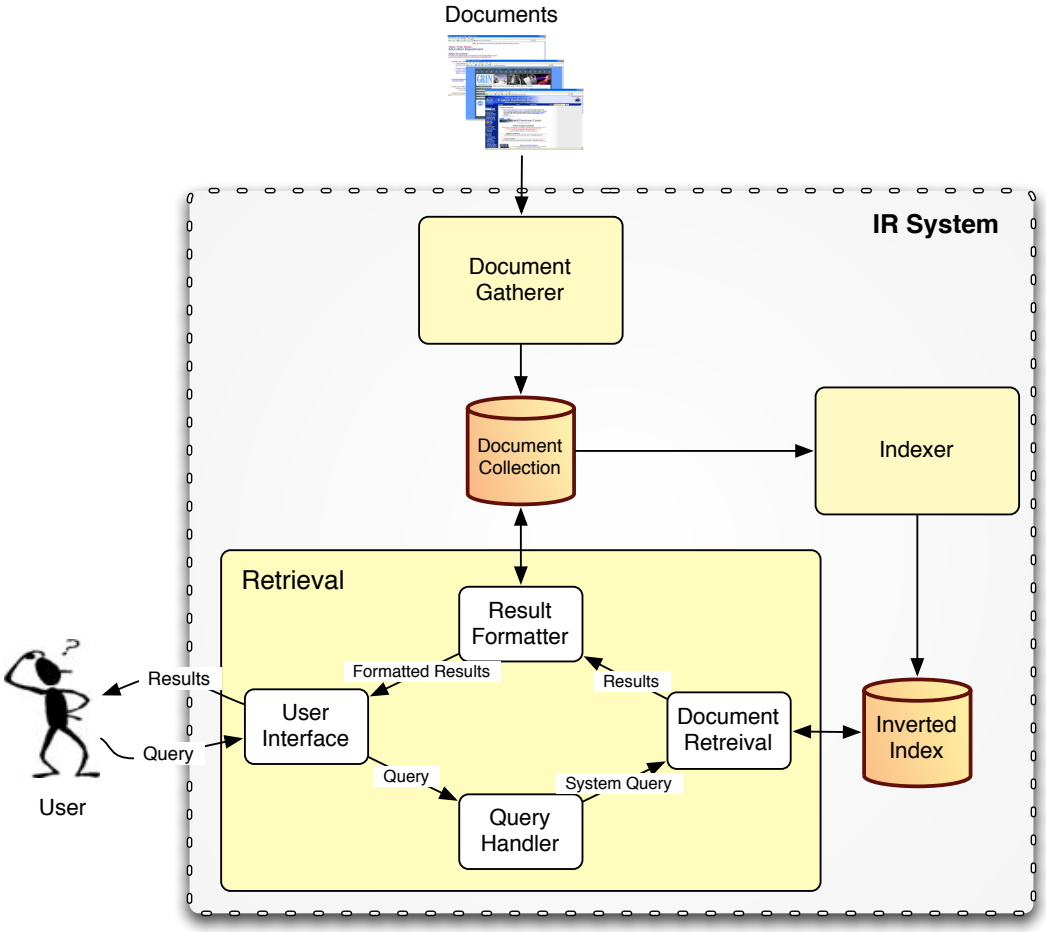


Figure 2.1: Basic IR System

2.1.1 Document Gathering

Document gathering (or document crawling, in the context of the Web) involves gathering together the documents that are subsequently searched by the IR system. For large search systems, such as a web search engine, this would involve the *crawling* of web documents; following the links from one page to the next and downloading the required documents into a central location. For other types of documents, this would

require other forms of acquisition. For example books or handwritten documents may be digitised by scanning, and by using Optical Character Recognition (OCR) techniques, words within the documents may be identified. Other documents such as emails may be gathered from a central mail server for example.

There are also many issues when dealing with a constantly changing collection of documents such as the Web, for instance, consideration would need to be given to how often to download each document, in order to keep the documents up to date. A knowledge of the rate of change of these documents allows the crawler to estimate how often to visit each document. Some document crawlers (*focused crawlers*) aim to only download documents relating to specific topics, or are personalised for a particular person, or group of people (Chakrabarti et al., 1999).

2.1.2 Indexing

In order to facilitate fast searching on a large collection of documents, in response to a user's query, it is necessary to store the data associated with each of the documents in specialised data structures. As the documents are to be retrieved in response to a query, we wish to be able to quickly find the location of all occurrences of that query term (or terms) in the collection. Although there are many ways to achieve this, "in applications involving text, the single most suitable structure is an *inverted index*", as described by Witten et al. (1999). The index supplies information about the terms that appear in each document, as well as the number of times they appear. The index may store additional information, such as the position that the terms occur at, which may be useful to the IR system, depending on the type of retrieval that the system is to provide (which will be discussed further in this section).

2.1.2.1 Inverted Index

The conventional inverted index structure consists of a *lexicon*, which stores all the indexed terms in the collection, as well as additional information about these terms. The lexicon contains the location of each term's *postings list,* which holds the

location (or document number, d) of each occurrence of that term in the collection, along with the count of how many times the term occurs in that document, $f_{d,t}$. The lexicon itself may also hold information regarding each of the terms that are indexed, such as the number of documents that a particular term occurs in. Figure 2.2 illustrates a basic inverted index structure.

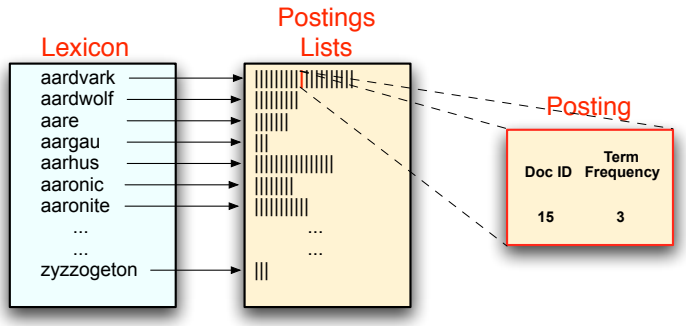


Figure 2.2: Basic inverted index structure, which contains a lexicon as well as a postings list for each term in the lexicon

For each postings list there is a list of postings <d, $f_{d,t}$>, for example the postings list for a given term might look as follows:

```
<15, 4> <17, 1> <104, 2>
```

meaning that the searched term occurs in document 15 four times, in document 17 once and in document 104 twice.

This type of index organisation is sufficient to process both Boolean queries (which provide a set of results in response to a query, based on a binary relevance assessment) and ranked queries (which provide a ranked list of results in response to a query, in order of the document's similarity to the query), both of which will be discussed in greater detail in section 2.2. In order to support phrase queries (where the user is searching for the occurrence of an exact phrase, for example "to be or not to be"), for each posting it is also necessary to store the position(s) within each document where the term occurs. Expanding the previous example of an inverted index to include positional information, i.e. the position within the document of each term occurrence, the posting list might look like the following:

```
<15, 4; 10, 15, 35, 80> <17, 1; 38> <104, 2; 19, 57>
```

This additional information tells us that the term occurs in document 15 four times at word offset positions 10, 15, 35 and 80.

These postings lists can then be optimised further: as the document numbers will be processed sequentially from the beginning of the file, the list can be stored as the initial position, followed by a list of *d-gaps* or *run-lengths* in document identifiers (Witten et al., 1999). This generally results in smaller integers, which are more suitable for compression using schemes such as those of Elias (1975) or Golomb (1966). This would result in the previous example (without positional information) being stored as follows:

```
<15, 4> <2, 1> <87, 2>
```

As this list will be processed sequentially from disk, the original values can be found by summing the previous values. For example the sum of 15 and 2 gives back the original value of 17. All this has a significant positive effect on the compressibility of the inverted index, meaning that the time to read each postings list from disk is reduced.

2.1.3 Retrieval

Having represented the information contained with the documents in the form of an inverted index, we now look at how this information is then accessed in response to a user request for information.

There are many different ways in which a user may specify their *information need*. This is done most often by specifying a query, which usually consists of a few words that the user believes best encompasses their need. Depending on the application, the process for inputting this information into the IR system can differ. For example Figure 2.3 shows the Google web search engine interface (Google Inc.,

2006), which provides the option for "Advanced Search", "Preference", as well as specifying the type of content, web, images, etc, yet the main component is the facility to input a user specified query. Figure 2.4 shows the user interface for the the search facility on the Windows XP operating system, to search for files and folders contained on the system. Again, as well as extra options allowing the filtering of results based on certain criteria, we can also see that the main input is in the form of text input, to specify the file name of the document being sought and/or text within that document.

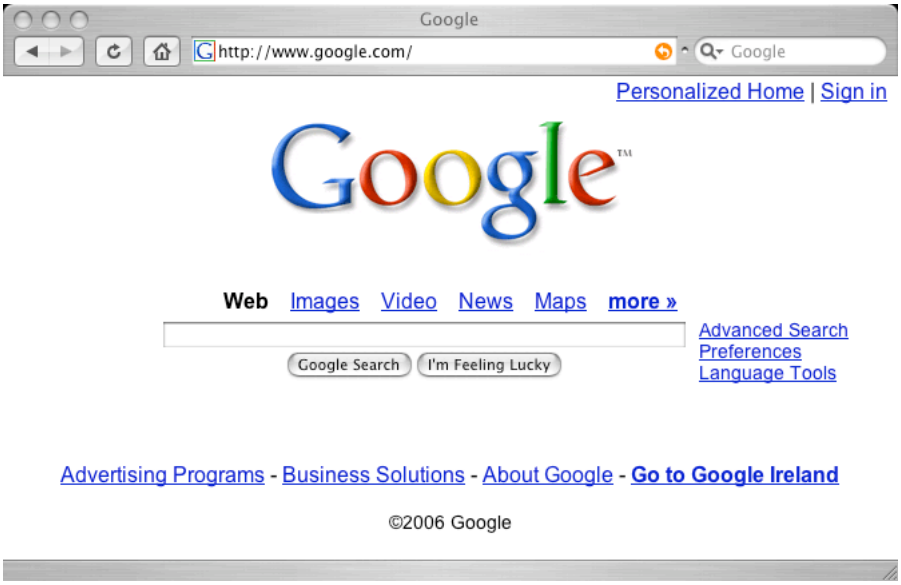


Figure 2.3: Google user interface

It is then the job of the search system to take this user specified information as input and to provide the user with the information that they require. Depending on the system being used, as well as the type of information that is being searched, the ambiguity associated with the user's information need may vary. For example for a web search, a query consisting of only one or two keywords may be very vague and may relate to many distinct topics (many of which may not be of relevance to the user). This may pose difficulties to the IR system, whose responsibility it is to decide which of the documents that are available to it (if any), are the most likely to satisfy the user's need. In section 2.2 we look at different ways in which an IR

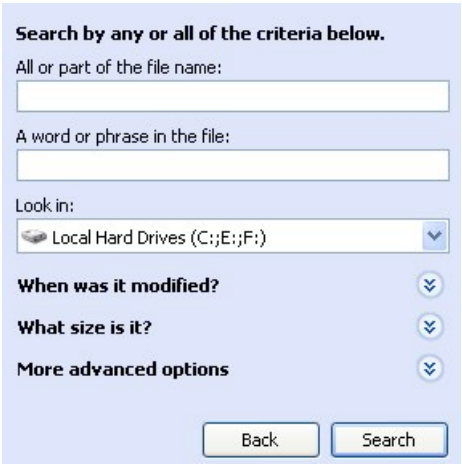


Figure 2.4: Windows XP file search

system may choose which documents to return to a user, in response to a specified query.

Once this list of documents has been chosen by the system (depending on the type of system), it may return the actual documents, or simply a reference to the location of the documents, and present these to the user. As shown in Figure 2.1, there may also be a feedback mechanism, which allows the user to refine their search, based on the information that has been presented to them.

2.2 Retrieval Strategies

In order to satisfy a user's information need, a search engine must supply the user with a list of the most relevant documents that can be found in its collection. In order to supply this list there must be some process by which the relevance of a document can be assessed between the documents in the collection and the query itself.

In order to supply each document in the collection with a degree of similarity to a query (or simply a document score), a retrieval model is employed. This defines how the relevance between a query and a document is to be assessed, and so allowing the retrieval of documents that the model deems to be likely to be relevant to the

query.

In this section we shall outline some of the classic models that have been used in information retrieval: *Boolean*, *Vector-Space* and *Probabilistic*. Each of these models define relevance based on the term distribution within documents, however other evidence such as the links between documents may also be considered for assessing potential relevance for certain applications (this will be discussed in section 4.2.1).

2.2.1 The Boolean Model

The Boolean model of document retrieval is based on set theory and Boolean algebra. Queries are formulated by combining query terms with AND, OR and NOT logic. Using this approach, there is no sense of a partial match between the query and document: a document is either a match or not, and is so known as an *exact match* technique. Figure 2.5 shows graphically how documents are matched to the Boolean query "(tourism AND information AND Prague)".

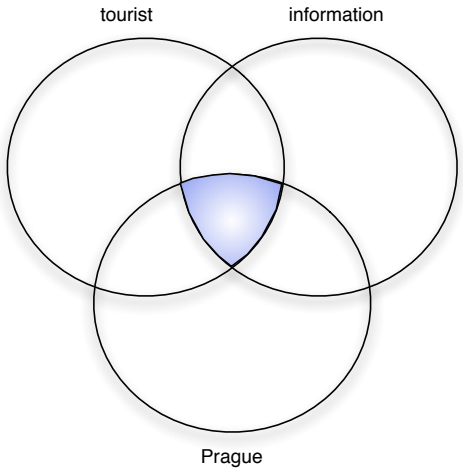


Figure 2.5: Boolean query "(tourism AND information AND Prague)"

Although for many years most commercial systems used a Boolean model of retrieval, the model does suffer from the absence of ranking of documents. This can be problematic, particularly when dealing with large collections of documents, due to the problems associated with navigation through large sets of retrieved documents

(Baeza-Yates and Ribeiro-Neto, 1999). Boolean search can be advantageous, in that it can successfully use very restrictive search, and can often be used more effectively by an experienced user (Cleverdon, 1984). However the complex query formulation poses a difficultly to novice users, who generally prefer the use of natural language queries.

2.2.2 The Vector-Space Model

The Vector-Space model, proposed by Salton et al. (1975), represents queries and documents as high dimensional vectors, with an orthogonal dimension for each term in the collection. The model provides the general retrieval framework, that requires a choice of term weighting scheme, as well as a similarity function to calculate the distance between the vector representations.

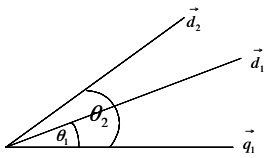


Figure 2.6: Vector-Space Model.

The most basic term vector representation is known as the binary vector model, which identifies a term as being present or not, i.e. binary [1,0]. The document representation can be extended to include document and term statistics, such as information regarding the frequency of occurrence of terms within a document (*term frequency, tf*) (Salton, 1968). A term may also provide a means for discrimination, based on the fact that commonly occurring terms are less likely to provide meaningful retrieval information. A frequently used measure based on this is the *inverse document frequency* (*idf*) (Sparck Jones, 1972). Using these *tf* and *idf* measures, the weight of a term in a document can be defined as:

$$w_{t,d} = tf_{t,d} \times idf_t \tag{2.1}$$

where idf_t is the inverse document frequency of a term t:

$$idf_t = log\left(\frac{N}{n_t}\right) \tag{2.2}$$

where N is the total number of documents in the collection, and n_t is the number of documents in the collection that contain the term t.

Work was done subsequently on improving upon this basic combination of the primary *tf-idf* weights (as shown in equation (2.1)) by Salton and Buckley (1988). This resulted in an improved weighting scheme, which did not allow single matching terms with a high term frequency to skew the results over other term matches, as can happen with the basic *tf-idf* formulation (2.1). This weighting is calculated as follows:

$$w_{ij} = \frac{(logtf_{ij} + 1.0) \times idf_j}{\sum_{j=1}^{t}[logtf_{ij} + 1.0) \times idf_j]^2} \tag{2.3}$$

A similar normalisation can also be carried out by incorporating the maximum within-document term frequency, $maxtf$:

$$w_{ij} = \frac{tf_{ij}}{maxtf_{ij}} \times log\left(\frac{N}{df_j}\right) \tag{2.4}$$

Singhal et al. (1996) extended this work further, incorporating *pivoting* to compensate for the favouring of long documents in retrieval

Although there are a number of functions that can measure the distance between a document vector (D) and query vector (Q) (Sparck Jones, 1972), the most commonly used is the Cosine similarity measure,

$$sim(D, Q) = \frac{D \cdot Q}{|D| \times |Q|} \tag{2.5}$$

or:

$$sim(D,Q) = \frac{\sum_{t \in Q} w_{t,D} \times w_{t,Q}}{\sqrt{\sum_{t \in Q} w_{t,D}^2 \times \sum_{t \in Q} w_{t,Q}^2}} \tag{2.6}$$

The main benefit of the vector-space model in comparison to the Boolean model is that it provides a ranking of the results, based on their degree of similarity to the query, as well as not requiring a full match between all query and document terms. This model has proved very effective across a number of test collections and so remains a popular and widely used method for document retrieval.

2.2.3 The Probabilistic Model

The *Probabilistic Retrieval Model* is similar to the vector-space model in its representation of documents and queries as vectors. However, instead of retrieving documents based on their similarities to the query, the probabilistic model retrieves documents based on their probability of relevance to the query. The model was proposed by Maron and Kuhn (1960) and was later extended by Robertson and Sparck Jones (1976). The *Probability Ranking Principle* (Robertson and Sparck Jones, 1976) suggests ranking documents by decreasing probability of relevance between a query Q and document D, $P(R|Q,D)$. We next discuss the use of the Binary Independence Model and Okapi BM25, which are examples of the probabilistic model.

2.2.3.1 Binary Independence Model

Assuming term independence, the probability of relevance for a given document can be calculated by summing its individual term relevance weights. Documents are represented as binary vectors of terms, which are the estimations of the probabilities that the given terms in a query will appear in a relevant document, but not in a non-relevant document. These term probabilities can be estimated from a sample set of documents and queries with corresponding relevance judgments (Robertson and Sparck Jones, 1976; Robertson et al., 1982). This was later revised by Croft and Harper (1979a) so that the model did not include these prior estimates of relevance,

as these are not always available.

2.2.3.2 Okapi BM25

The Okapi BM25 model was proposed by Robertson et al. (1994), and is a more effective probabilistic model than the Binary Independence model. The BM25 model approximates the 2-Poisson model (Harter, 1975; Bookstein and Swanson, 1974). The base BM25 formulation is:

$$BM25(q,d) = \sum_{t_i \in Q} \log \frac{(r_i + 0.5)(N - n_i - R + r_i + 0.5)}{(R - r_i + 0.5)(n_i - r_i + 0.5)} \times \frac{(k_1 + 1)tf_i}{K + tf_i} \times \frac{(k_3 + 1)qtf_i}{k_3 \times qtf_i} \tag{2.7}$$

where

$$K = k_1((1 - b) + b \times dl/avdl) \tag{2.8}$$

Here tf_i represents the term frequency within the document, qtf_i is the term frequency within the query, dl is the document length and $avdl$ is the average document length. R is the number of documents known to be relevant to a topic, and r_i is the number of these containing the term i. The parameters k_1, k_3, b depend on the nature of the queries and the collection, and can be optimised, for example to suit small or large queries.

Here k_1 controls the influence of tf_{ij}: k_1 approaching 0 reduces the influence of the term frequency, while a larger k_1 increases its influence. Likewise k_3 controls the query term weight. The b parameter adjusts the document length normalisation: b approaching 1 increases the document length normalisation, while b = 0 results in no document length normalisation.

For a typical retrieval task of retrieving a list of results in response to a user specified query and ignoring any repetition of terms in the query, as is the case for the vast majority of web queries, this function can be simplified to:

$$bm25(q,d) = \sum_{t \epsilon q} log\left(\frac{N - df_i + 0.5}{df_i + 0.5}\right) \times \frac{(k_1+1)tf_i}{k_1((1-b) + b\frac{dl}{avdl}) + tf_i} \tag{2.9}$$

where df_i is the number of documents in the collection that contain the term i.

The BM25 model has been extensively tested on the Text Retrieval Conference (TREC) test collections (discussed in section 2.3.4), and in general performs very well on these collections. For this reason we will make use of this model to provide a baseline ranking in our experiments in Chapter 6.

2.2.4 Linkage Analysis

Linkage analysis utilises the hypertext structure of the web to provide a ranking for documents, which is independent of their content, and is primarily used in combination with content-based retrieval techniques for web retrieval. Many of the techniques used within linkage analysis have been developed from social network analysis, and have been originally used in information retrieval to promote documents based on the analysis of the citations between documents (Garfield, 1955, 1972). Certain linkage analysis techniques have also been applied to other forms of documents such as emails, where the importance of an email may be deduced, not only by the sender and receiver(s), but also the level of response that it generates, for example the number of replies, as well as other sub-emails that this spawns.

Over the last number of years the more popular web search engines appear to have integrated linkage analysis into the scoring of their systems. Anecdotally at least, this appears to have increased the performance of these systems, although up until recently there has been little scientific evidence in support of better quality results, particularly using the conventional TREC evaluation (Gurrin and Smeaton, 2004).

Linkage analysis is said to exploit the latent human judgement on the web, in the form of the hyperlink structure. The basis for this latent human judgement is

as follows:

- A link between two documents carries an implication of related content.
- The author of one document found the content of another document useful (provided they were created by two different authors)

Linkage analysis techniques utilise these in different ways, in order to provide a measure of importance for each of the documents in the collection.

Networks of interaction have been studied for a long time in social sciences (Wasserman and Faust, 1994), where nodes represent people or organisations and edges represent connections or interactions. Intuitively, increasing the number of connections to a node should increase its "importance". However it would also seem intuitive that measuring the number of incoming links should not be as accurate a measure of prestige as also taking into account the type of nodes that are providing the links, as some nodes offer more prestige than others. Similarly with linkage analysis on the web, these types of consideration are also made: most web based linkage analysis techniques are calculated via an iterative process, which allocates a popularity score to each of the documents and so allowing these scores to be propagated to any documents that they are linked to. Although many linkage analysis techniques have been introduced, we choose to discuss two of the most popular and well understood approaches: *PageRank* and *HITS*.

2.2.4.1 PageRank

PageRank is one of the best known linkage analysis techniques, popularised by the Google search engine (Google Inc., 2006) and is a query-independent method. The description of the model has been compared to a "random surfer" who is given a web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank, and the *damping factor* is the probability at

each page the "random surfer" will get bored and request another random page, (Page et al., 1998).

PageRank is calculated as follows: for a set of documents (S) that link to a particular document (d), the PageRank of d is the combined PageRank of every document in the set S, divided by the number of outlinks (*outdegree*) from each document in S. PageRank is then calculated over a number of iterations: Page et al. (1998) report acceptable convergence ranks in 52 iterations for a crawl of 322 million links, while convergence on half that data takes roughly 45 iterations. To calculate PageRank, firstly all documents are assigned an initial PageRank score PR_n, and we then calculate a simple PageRank score for each document as follows:

$$PR'_n = c \cdot \sum_{m \epsilon S_n} \frac{PR_m}{outdegree_m} \tag{2.10}$$

where c is a constant that is maximised and S_n is the set of documents that link into document n.

After calculating a PageRank value for each document we store the new value PR'_n as the value PR_n and continue this process until convergence occurs. However this initial PageRank formula is susceptible to certain problems, such as *dangling links* and *rank sinks*, which do not allow their scores to be propagated back into the rest of the linkage graph. These problems can be overcome by introducing a vector $\overrightarrow{E}$, which receives a link from all nodes in the graph, thereby ensuring that their weight can be distributed back into the graph, as illustrated in figure 2.7. The reasoning behind this is that a web surfer will traverse through the web by following links, but will eventually become bored and want to jump from their current document to some other location.

The weight accumulated in the $\overrightarrow{E}$ vector is usually distributed equally across all nodes in the graph, however by tailoring the distribution of weights we can create a personalised PageRank, based on a user's preference. This can be introduced into the PageRank calculation as a preference vector, and so changes the browsing

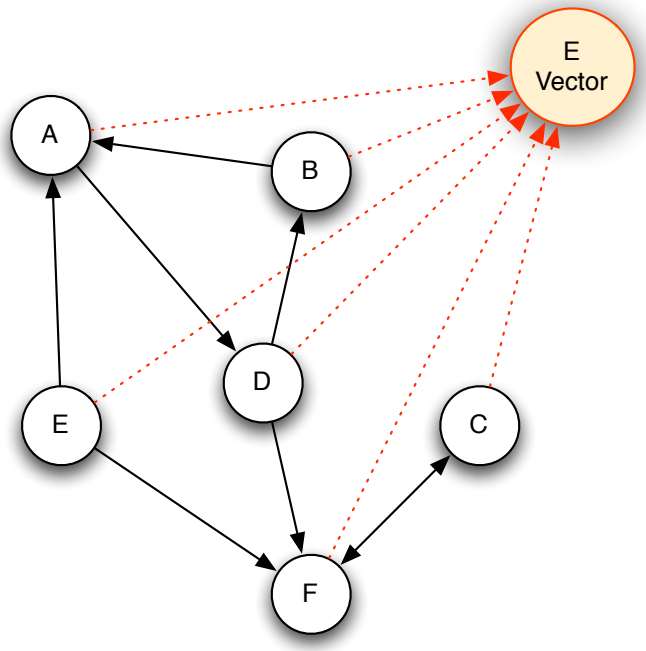


Figure 2.7: The introduction of an E Vector

behavior of the "random surfer" to be more like a specific user, or group of users (Page et al., 1998; Haveliwala, 2002, 2003). The calculation of PageRank, including the $\overrightarrow{E}$ vector is calculated as follows:

$$PR'_n = c \cdot \sum_{m \epsilon S_n} \frac{PR_m}{outdegree_m} + c(E(m)) \tag{2.11}$$

where E(m) is the value of the E vector that is to be be distributed back to document m.

The main benefit of PageRank is that its calculation occurs prior to query execution and so does not effect the query execution time. However this does lead to the problem of trying to effectively combine a purely linkage based score with a query-specfic score, without introducing the problem of *topic drift*. This can arise because the documents with the highest PageRank scores are not necessarily going to be relevant to a particular query (as the documents' PageRank scores are query-independent) and so the incorporation of PageRank into the ranking process may

promote documents that may not be relevant to the query.

2.2.4.2 Hyperlink Induced Topic Search

As opposed to PageRank, Hyperlink Induced Topic Search (HITS), proposed by Kleinberg (1999) is a query-dependent form of linkage analysis. An initial subset of documents is formed from the output of a standard IR system for a query. From this initial subset it calculates two scores for each document: an authority and a hub, defined as follows:

Authority: A good authority page is one that contains a lot of information relevant to the query, as well as being linked to by many pages that are relevant to the query.

Hub: A good hub page is one that contains many links to pages that are good authorities.

The basic hub and authority scores are calculated as follows: an initial set of relevant documents (top 200 for example) is retrieved from a standard IR system for a specific query (known as a *base set*). This set is then expanded using off-site inlinks as well as off-site outlinks to produce an expanded set of relevant documents (1,000 - 5,000 documents). We make use of the relationship between hubs and authorities via an iterative algorithm that maintains and updates numerical weights for each page. Thus, with each page (p), we associate a nonnegative authority weight ($x^{(p)}$) and a nonnegative hub weight ($y^{(p)}$). We maintain the invariant that the weights of each type are normalised so their squares sum to 1:

$$\sum_{p \in S_\sigma} (x^{(p)})^2 = 1 \tag{2.12}$$

and

$$\sum_{p \in S_\sigma} (y^{(p)})^2 = 1. \tag{2.13}$$

We view the pages with larger x and y values as being "better" authorities and hubs, respectively (Kleinberg, 1999).

The HITS approach however, has the disadvantage that it is calculated at query time: not only as it requires extra resources from the search system at query time, but also as the system response time is increased. This represents a major disadvantage to the general user, who requires the minimum delay in the system response.

Having examined various ways in which an IR system may provide an estimate of the relevancy of the documents in the collection to a user's information need, we now look at the ways in which these documents may be assessed for relevance to the user's need. We also look at various evaluation measures, by which to ascertain which approach performs best.

2.3 Evaluation

To consider the true effectiveness of a search system, there are many areas that need to be accounted for: accuracy of results, speed, efficiency, to name but a few. Within each of these areas there are various metrics by which to measure their effectiveness, and depending on the type of *information need* of the user, certain evaluation metrics may be more appropriate. In this section we outline the main types of information needs of a IR system user. We then show how the retrieval system's effectiveness, in meeting those needs, may be evaluated.

2.3.1 Information needs and search tasks

> "*Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it.*", Samuel Johnson (1775)

According to both Broder (2002) and Rose and Levinson (2004) web queries can be classified into three different types: *informational*, *navigational* and *transactional/resource*. The following describes each of these query classes, as well as outlining the various search tasks used in research to replicate these search scenarios.

2.3.1.1 Informational Search

An informational query occurs when we are seeking information on a particular topic. *Ad hoc* search is the typical informational search task. Ad hoc search involves a user searching for a particular topic on a static collection of documents. In this task the user's query or topic comes from a general information need, in which they are looking for information regarding some topic, which (depending on the collection used) is likely to have many relevant documents. An example ad hoc topic might be: "Tourist attractions in the United States".

2.3.1.2 Navigational

With a navigational query the user is trying to navigate to a particular document, and possibly has a clear idea of the document that they want. This type of query is discussed below in terms of *named page finding* and *homepage finding*.

- **Named Page Finding:** In the Named page finding task there is only one particular page (or near duplicates) that is considered relevant to the query. This task mimics the case where the user knows exactly the document that they are looking for (as they may have viewed the document previously), and they wish to find that specific document again. Here the emphasis for the search system is to return just that document to the user, although in practice the system usually returns a small list of the most likely candidate documents. For example, the query "*DCU library opening hours*", specifies the user's requirement for very specific information about the opening hours of the Dublin City University library, to be found at *http://www.dcu.ie/ library/about/openhours.htm*.

- **Homepage Finding:** In a homepage finding task, the aim is to find the relevant homepage associated with the query. This is similar to the named

page finding task, and is in fact a more defined case of that task, in that we know that the named page is a homepage (not just any page in the entire collection). The query is the name of an entity which has a specific homepage associated with it, and the aim of the task is to find this page and return it at the top of the result list. For example with the query "Dublin City University" the associated homepage would be *http://www.dcu.ie*. The user may be seeking particular information that they know to be available on that specific page, as would be the case in the named page task. Also, quite often with the homepage finding task the user is looking for general information on a particular topic, or looking for a good entry point to explore from, and usually a homepage provides a good entry point for this type of browsing.

2.3.1.3 Transactional/Resource Search

With a transactional, or resource seeking information need, the user has a particular transaction that they need to perform or a particular resource that they are seeking, such as to book a hotel or download a file. The task of the IR system is to identify this need, then find the most suitable web page at which to perform this transaction. These types of queries can often be quite difficult to deal with correctly. Aside from identifying a relevant result, based on the topic match between the document and the query, the search system may also have to provide different results to different users, depending on certain criteria, such as the the user's location. For example, if an Irish web user issues the query "pay motor tax", they would most likely be looking for www.motortax.ie, whereas this page would be of little use to web users from Singapore wishing to pay their motor tax.

Although these types of queries are important to understand and worthy of substantial research, there is yet no test collections available to allow for this. Most of the current research in this area is carried out by large commercial search engines, who have an interest in providing more accurate answers for their users.

2.3.2 Evaluation Strategies

In order to evaluate the efficacy of an IR system it is necessary to evaluate the performance of the system using a number of criteria. In terms of efficiency, the query throughput, as well as the system resources required to execute a user's query must be evaluated. These are elements of a search system that can be rigorously tested in a laboratory environment, in which accurate performance figures can be gathered. However measuring the degree of user satisfaction and fulfillment of information needs are much more difficult concepts to evaluate.

The satisfaction of a user's information need is something that can be difficult to evaluate, as we must consider all factors that the user brings to the situation: their prior knowledge; awareness of information available; use of the information; time constraints, etc. We must also be aware that a user's information need may be constantly changing or be updated, due to new information that the user receives from the IR system.

When trying to determine the accuracy of the returned set of documents in response to a user's query, the key concept underlying this evaluation is *relevance*. However it has been found that this concept of relevance is not a static relationship between an information need and information, "relevance is not fixed but is a temporal and fluid concept that is sensed or observed at a specific moment for a particular need" (Park, 1993), and "relevance judgments are users' evaluations of information...in relation to their information need situations at particular points in time" (Schamber et al., 1990). Nonetheless, currently the majority of IR system evaluations are computed within a laboratory setting, using document collections and pre-determined relevance judgments between queries and documents. There are also issues with the task of completely assessing a document collection for relevance to a set of queries. This is a non trivial task, particularly with a large collection of documents, which is currently addressed using *document pooling*.

In order to evaluate an IR system for a certain search task, a suitable collection of documents must be gathered, so that repeated experiments can be run on the

same collection. These collections (which are referred to as *test collections*) remain static, as changes to the collection may cause some deviation in the results of the experiments. Often the same test collection is used between a large number of research groups, so that their results may be compared.

The remainder of this section describes methods used in research to evaluate the performances of search systems, on a test collection of documents and generated queries.

2.3.2.1 Human Relevance Judgments

The most effective way of judging the relevance of a document to a query is to use human relevance assessors, who evaluate each document's relevance to each query. However, there may be discrepancies among human judges about which documents are relevant and which are non-relevant. The difficulty of this assessment is also dependent on what is being evaluated, for example if it is the quality of a document that is being assessed, this is a somewhat more ambiguous notion, when compared to the relevance of a document to a query, and so the discrepancies between judges can become more pronounced (Amento et al., 2000). Attention also needs to be given to other aspects of the evaluation process, such as ensuring that the judge is representative of a general searcher using the system.

2.3.2.2 Document Pooling

Due to the increasing size of test collections, it has become increasingly difficult for a human assessor to assess every document in a collection as being either relevant or non-relevant for every query. To allow effective relevance judgments to be carried out on large test collections, document pooling can be employed. This involves taking the top N results from a number of different IR systems for a set of queries, and these documents are then *pooled* together. Then it is this pool of retrieved documents that is assessed (rather than the entire collection) (Sparck Jones and van Rijsbergen, 1976). It is hoped that the diversity used by the different systems

provide a large enough pool of results, so that the majority of relevant documents can be found for each query. This process of document pooling and judging has been shown to be sufficient for research purposes (Zobel, 1998; Voorhees, 1998).

The *variable pool depth* approach (Zobel, 1998) has been shown to improve on this initial method by increasing the number of relevant documents found. This works by firstly judging each query to an initial depth, and then for each query using extrapolation to predict the likely number of relevant documents to be found if the per-query pool depth is increased. One can then, either identify the most promising runs and judge them to a greater depth, or remove the runs with the least number of relevant documents, so that they need not be considered further.

Aslam and Yilmaz (2006) provide a method of evaluating retrieval systems, using only a limited number of relevance judgments, by inferring document relevance based on average precision. Given: the ranked lists of documents returned in response to a given topic; the average precisions associated with these lists; R (the number of documents relevant to the topic), their approach aims to find the binary relevance judgments associated with the underlying documents. They define this as a *constrained integer optimisation* problem. However to alleviate the problem of this being intractable, they relax the condition that the inferred relevance assessments must be binary, instead allowing the inferred relevance assessments to be probabilities of relevance. They can then infer an *expected value* for average precision from these probabilistic relevance assessments, which are calculated as follows:

$$E[AP] = \frac{1}{R} \sum_{i=1}^{Z} \frac{p_i}{i} 1 + \sum_{j=1}^{i-1} p_j \tag{2.14}$$

where p_i is the probability of relevance associated with the document at rank i in the list of length Z. To ensure that the inferred relevance judgments incur average precision values "close" to those given, they minimise the sum squared error between the real and the inferred average precision values. Using this approach they show that once given values or estimates of average precision, they can then accurately

infer the relevances of unjudged documents and so allowing a large judgment pool to be created, from a relatively small number of judged documents. This is something that is particularly appealing for carrying out effective evaluations on large document collections.

2.3.3 Evaluation Measures

How the efficacy of a search system is to be evaluated is very much dependent on the task that is being performed. For example, for a general web query there may be a huge number of relevant documents, but the user may only consider the top 5 - 10 documents. On the other hand, for certain medical searches it may be vital to have all relevant documents, with the user willing to browse all documents. With this in mind we must consider different performance measures, depending on the search task being performed.

2.3.3.1 Precision and Recall

To examine precision and recall, consider a query that is issued to a search system, and this produces a set of candidate relevant documents C (with the actual relevant documents to the query being the set R). If the intersection of the two sets R and C forms the set Rc, i.e. the set of relevant documents found by the system, then precision and recall can be explained as follows:

Precision

Precision is the fraction of the documents found within a certain cutoff point which are relevant:

$$Precision = \frac{|Rc|}{|C|} \tag{2.15}$$

Recall

Recall is the fraction of the total relevant documents found by the system within a certain cutoff point:

$$Recall = \frac{|Rc|}{|R|} \tag{2.16}$$

Precision and recall are generally used to evaluate the usefulness of informational type queries. Depending on the actual application, one may be more important than the other. The goal is to have high precision and high recall, however in practice one is generally achieved at the expense of the other (as shown in figure 2.8). Finding a suitable balance between the two is the realistic goal.

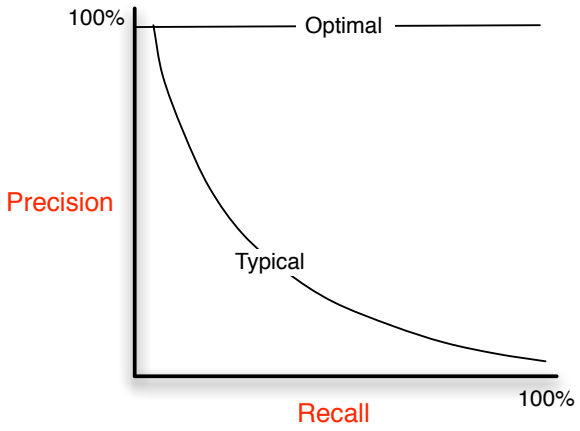


Figure 2.8: Precision-Recall Tradeoff.

Mean Average Precsion (MAP)

A widely used single measure of retrieval performance, which gives an indication of the number of non-relevant documents incurred before all relevant documents are found is *mean average precision* (MAP). If there are relevant documents that are absent, each of these are also taken into account, therefore indirectly measuring recall. In general, the value of MAP also depends on the cut-off level (the number of documents to be evaluated for each query).

2.3.3.2 Mean Reciprocal Rank and Success Rates

The Mean Reciprocal Rank (MRR) measure is commonly used when there is only one correct answer to a query, as in a homepage finding search task. For a given query the *reciprocal rank* is calculated as the reciprocal of the position in the ranked list that the document is found at. For an overall measure for a set of queries the average across all queries is taken.

Similarly, *success rate* evaluation measures are often used when there is only one correct document corresponding to a query. The success rate is indicated by S@k, where k is the cutoff rank and indicates the percentage of queries for which the correct answer was retrieved in the top k ranks.

2.3.4 The Text REtreival Conference

In the early 1960's the Cranfield College of Aeronautics created a static document collection of 1400 documents with 225 queries. Due to the relatively small size of the collection, each of the documents could be manually classified as relevant or not to each of the user queries. Based on these queries and on manual relevance judgments, the group were then able to evaluate different indexing and retrieval techniques, using such measures as precision and recall.

In an effort to scale the size of these document test collections to more realistic sizes, the Text Retrieval Conference (TREC) was established in 1992, by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA), as part of the TIPSTER Text program. Its purpose was "to support research within the information retrieval community, by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies" (TREC, 2000). This has proven very successful, as it provides the framework to allow researchers to test the effectiveness of different approaches on a common test collection. TREC's primary focus began on ad hoc search, and has since grown to include dozens of other IR related tasks, with a total of 117 partici-

pating research groups in the 2005 TREC workshop. Table 2.1 shows a comparison of some of the major test collections used by TREC.

Table 2.1: Overview of TREC collections

Collection	No. of Documents	Size	Year	Composition
VLC2	18.5 M	100 GB	1998	Crawl from Internet Archive
WT2g	0.25 M	2 GB	1999	Subset of VLC2
WT10g	1.7 M	10 GB	2000	Subset of VLC2
GOV	1.25 M	18.1 GB	2002	2002 crawl of .gov
GOV2	25 M	426 GB	2004	2004 crawl of .gov

Next we discuss the GOV2 collection (which is the largest test collection used by TREC), as we utilise this collection in our experiments in Chapter 6.

2.3.4.1 GOV2 Collection

The GOV2 collection consists of a collection of documents crawled from the .gov domain of the web in early 2003, and amounts to a large proportion of the available pages in .gov. The collection amounts to over 25 million documents, from over 17,000 distinct hosts, and occupies 426 gigabytes of disk space. This is composed mainly of html and text, as well as the extracted text of pdf, word and postscript files. Figure 2.9 shows a sample document from the GOV2 collection.

2.3.4.2 The Terabyte Track in TREC

The terabyte track in TREC began in 2004 (Clarke et al., 2004), in an effort to again scale the size of the test collections being used, from approximately 18 gigabytes in size to around one terabyte. This provides a more realistic representation of the size of document collections being dealt with by commercial search engines. As it turned out, the crawl of the .gov web sites amounted to less than half a terabyte of text, nonetheless this provides a substantial increase in size over the previous GOV collection.

The goals of the terabyte track in TREC were to:

```
<DOC>
<DOCNO>GX000-25-0717165</DOCNO>
<DOCHDR>
http://www.nysb.uscourts.gov/misc.html
HTTP/1.1 200 OK
Date: Tue, 09 Dec 2003 17:48:21 GMT
Server: Stronghold/4.0 Apache/1.3.22 (Unix) PHP/3.0.18 mod_perl/1.26 mod_ssl/
2.8.7 OpenSSL/0.9.6c
Last-Modified: Wed, 19 Nov 2003 20:59:44 GMT
ETag: "48c2e-958-3fbbd9c0"
Accept-Ranges: bytes
Content-Length: 2392
Content-Type: text/html
Connection: Close
</DOCHDR>
<HTML>
<HEAD><TITLE>New York Southern Bankruptcy General Information</TITLE>
</HEAD>
<BODY>
<FONT SIZE=+2>
UNITED STATES BANKRUPTCY COURT <BR>
<EM>Southern District of New York</EM> <BR>
</FONT>
<BR>
<H1><FONT COLOR="BLUE">GENERAL INFORMATION</FONT></H1>
PROCEDURE: <br>
...
</BODY>
</HTML>
<DOC>
```

Figure 2.9: Sample GOV2 document

- Investigate the performance of traditional search algorithms on a terabyte sized collection.
- Investigate the performance of evaluation measures, as well as the relevance judgement process, on a larger test collection than previously used.

The main task associated with the terabyte track is the *ad hoc* task: "The adhoc task in TREC investigates the performance of systems that search a static set of documents using previously-unseen topics" (Clarke, 2006). Figure 2.10 shows a sample ad hoc query from the 2004 task:

We can see from a sample query, or *topic*, (shown in Figure 2.10) that the topic is divided into different sections: each topic has a unique number (num); a title, which

<top>
<num> Number: 705
<title>
Iraq foreign debt reduction
<desc> Description:
Identify any efforts, proposed or undertaken, by world governments to seek reduction of Iraq's foreign debt.
<narr> Narrative: Documents noting this subject as a topic for discussion (e.g. at U.N. and G7) are relevant. Money pledged for reconstruction is irrelevant.
</top>

Figure 2.10: Sample ad hoc topic

is the actual user query; a description, and a narrative section, which help to clarify the information need of the topic, as well as helping to remove any ambiguities that may exist.

For the ad hoc task, participants may submit either a manual or an automatic query. For an automatic query the system may use any or all of the topic fields specified. For the experiments contained within this thesis we have utilised only the title field of the topic, as we feel that this is the most representative of a typical user query, which we are trying to replicate. Participants may also submit a manual run, in which a searcher manually forms what they believe to be the best query for the topic.

In both 2005 and 2006 the track also consisted of a named page finding task (as discussed in 2.3.1.2), as well as an efficiency task, which attempts to measure (and allow comparisons of) the effectiveness of IR systems, in terms of their efficiency.

2.4 Summary

In this chapter we described the operation of a typical information retrieval system, outlining its main components and describing how each contributes to the system.

We then took a more detailed look at the operation of the retrieval process, how relevance is assessed between a document and a query using the classic retrieval

models. We also presented the use of linkage analysis, as an additional way to provide document ranking.

Having looked at the way in which documents are returned to the user, we then focussed on how to assess the effectiveness of the system, describing some of the typical user tasks, then discussing how to assess the performance of the system using certain measures. Finally we introduced TREC, which provides the framework for the evaluation of system effectiveness that we utilise for the experiments carried out in this thesis.

After discussing the background area (in this Chapter), we can now move closer to the area of work carried out in this thesis. We do this by, firstly detailing the related work in the domain (in Chapter 3). We then discuss the main subject of the thesis in Chapter 4.

Chapter 3

Related Work

In this chapter we describe related research to the work carried out in this thesis. This is done in an effort not only to show what currently has been done, but also to explain how the work carried out in this thesis correlates with these other works. Specifically, we describe associated work in the area of *reducing the search space*, which prunes the number of documents examined at retrieval, in order to reduce the effort from the retrieval system at query time.

We firstly provide a detailed analysis of the typical steps involved for an IR system to retrieve documents in response to user input. This will provide us with an example, that we will use to compare and contrast different methods for reducing the search space. We also use this example as a means to illustrate our own approach in this area.

3.1 Information Retrieval Search Implementation

In Chapter 2 we looked at the operation of an IR system, as well as making a more detailed examination of the retrieval process, namely, how relevance is assessed between a document and a query during the retrieval stage. Here we wish to examine in more detail how the IR system performs this retrieval task. Specifically, we examine how it extracts information from the inverted index, and then calculates a

similarity measure between the user query and the documents.

If we look at the retrieval process of the IR system (as shown in Figure 3.1), we can see the operations undertaken in order to provide the user with a list of documents, that aim to answer their query.

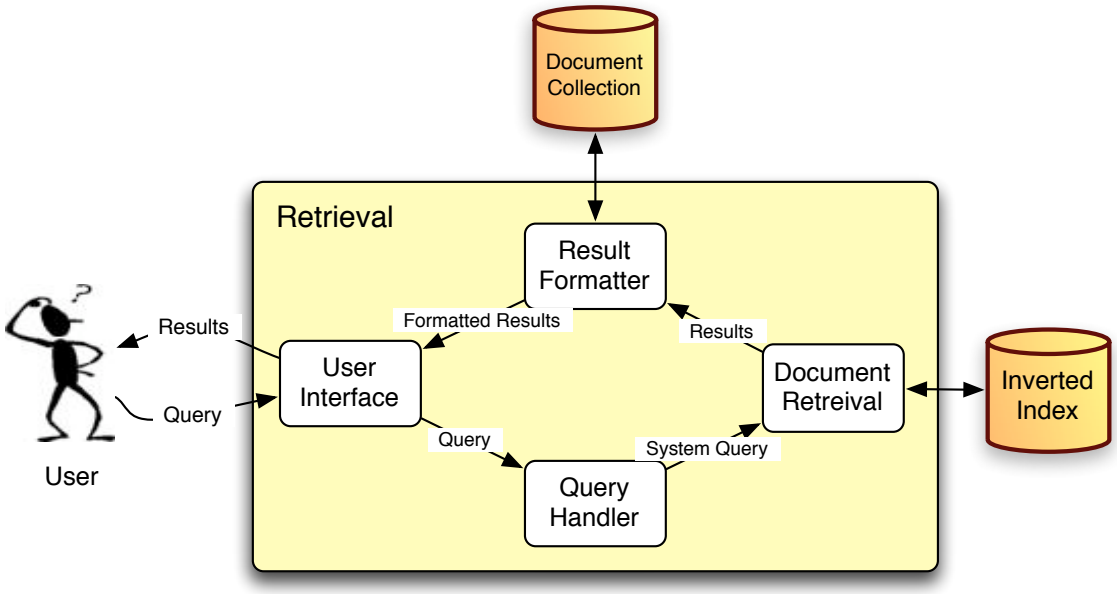


Figure 3.1: Retrieval component of the IR system

The process that we are chiefly concerned with is that of *document retrieval*, as highlighted in Figure 3.2.

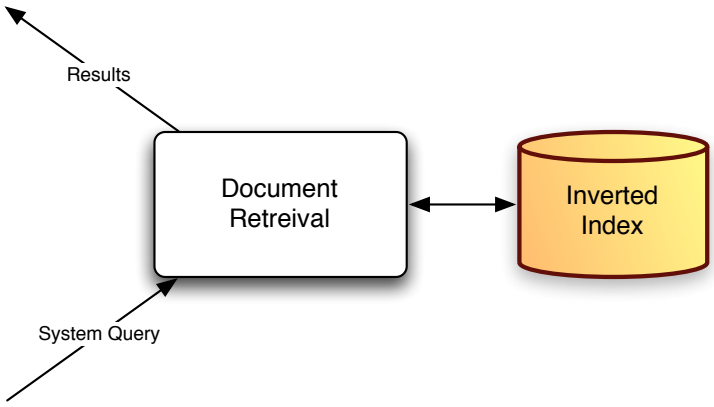


Figure 3.2: Document retrieval process highlighted

This process receives a system query[1], and based on this system query the retrieval process will consult the inverted index, in order to provide the user with a list of relevant documents. As described in Chapter 2, the inverted index consists of a lexicon, which holds each of the terms in the index of the system. The lexicon also holds the location of the postings list associated with each of these terms. When the system wishes to execute a query, the document retrieval process consults the lexicon for each term in the query. This provides the locations of each postings list, and so allows the postings to be retrieved (usually from disk, particularly for a large document collection) for each term. Then for each term, the postings are evaluated for potential relevancy according to a specified retrieval strategy, as discussed in section 2.2. This provides a list of potentially relevant documents, that are to be returned to the user.

To illustrate this process further we provide the following example, with the user specified query: "Tourism in Italy". Firstly the user query is transformed into a system query; for example, by converting all characters to lower case, this becomes "tourism in italy". The document retrieval process receives this query, then looks up each term (in the lexicon), on finding a term the postings list for that term can then be located and all the postings for that term are accessed. This process of accessing all postings for each query term is shown in Figure 3.3.

All the postings associated with a particular term show us in which documents that term occurred, as well as the number of times it occurred in each document (in addition to other more detailed positional information, if the retrieval being performed requires it). The next step in the retrieval process is to decide on the documents that are to be returned to the user. For this we must evaluate each of the postings associated with each of the query terms, and based on the retrieval strategy we employ, we update an *accumulator's* "score" with each new posting encountered. An accumulator holds the score accumulated for each document, as

[1]A system query is the result of a user query undergoing the same transformation as the documents' terms in the collection, before they are stored in the inverted index.

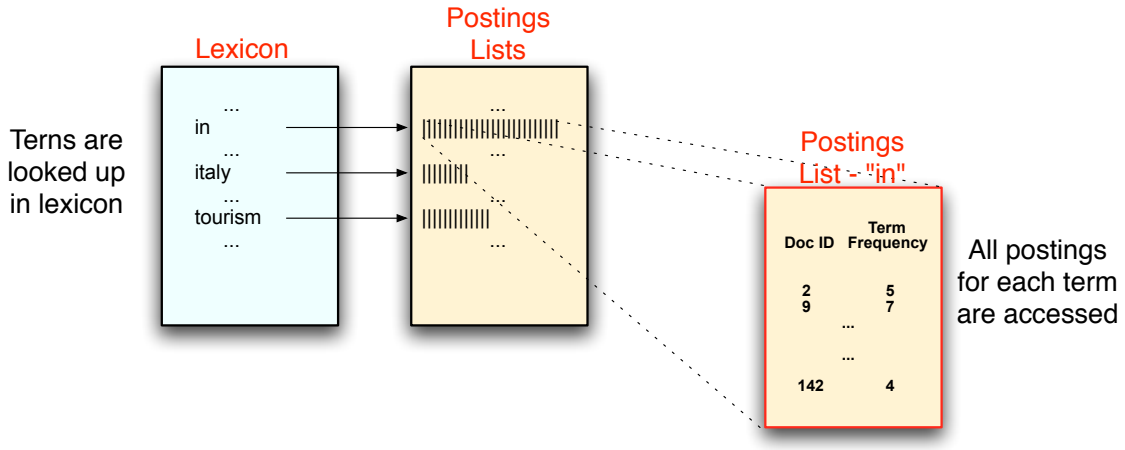


Figure 3.3: Accessing postings for each query term

each of the postings are processed (according to the given retrieval strategy). This step is demonstrated in Figure 3.4.

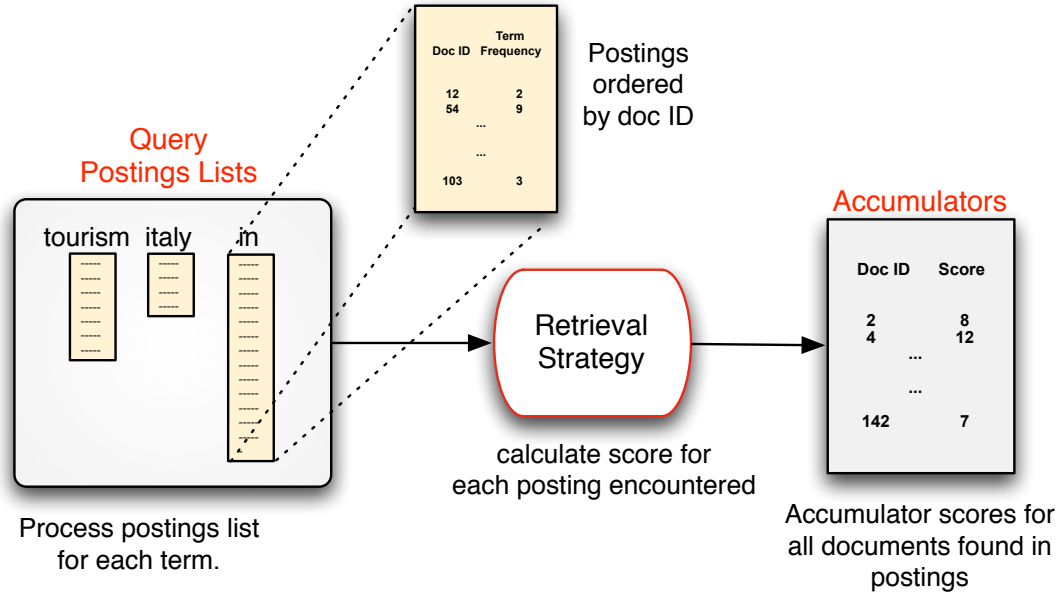


Figure 3.4: Calculate accumulator scores

It is worth noting, that for most purposes a retrieval strategy that provides a ranked list of results is most favourable, although for Boolean search we may also hold binary scores in the accumulators. Once all the postings lists have been

evaluated, the documents are sorted based on their accumulator's score, and then this list, or a portion of it (for example the top 20) is returned to the user.

3.2 Reducing Search Space

Having outlined how the typical document retrieval process deals with a user query, we now illustrate some alternative approaches that aim to reduce the search space, while also aiming to providing the same accuracy of results.

3.2.1 Nearest Neighbour Search

In the context of information retrieval, finding the "nearest neighbour" corresponds to finding the n closest documents to a query, where "closeness" is measured by a similarity measure. This typically can be done by performing a sequential search through the complete collection, and then selecting the top n documents (as shown from our previous example). In order to reduce the search complexity, it has been proposed that the query terms be evaluated in order, with terms with the lowest f_t starting first. These are the terms with the lowest number of occurrences in the collection, and therefore (according to most retrieval strategies) of most influence to the ranking process. Using this approach, the inverted file records containing query term(s) with the highest f_t are not searched at all once an *upperbound* is reached (Smeaton and van Rijsbergen, 1981).

Using the previous query example, "tourism in italy", this approach would have the effect of changing the order in which the postings lists are processed by the retrieval system. For example, with the terms, *tourism, in,* and *italy* having f_t values of 28, 63 and 21 respectively, the order in which the postings lists are processed is changed, so that the postings of the term with the lowest f_t are processed first, i.e. *italy*, followed by *tourism*, then finally *in*.

The algorithm for calculating the nearest neighbours maintains two sets: R and S, where R contains all documents, which at any point are candidates for the final

set of nearest neighbours, and S contains all documents that have been examined so far. When processing each of the postings, if the document is in the set S it is ignored (as it has been encountered previously), otherwise it is added to S, and a similarity value is calculated between the document and the query. If this value is higher than any of the values in R, then the document is also added to R (as a candidate for the list of nearest neighbours). After processing all documents for a given term, a maximum possible similarity value of the documents containing the unprocessed query term(s) can be calculated: the *upperbound*. If this upperbound is lower than the current best value, then the process can be terminated. This means that whole postings lists for a particular term, or terms, may not need to be processed in order to find the nearest neighbours, and so allowing faster query throughput. This process is illustrated in Figure 3.5.

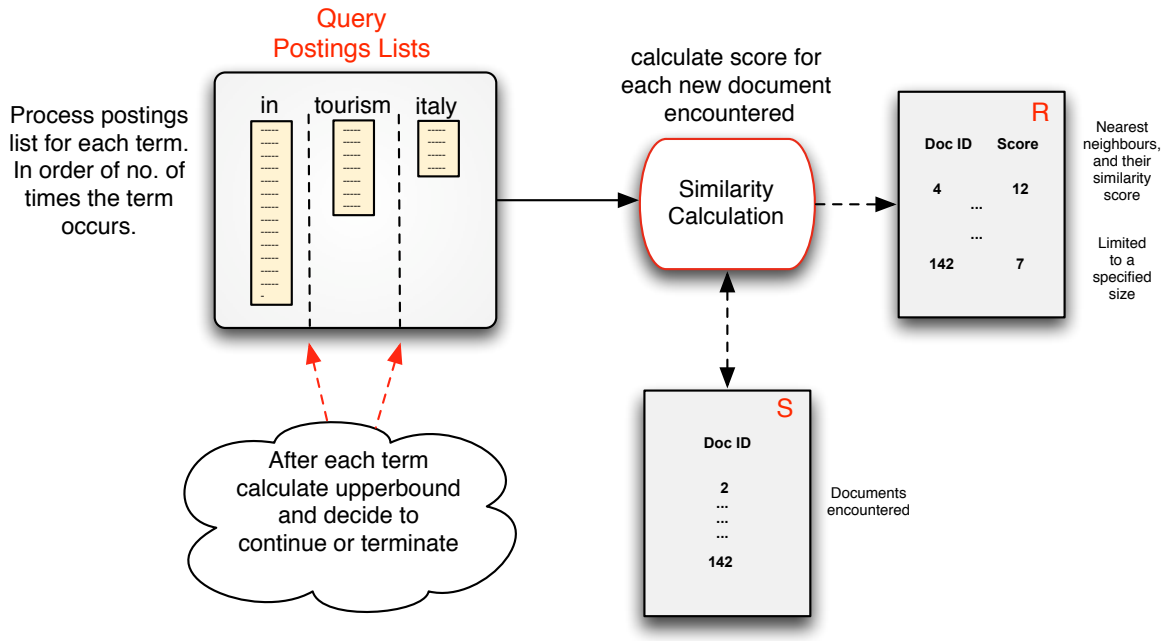


Figure 3.5: Nearest neighbour calculation

This approach performs substantially faster than a complete search, while at the same time it maintains the same level of performance. However, it requires a list to be maintained in order to calculate the threshold. Also the queries used in the experiments of Smeaton and van Rijsbergen (1981) contained on average more

than 7 terms, which is much more than the number of terms used in an average web search (2-3 terms on average according to Spink and Jansen (2004)), and so the gains of this approach would not be as great if used in this typical web search scenario.

3.2.2 Self-Indexing Index

Moffat and Zobel (1994) proposed modifying the inverted index, to create a *self-indexing* index, to allow faster scanning, or *skipping* into the inverted index. This was shown to potentially add 20% to the size of the inverted file. However it allowed Boolean queries to be processed in 20% of the time (5-10 query terms), and ranked queries could also achieve a "time saving of about 50%...without measurable degradation in retrieval effectiveness", by restricting the accumulators used at query time (40-50 query terms) (Moffat and Zobel, 1994).

3.2.2.1 Skipping

By inserting additional access points, or *synchronisation points*, in each of the inverted lists (where decoding can commence from), a mechanism is provided where postings that are not required to be evaluated for a particular query can be skipped. According to Moffat and Zobel (1994), when these values are "interleaved with the runlengths of the list as a sequence of skips, a single self-indexing inverted list is created".

If we extend our index example of <d, $f_{d,t}$>, from section 2.1.2:

<15,4><17,1><104,2><112,19><142,8><157,2><204,3><248,4><304,1>

and then storing these using d-gaps, this becomes:

<15,4><2,1><87,2><8,19><30,8><15,2><47,3><44,4><56,1>

With skips over every three pointers, this becomes:

<<15,a2>><15,4><2,1><87,2><<112,a3>><8,19><30,8><15,2><<204,a4>><47,3><44,4><56,1>

where a_2 is the address of the first bit of the second skip pair, a_3 is the address of the first bit of the third skip, etc. The last of the redundancy in the above representation can be alleviated by storing only the differences between the documents and the pointers in the skips. Also the first document number in each set of three <d, $f_{d,t}$> is no longer needed, as it is held in the skip. The final inverted list looks like the following:

```
<<15,a2>><15,4><2,1><87,2><<8,a3-a2>><19><30,8><15,2><<47,a4-a3>><3><44,4><56,1>
```

For a conjunctive Boolean query, the maximum number of accumulators required is the frequency of the least common query term. If the query is processed from least to most common query term, only the documents that are common to all need to be evaluated. This self-indexing allows the inverted lists to be scanned quickly in order to retrieve these required postings. It also allows postings that are not common to all query terms processed at that point to be skipped (enabling faster evaluation of Boolean queries).

Moffat and Zobel (1994) identify "the principal costs of ranking" as "the space in random access memory, and the time required to process inverted lists". In an attempt to eliminate the number of accumulators necessary to process a query, they proposed two strategies: *quit* and *continue*. The quit strategy is carried out in a similar way to that of the nearest neighbour search: ordering query terms by decreasing weight, and then stopping once a certain criterion is met. An alternative to the quit strategy, is to continue processing once this criterion is met (however at this stage no new accumulators are created): this is known as the *continue* strategy. These are demonstrated further in Figure 3.6, again using our previous example query of "tourism in italy".

3.2.3 Sorted indices

As alternatives to the traditional document-ordered index, research has been carried out on sorting posting lists by various metrics, as well as other methods of reducing the time and resources necessary to produce a ranked set of results in response to a

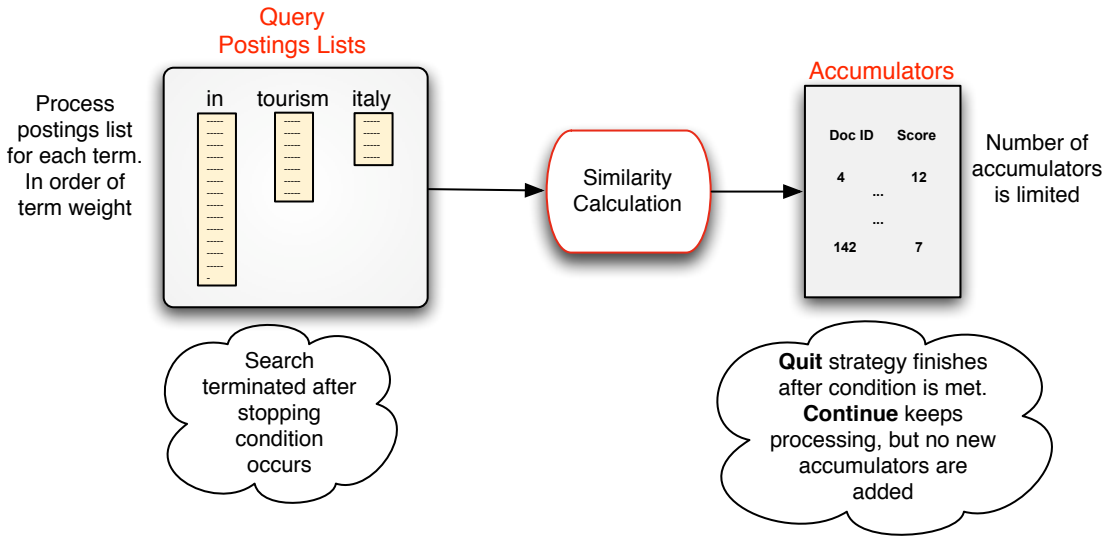


Figure 3.6: Processing using quit and continue strategies

query. These are now described.

3.2.3.1 Frequency sorted

Persin (1994) and Persin et al. (1996) introduced frequency-sorted indexes, where the posting lists are sorted by decreasing within-document term frequency $d_{t,d}$. Again using our previous example of a postings list:

<15,4><17,1><104,2><112,19><142,8><157,2><204,3><248,4><304,1>

which is currently ordered by the document identifiers. By sorting in order of the within-document frequency, $d_{t,d}$, this becomes:

<112,19><142,8><15,4><248,4><204,3><157,2><104,2><17,1><304,1>.

Persin (1994) suggests that the majority of query time and memory is spent on processing the most commonly occurring terms, which are the least informative in terms of contribution to query-document similarity. He proposes a frequency sorted index, so that documents may be filtered in order to reduce the number of documents to be processed. In doing so, both the main memory in use, as well as the query time, are reduced. This reduction is achieved by "processing only the most

informative parts of the term entries"; to allow this the postings lists are sorted in descending order of the within-document frequency of each posting.

In order to filter documents at query time, two thresholds are calculated for each query term: "an insertion threshold k_{ins} and an addition threshold k_{add}, where $k_{ins} \leq k_{add}$". During processing, a $tf.idf$ score is calculated for each document in the postings list, and if this $tf.idf$ score is more than k_{ins}, the document is considered as a candidate. If the document is already present as a candidate then this gets added to its previous score, otherwise a new accumulator is created for this document. If $tf.idf$ is less than k_{ins}, but more than k_{add}, then if this document has an accumulator, $tf.idf$ is added to the accumulator. Finally, if the $tf.idf$ score is less than k_{add} this information is ignored and the next posting is processed. The documents that are skipped are considered unimportant as if a large enough number of documents are found with high similarity to the query, then those documents with small partial similarities are unlikely to have a major impact on the outcome of the final ranking. However if each of the postings lists are sorted in decreasing order of $d_{t,d}$, then there is no need to keep processing more postings once $tf.idf < k_{add}$, as all postings below this point will have the same or lower $tf.idf$. Figure 3.7 illustrates how this frequency sorted index processes an example query.

Using this type of index it has been shown that the CPU time and disk traffic can be reduced to approximately one third of the original. Persin (1994) also showed how a net reduction could be achieved in the index size, even though the conventional d-gap compression approach used in document-ordered indexes (described in Witten et al. (1999)), could no longer be utilised. However, it must also be noted that these experiments were carried out using queries ranging in length from 35 to 130 terms, which would allow greater gains to be achieved using this approach, rather than the much shorter queries that a user would typically use in interactive web searching.

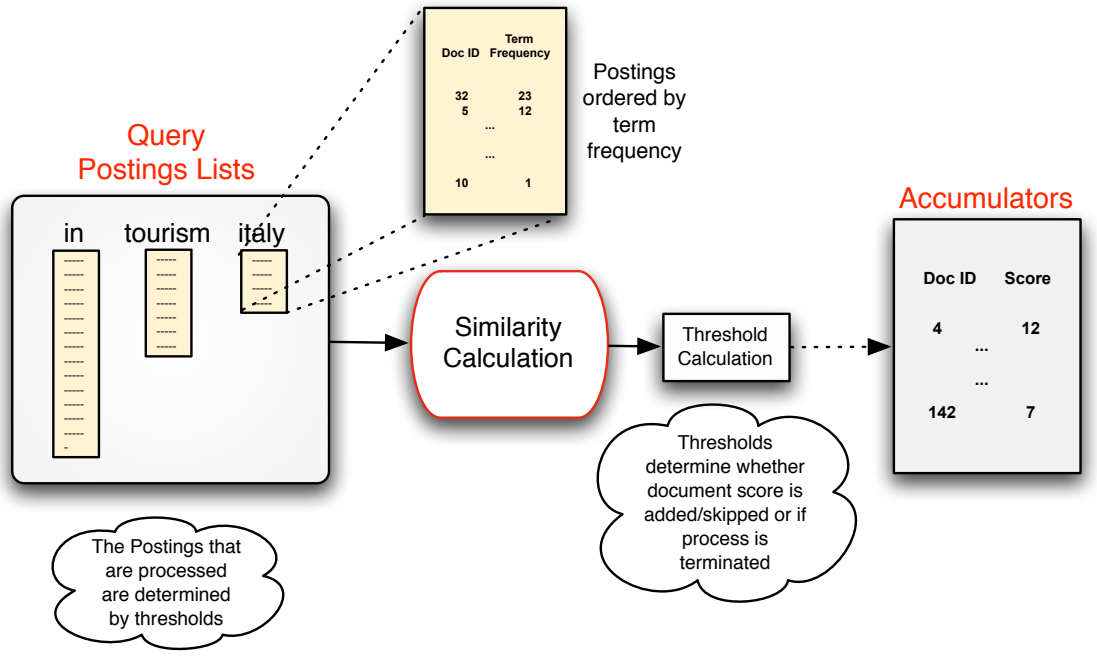


Figure 3.7: Query processing with a frequency sorted index

3.2.3.2 Impact sorted

Impact-ordered indexes presented by Anh et al. (2001) and Anh and Moffat (2002a,b), are a form of inverted index that is ordered by quantised weights (or *impacts*). This provides the facility to only decode the postings with the highest impact, thereby reducing the query time, while at the same time improving the retrieval effectiveness.

Moffat and Zobel (1994) proposed filtering documents based on the *idf* component of the *tf.idf* term weighting scheme, while Persin (1994) and Persin et al. (1996) sorted documents based on the term frequency, thus allowing filtering based on the $tf.idf$. Anh and Moffat suggest sorting using a normalised $tf.idf$ term weighting scheme, or based on the *impact* of a term in a document, similar to Buckley and Lewit (1985). This is known as the *term impact*.

The obvious drawback to using this impact value for sorting posting's entries is that it is a floating point number, and so would be unsuitable for compression. However, following on from Moffat et al. (1994), they make use of quantisation, where each impact value is approximated by a b-bit integer. The size of the b value

effects both the size of the index as well as retrieval effectiveness. A low b value does not approximate the impact value well (and so degrades retrieval effectiveness), although it does help the compressibility of the index. On the other hand, a large b value will approximate the impact value much better, but in doing so will increase the size of the index.

Anh and Moffat later produced a "document-centric impact" (Anh and Moffat, 2004), which calculates the impact of the term within the document which it is contained (rather than a global impact that had previously been applied). The rationale for this is to give all the documents an equal spread of high impact terms and low impact ones.

An *impact sorted* index is then created by sorting the posting lists by their quantised impact value, with groups of postings having the same impact value being blocked together.

Anh and Moffat experimented with various early termination schemes, such as the *quit* and *continue* schemes introduced by Moffat and Zobel (1994). Also the use of the quantised components of their *tf.idf* similarity measure allow for faster and more memory efficient calculation of similarity scores. However in Anh et al. (2001), their term impact sorted index did not perform as well (on a per postings processed basis), when compared to a term frequency sorted index (as in Persin et al. (1996)) when low numbers of postings had been processed. That is to say, that for the same number of postings processed, the term frequency sorted index performed more effectively in terms of *precision at 10*, even though more information and computational effort would have gone into the calculation of the impact, compared to taking the raw term frequency. This may be explained by longer documents being penalised more severely than shorter documents, due to the normalisation of the *tf.idf* factors by the document length in the generation of the impact score. The effect of this over penalisation of shorter documents, due to the normalisation of the term frequency is addressed by Ferguson et al. (2005c), in which an increase in terms of *precision at 10*, is gained over term frequency sorted indexes on a per

postings processed basis.

3.2.3.3 Access ordered

A further form of sorted index (introduced by Garcia et al. (2004)), is an Access Ordered Index, in which they order the postings lists based on previous queries. Founded on the idea, that even with a large number of different queries the same documents are ranked highly, the postings are ordered so that the documents returned by the IR system for the most queries are towards the top of the lists and the less retrieved documents are stored towards the bottom. Again, using this approach not all postings need to be decoded, as the most accessed postings will be evaluated first, and so effective retrieval can be performed using much less resources than in a conventional approach.

In order to generate the *access counts*, Garcia et al. (2004) firstly processed 1.9 million web queries from an Excite search engine log, on a collection of 1.6 million documents. For each of these queries they generated a ranked list of documents, using the Okapi BM25 formulation (as discussed in Chapter 2). For every time a document occurred within the top 1000 of this ranked list for a query, its *access count* was incremented. Following the execution of these queries each document has an associated *access count*, and so allowing each postings list to be sorted in descending order, based on the document's access count.

Again taking our previous example of a postings list in the form <d, $f_{d,t}$>:

<15,4><17,1><104,2><112,19><142,8><157,2><204,3><248,4><304,1>

and extending this to the form <a_d, d, $f_{d,t}$>, where a_d corresponds to the access count score for that document gives us:

<3,15,4><12,17,1><0,104,2><2,112,19><1,142,8><1,157,2><2,204,3><1,248,4><5,304,1>.

With this extra information, the postings can now be ordered by the access count values (rather than the document numbers). However this a_d value is not actually stored in the index, and is instead held in an in-memory mapping table. The postings

are thus ordered by the a_d values, with the a_d values themselves not being stored. This appears as follows:

<17,1><304,1><15,4><204,3><112,19><248,4><157,2><142,8><104,2>

Garcia et al. (2004) experimented with various query pruning techniques, again including the *quit* and *continue* strategies from Moffat and Zobel (1994). They also introduced a number of other strategies to take advantage of the access ordered index, in order to limit the number of postings being processed. The most successful of these was their *MAXPOST* scheme, in which a maximum number of postings are processed from each of the postings lists for each of the terms in the query. Interestingly this is the same scheme developed independently by (Blott et al., 2004; Ferguson et al., 2005b,c), although referred to as *top subset* retrieval. Garcia et al. also introduced an extension to this *MAXPOST* procedure, by which the posting lists can be pruned past the point of this MAXPOST (similar to Carmel et al. (2001b)), the postings lists can then be rearranged to the conventional document number ordered index (which is more suitable for compression). This results in a substantial reduction in the size of the index. However the drawback of this *access pruned* index is that the system is effectively limited to using this fixed MAXPOST size. Query processing using an access sorted index is shown in figure 3.8.

3.2.4 Index pruning

Another area of work relating to *reducing the search space* is *index pruning*, where pruning methods are used to remove the least important postings from the index. Pruning methods include stopword removal and static index pruning, both of which are explained below.

3.2.4.1 Stopword removal

An important issue when indexing a text collection is deciding what words are important, in order to represent the information contained within the documents.

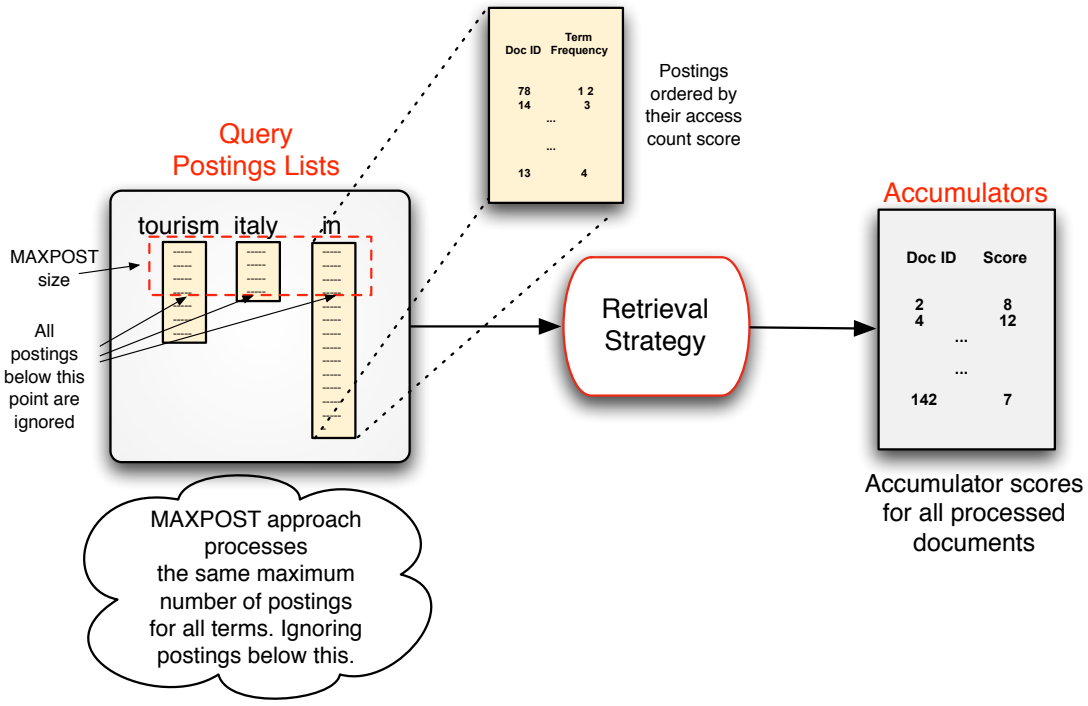


Figure 3.8: Query processing with an access sorted index

Much of the work, from an automatic text analysis point of view, is based upon work done by Luhn (1958). Here he proposes, "that the frequency of word occurrence in an article furnished a useful measurement of word significance". His assumption that the importance of a term is contained in the frequency of its occurrence is represented by the *idf* components in the calculation of the Vector-Space and Okapi BM25 formulae.

Good keywords are not the most frequent, or the least frequent, but rather those that occur a moderate number of times. This seems intuitive, as very frequently occurring terms, that occur in most documents do not offer much discrimination to any document that they occur in, and on the opposite end of the spectrum, extremely rarely occurring terms, particularly in large collections can often correspond to misspellings.

This curve demonstrates Zipf's Law (Zipf, 1932), which states that:

$$frequency(f) \times rank(r) = constant \tag{3.1}$$

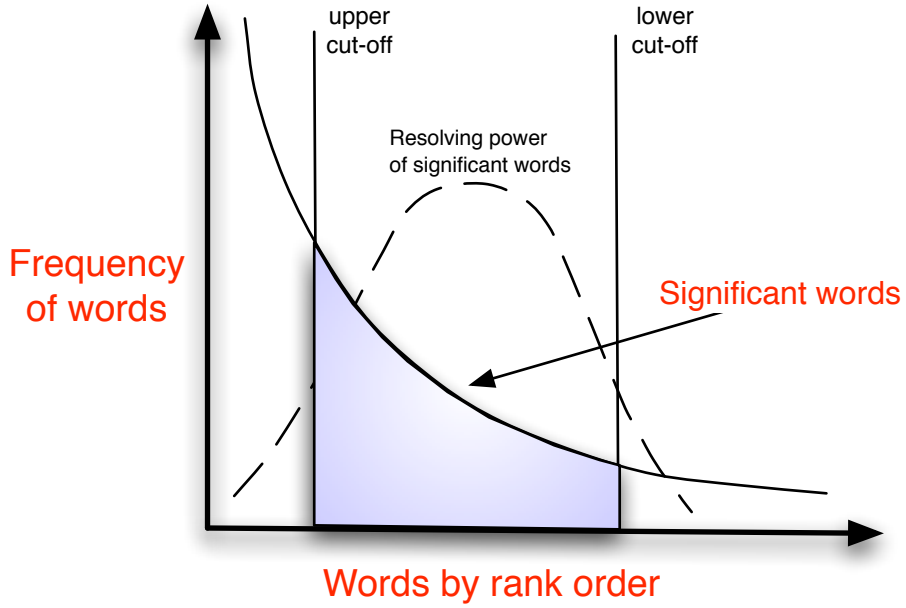


Figure 3.9: Resolving Power

Figure 3.9 shows two cut-off points, the lower being the point at which words become too rare as to be useful, and the upper cut-off being the point at which words become too common to be of benefit to the retrieval process. Since these frequently occurring words do not offer much in terms of discrimination between documents they are often removed from the index. These are known as *stopwords*, and since these are the most frequently occurring terms, their elimination can result in a significant reduction in index size, up to 40%, or more, according to Baeza-Yates and Ribeiro-Neto (1999). Common stopwords for a general English text collection should include prepositions and conjunctions, such as: "a, and, the, is, of", etc. The omission of these terms can have a positive effect on both the size of the index, as well as the time required to process a query (particularly with long, natural language queries). However it is also worth noting that the elimination of these stopwords from the index, can have a detrimental effect on retrieval performance. For example, a query that contains only stopwords, such as, "to be or not to be" will not return any documents if all these terms have been removed from the index. Figure 3.10 shows how this might affect the processing of the query, "tourism in italy", where

"in" is eliminated due being a stopword.

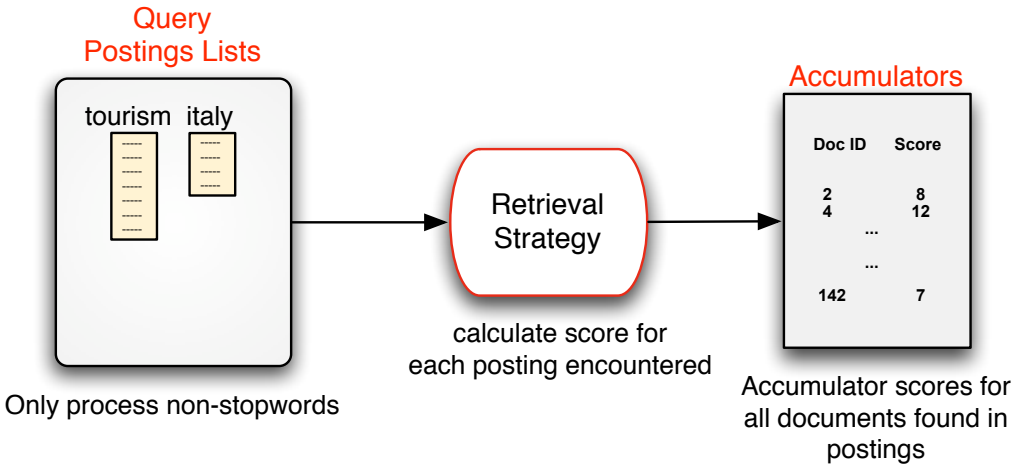


Figure 3.10: Query processing with stopword removal employed

3.2.4.2 Static Index Pruning

The aim behind static index pruning is to effectively reduce the size of the inverted index, in such a way that retrieval performance is not affected, or at least limited, and can essentially be viewed as a type of lossy compression. The main way in which this is done is to identify the postings that are unlikely to affect the accuracy of the system.

Based on the fact that a *scoring function* assigns a score to each document for a query, so that the highest scores are the most relevant, Carmel et al. (2001b) proposed two methods for static index pruning.

- **Uniform Pruning:** Their *uniform pruning* method removes all postings entries from the index whose score falls below a threshold, although for low scoring terms this may result in a large number, or all of their postings being removed.

- **Term-based Pruning:** A more advanced pruning method is their *term-based* approach, which assigns different thresholds for each term. This approach

guarantees that each term will have some representative postings. They try to execute the pruning in such a way that the top k documents returned to the user remain the same (or as unchanged as possible). They supply their algorithm with the variables k and ϵ, that control the level of pruning as follows:

For each postings list the top k postings of each list remain, as calculated by the scoring model used. The rest of postings are pruned if they fall below the threshold of of τ_t, where τ_t is:

$$\tau_t = \epsilon * z_t \tag{3.2}$$

where z_t is the score of the zth best entry in the postings list. Fundamentally, this means that the top k postings are kept, and then the rest are pruned, based on a percentage of the lowest score of the top k postings.

Based on experiments by Carmel et al. (2001b) it was found that the *term-based* approach performed best, however in Carmel et al. (2001a) they found that the *uniform-based* approach performed better in terms of similarity with the top 10 results of an unpruned index.

Stopword removal may also be though of as static index pruning, where the whole posting lists of certain terms, i.e. the stopwords, are removed from the index. As stopwords are perceived to be the least informative of the terms in the index, along with terms falling beyond the upper cut-off of Figure 3.9 from Luhn (1958), these terms occur so infrequently and rather than being of high value are actually more likely to correspond to misspellings. Similarly the postings lists of these terms may also be removed from the index, without any significant performance degradation.

3.3 Summary

As we have outlined in this chapter, there are various ways in which the search space may be reduced while providing the user with a list of the most relevant documents to their query (as defined by a certain retrieval strategy). Some of these, such as the frequency sorted (Persin, 1994; Persin et al., 1996), and access-ordered (Garcia et al., 2004) approaches utilise query-independent measures in order to sort the postings lists. However, these approaches usually utilise a single measure, derived by the distribution of terms within the documents. We wish to investigate the usefulness of various query-independent sources of evidence, both static and term-dependent measures to sort postings lists by. As described in chapter 2, there are certain linkage analysis techniques, such as PageRank, which are incorporated with term-based scores at query time in order to promote higher quality documents. We investigate the use of the PageRank measure, as well as other means of identifying quality documents, and incorporate these into the index ordering process.

Chapter 4 describes the importance of the use of quality measures in information retrieval, and shows how these may be useful for filtering documents in the search process, which we investigate through the use of a sorted index.

Chapter 4

Query-Independent Sorting

In this chapter we discuss the use of *search quality* and its application in IR, we introduce various query-independent measures that may be used to promote the more important documents in the collection, which allow less important documents to be filtered without adversely affecting the retrieval accuracy. We also explain how a sorted inverted index is created using these query-independent sources of evidence.

Quality information is constantly referred to as information that will satisfy the needs of the user. According to Strong et al. (1997), high quality data is that which is fit for use by the data consumers. Good quality data would therefore meet the requirements of its intended use, therefore it is relative, and dependant on the needs and knowledge of the user. Although this notion of quality information is relative to the user, as well as their information need, we still believe that there may be an inherent level of quality (or lack of quality) within all documents that will correspond to their likelihood of being relevant to a user. By measuring this level of quality within all documents we can then come up with a query-independent measure of quality that can be used to sort the postings in the inverted index. By sorting the postings based on this quality likelihood measure we provide a mechanism for limiting the number of documents that are evaluated during the retrieval process, while at the same time aiming to provide at least the same accuracy of results as a conventional index (processing all postings).

4.1 Search Quality

> "*Now it's a question of integrity - can we really trust our information?*", Kevin Mitnick (2004)

Information retrieval systems were originally developed to manage and retrieve information in libraries, and not only would the information within the libraries have been carefully selected, but document quality would also be inferred from the reviewing process used by the publishers. Therefore all the information would tend to be of a high quality, and so retrieval approaches based purely on term distribution statistics were sufficient. However this is not the case with large collections of web documents such as the World Wide Web, where no reviewing process is needed in order to publish a document. Not only that, but as many of the queries issued to web search engines may have hundreds of thousands of returned documents (which may vary greatly in terms of quality), it seems intuitive that the use of information regarding the quality of documents should be of benefit in a ranking process in order to promote higher quality results.

In an effort to have their web documents ranked more highly by search engines, some malicious users may create what is known as *spam* documents. For commercial web sites in particular, an increase in search engine referrals translates into an increase in revenue. Since 80% of web searchers view no more than the top 10 to 20 results (Jansen and Spink, 2003), it is advantageous for a successful commercial web site to achieve a high ranking for user searches. Not surprisingly many operators make use of *Search Engine Optimisation* (SEO) techniques in order to gain a higher ranking. Some of these techniques use legitimate means, such as improving the quality of the content, whereas others use unethical approaches in order to achieve a higher ranking, this is known as *web spam*. With the first generation of search engines using document content alone to rank documents, this involved interfering with the frequency of which certain words appear, as well as introducing new words to the document, in a process known as *keyword-stuffing*. This would result in a

better ranking using term-based ranking algorithms such as the vector-space model, as well as the Okapi BM25 algorithm (discussed in Chapter 2). As search engines began to incorporate linkage analysis techniques in order to identify popular pages (based on the linkage structure of the web), SEO techniques were also developed in order to artificially boost the popularity of a page. These work by creating non-relevant pages and linking these to the target document, in a process known as *link-stuffing*. Given that the amount of spam on the web is estimated at 13.8% for English-language pages (Ntoulas et al., 2006), the identification of such pages can save a large amount of search engine resources (in terms of indexing and retrieval), as well as improving the quality of the results produced by the system.

The fact that many documents are not returned in response to users' queries was illustrated by Garcia et al. (2004), when they ran 1.9 million queries on a collection of around 1.6 million documents (from the WT10g Web Track collection, (Hawking, 2001), which would most likely contain no spam content). They concluded from their experiments that document accesses are highly skewed, and that there is a correlation between the access count and the likelihood of being relevant to a query. We suggest that this would become even more pronounced as the size of the collection increased, also taking into account the presence of spam documents that are present in the Web and were not present in this test collection.

Without the search system taking into account the quality of documents, it is necessary for Web users to make these judgments of quality and authority for themselves on results returned from a Web search engine. In order to reduce the cognitive load of this process for users, and to present users with quality information, we suggest to firstly investigate how to measure this concept of quality in information retrieval. Being able to identify quality documents then allows us to reduce the search effort of the IR system, by only searching higher quality documents, yet aiming to return the same level of retrieval effectiveness for the user.

4.1.1 Measuring Quality of Information

The notions of quality and authority have been discussed under various forms, by a number of relevance criteria studies; "goodness" (Cool et al., 1993), "perceived quality" (Park, 1993), "actual quality", "expected quality" (Wang and White, 1999) and "authority" (Cool et al., 1993; Wang and White, 1999; Cooke, 2001), although any assessment of quality is dependent upon the needs of the individual seeking the information, as well as on the nature of the source being evaluated.

While the quality of a web site is a matter of human judgement, there are major factors influencing the notion of quality. The following subsections outline various factors that can be considered in a query-independent manner in order to determine the quality of a document.

A lot of work has recently focused on the use of link analysis algorithms to identify high quality documents, with Page et al. (1998), Kleinberg (1999), Chakrabati et al. (1999), Bharat and Henzinger (1998) and Lempel and Moran (2001), all using the linkage structure of web documents. The aim of such linkage analysis measures as PageRank (Page et al., 1998) is to "measure the relative importance of web pages", or in other words their relative quality.

There are however other methods that can be used to identify quality documents, these use other static features of documents, such as the URL length, document length, number of in-links, number of visitations, etc. We propose that these may be used as single measures of the quality of a document, and that these can then be combined with multiple measures of quality, in order to produce a better overall measure of quality for each document in the collection.

Zhu and Gauch (2000a) identify and investigate six quality metrics to incorporate into the retrieval process:

- *Currency*: how recently a document was last modified (using document time stamps).

- *Availability*: how many links leaving a document were available (calculated as

the number of broken links from a page divided by the total number of links).

- *Information-to-noise*: a measurement of how much text in the document was noise (such as HTML tags or whitespace) as opposed to how much was useful content.

- *Authority*: a score sourced from Yahoo! Internet Life reviews and ZDNet ratings in 1999. According to these reviews each site was assigned an authority score. Sites not reviewed were assigned an authority score of zero.

- *Popularity*: how many documents link to the site (in-degree).

- *Cohesiveness*: how closely related the elements of a web page are. This was determined by classifying elements using a vector space model into a 4385 node ontology and measuring the distance between competing classifications. A small distance between classifications indicates that the document was cohesive. A large distance indicates the opposite.

Zhu and Gauch (2000a) evaluated performance using a small corpus, with 40 queries taken from a query log file. They observed some improvement in mean precision, based on all the quality metrics (although not all improvements were significant). The smallest individual improvements were for Popularity and Authority (both non-significant). The improvements obtained through the use of all other metrics were significant. The largest individual improvement was observed for the Information-to-noise ratio. Using all quality metrics, apart from Popularity and Authority, resulted in a (significant) 24% increase in performance over the baseline document ranking.

Amento et al. (2000) evaluated a number of link and content-based algorithms, using a dataset of web documents rated for quality by human topic experts. They found that "three link-based metrics and a simple content metric do a very good job of identifying high quality items." However, surprisingly they found that indegree performed at least as well as the more advanced HITS and PageRank algorithms,

as well as a simple count of the pages on a site proving to be as successful as any link-based methods.

Having identified whatever quality metrics for use, there is then the issue of how to integrate these into the IR system, which we look at now

4.1.2 Incorporating quality metrics into ranking

Taking linkage based measures of quality such as PageRank and HITS into account, there has been a considerable amount of work concentrated on the improvement of the these measures in terms of accuracy, stability and efficiency (Bharat and Henzinger, 1998; Haveliwala, 2003, 2002; Richardson and Domingos, 2002; Ng et al., 2001; Lempel and Moran, 2001), although there has been relatively little work done on how to effectively combine this with term-based (baseline) results at query time. This is not something that is addressed in the original PageRank publications (Brin and Page, 1998; Page et al., 1998). While investigating the effectiveness of using link and content-based measures to identify high-quality documents, Amento et al. (2000) also do not discuss how they combine these content and link measures. One suggested way in which to combine these link-based scores with a baseline score is to add the baseline scores with a normalised link-based score (e.g. PageRank), as suggested by Richardson and Domingos (2002).

The main advantage of incorporating measures of quality into the retrieval process is to provide the user with relatively high quality results in response to their query. This may address the situation where the user has to search through large amounts of low quality documents, while higher quality documents may lie lower down in the result list. It seems that the type of query that should profit most from the incorporation of these additional features, are queries that are broad in nature. These by their very nature should return a large number of results, particularly when dealing with a large collection such as the World Wide Web, where document quality may vary considerably. On the contrary, a narrow query, which is quite concise in its nature should return a much lower number of documents (which also

may vary in quality), however as there may be relatively few relevant documents, and if the query is quite concise it would be better to take more influence from the content score. We may then choose to regulate the influence that we assign to a quality-based score depending on the type of query that is submitted. Kleinberg (1999) highlighted this, suggesting that there are two types of queries: *broad* and *narrow*. The narrow queries suffer from the *scarcity problem*, in that there may be very few documents containing the required information. Broad queries suffer from the *abundance problem*, and so the number of documents returned is too great to expect a human to process. He suggests that these broad queries are best suited for the incorporation of authoritative ranking.

Having introduced the concept of using quality measures in an IR system, the following section discusses in more detail some of the *query-independent evidence* that can be used to measure the quality of information (for various user information needs).

4.2 Quality Measurement Using Query-Independent Evidence

Most of the work carried out in IR ranking has focused on improving the results returned to the user through query-dependent means, such as the vector-space model, BM25, etc. (see section 2.1.3). However it is also important to have a good *query-independent* ranking algorithm. This provides a query-independent measure of importance for each of the documents in the collection, which is important for a number of reasons, including providing a means by which to identify high quality documents. PageRank (as mentioned in section 2.2.4.1) is one of the prominent static measures for ranking documents in an IR system. Despite the success of the Google search engine (Google Inc., 2006), which claims to incorporate PageRank into their ranking, there has been little independent evidence to verify the benefits of its inclusion in the ranking process. Within the TREC experiments, PageRank has not performed

as well as expected (Hawking and Craswell, 2005), where simple measures such as the indegree of a page were found to be at least as effective for the task of homepage finding (Upstill et al., 2003), and it was also found that features such as the number of pages on a site were as effective as PageRank for identifying high quality documents (Amento et al., 2000).

An IR System may use a variety of different static measures to help improve its performance, and much of the benefits of the inclusion of certain static measures can be seen for example in the homepage finding task in TREC. Here, such static measures as URL depth, as well as the number of incoming links of a page have been shown to aid performance. The advantage of these measures for a homepage finding task seems intuitive, as the relevant documents for this task (i.e. homepages) will mostly come from the root of the URL, and so URL depth should act as a good discriminator between relevant and non-relevant documents. Also the number of incoming links tend to be larger for homepages, mainly because more people tend to link to the homepage of an organisation, as it provides a good entry point for their site. However this may not be the case for an ad hoc search task, where relevant information may be found at different locations on a web site (other than at the root), and so selecting static features that give such a high level of discrimination between relevant and non-relevant documents is more difficult.

There are however certain static features that may be useful in identifying documents that are likely to be returned in response to a user's query. Depending on the query-dependent retrieval mechanism used, there are a number of features used in the calculation of a similarity score between a document and a query, for example the document length, the within-document term frequency, and the inverse document frequency (idf) (giving a measure of importance for a term within a collection). Based on a query-dependent method for identifying documents, if we can identify the documents that are most likely to be relevant to any given query with a certain degree of confidence, then we can use this as a means to promote documents that are likely to be relevant. These features may not directly correspond to the

relevance of a document, but should correspond to the likelihood that a document is returned by a search engine.

If for instance we use a BM25 formula to calculate a list of candidate results in response to a user's query, then documents which have a high BM25 score (taking all their terms into account) are more likely to appear in this list than those with a low overall score. We can therefore assign a static score to each document in the collection, based on their total BM25 score for all the terms within that document. This should correlate with the probability of that document occurring within the result list in response to a query, and this is essentially what Garcia et al. (2004) found.

The effectiveness of incorporating certain static measures into the IR process, may also be limited by the overall quality of the collection itself. This would seem natural, as query-independent measures that promote high quality documents can no longer provide much discrimination (among a set of documents), where the large majority of the documents are of high quality. Therefore, we would suggest that most gain can be achieved by incorporating these static measures on a collection which contains a considerable amount of low quality documents. We believe that the Web (which is estimated to contain 13.8% spam documents for the English-speaking documents (Ntoulas et al., 2006)) would be suitable for the incorporation of these measures, provided of course that these spam, or low quality documents can be identified with an acceptable level of confidence.

In this section we describe various query-indepepent methods that may be used to identify these more important documents.

4.2.1 Linkage Analysis

Two well known measures of the popularity of a document, based on the hyperlink structure between documents are:

- **Indegree:** the *indegree* of a document is a count of the number of documents

that link to it. Founded on the assumption that a link between two documents carries an implicit vote for another document, this is a simple (and direct) measure of popularity for a document.

- **PageRank:** as discussed in Chapter 2, PageRank is a more advanced method to calculate the popularity of a document, based on the linkage structure between documents.

Both these measures provide a query-independent measure of popularity for a document, based on analysis of the linkage structure connecting documents. These popularity measures do not directly model the likelihood of relevance: the most popular document in the web is not necessarily the document that is most likely to answer a user's information need, although there may be some inherent information contained within these measures of quality that may correlate with relevance.

These measures follow a *power-law* distribution: A power law relationship between two scalar quantities x and y is one where the relationship can be written as

$$y = ax^k \tag{4.1}$$

where a (the constant of proportionality) and k (the exponent of the power law) are constants, which can be seen as a straight line on a log-log graph. This shows that highly popular documents occur extremely rarely, while unpopular document occur the most frequently, this is illustrated in Figure 4.1 and Figure 4.2.

This distribution of document scores shows that a large portion of the documents have quite low scores, while there are a small number of documents with very high scores. In order to try and reduce the dominiating effect of these few very high scoring documents (when combining with other sources), these scores are quite often normalised in some way, by taking the log of their original score for example.

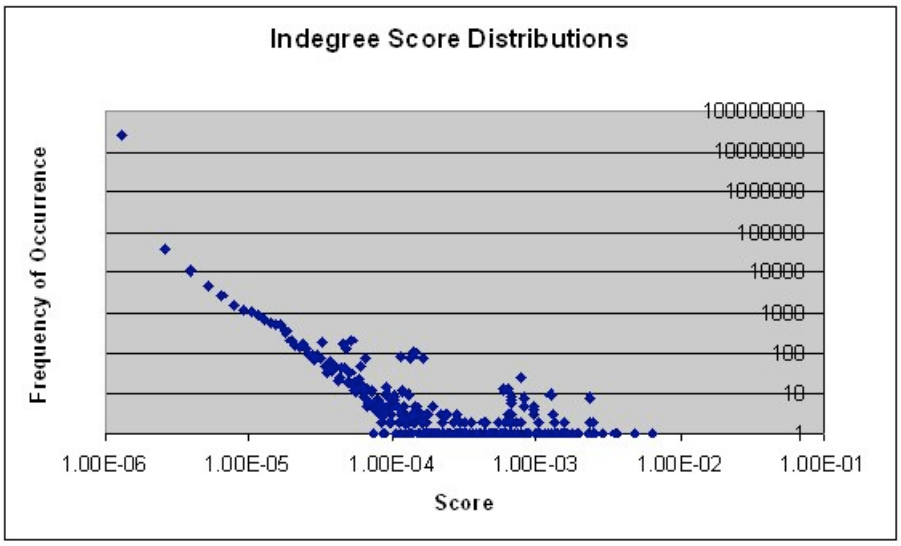


Figure 4.1: Distribution of indegree scores from the GOV2 collection on a log-log scale (scores normalised between 0 and 1)

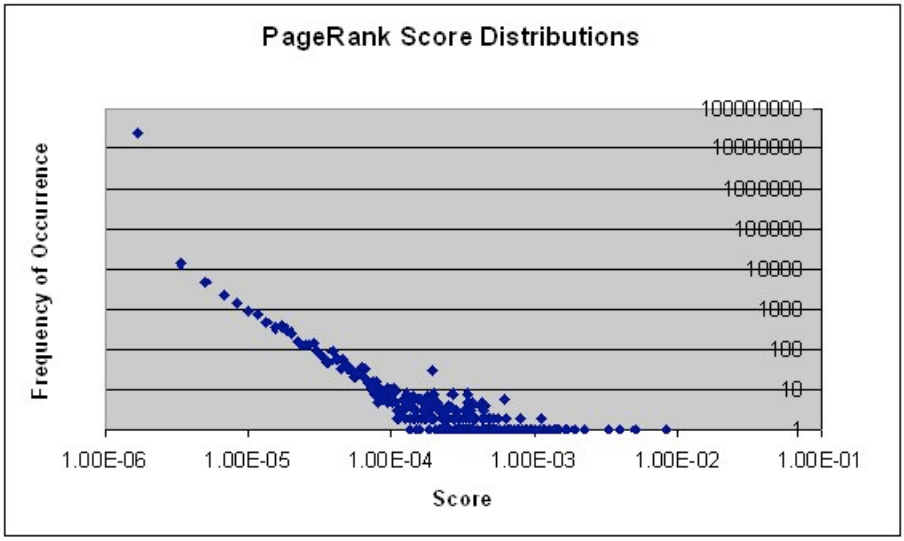


Figure 4.2: Distribution of PageRank scores from the GOV2 collection on a log-log scale (scores normalised between 0 and 1)

4.2.2 Access Counts

As described in Chapter 2, Garcia et al. (2004) proposed a measure known as access counts, as a means of determining the likelihood that a document would be returned in response to a query. These access count scores were generated for each document by the number of times it had previously occurred in a result list (in response to a

large number of user queries). We would also promote the use of this measure as a means to sort an inverted index by. Although we would suggest to combine this measure with others, in order to provide a richer representation of the documents in the collection, and then use this combined measure as a means to sort the inverted index by.

4.2.3 Information-to-noise ratio

The information-to-noise ratio is computed as the total length of the number of terms in the document after preprocessing, divided by the total length of the document (Zhu and Gauch, 2000b). It is believed that a document with a lower information-to-noise ratio score is generally of lower quality than a document with a higher score. This may (for example) be a result of a small number of words in a document that contains a table, as well as other HTML markup, and consequently this document would not contain much information for a user.

4.2.4 Document Cohesiveness

Document cohesiveness measures how closely related the information within a document is. Information may become incoherent with the incorporation of irrelevant details, which may lead to confusion for the reader. Although incoherence may occur naturally within a document, a high level of incoherence may indicate that a document is not of particularly high quality, and a user may find it difficult to find the information they are seeking surrounded by unrelated information.

4.2.5 Spam

Not all web document authors are truthful about the content of their web pages, some insert false content in order to gain increased exposure on the web. An author may manipulate the content of the document, for example by inserting a hidden dictionary of words in the text, so that the document would become relevant for

more keyword searches. This is known as *Search Engine Persuasion* (Marchiori, 1997). They may also manipulate the hyperlink structure of the web by linking to many other spam documents, in order to increase their ratings for measures such as PageRank (Page et al., 1998). All this means that although these documents may appear highly relevant using standard IR approaches, they are in fact (generally) highly irrelevant to users.

These spam documents are unlikely to be relevant for most users' information seeking needs. Therefore a static spam feature, which can give a certain score to each document, based on the prospect of each document being spam, should be of use in order to promote non-spam documents, as well as decrease the influence of spam documents. Provided the collection itself contains a sufficient level of spam documents, then a static feature that can identify these should be useful in aiding an IR system in not returning these to a user.

4.2.6 Click-Through Data

For a typical user query issued to a web search engine there may be thousands of results returned to the user, however the user may only be interested in one, or a very small number of these. As "users do not click on links at random, but make a (somewhat) informed choice" (Joachims, 2002), and although these clicked-on documents may not provide a perfect annotation of relevance (between a query and a document) they do provide useful information that may be used in order to improve the results presented to a user. A click-through is recorded when a document in the result list (returned by an IR system) is clicked on by a user. This click indicates that the user in making an informed decision to click on this document, and so one simple query-independent way of assigning document importance is to assign a score to each document based on the number of click-throughs they have accumulated.

4.2.7 Visitation Count

The visitation count of a document is the count of the number of times that a document is visited. This differs from PageRank and other linkage-based measures (which calculate popularity based on the linkage structure between documents). The visitation count is the actual number of visitations that a document receives. This is also different from click-through data, as a click-through is only registered when the document is accessed through the result list returned by an IR system. A visitation count corresponds to the total number of times that a document is viewed, and not necessarily coming as a result of a search request. This is quite a valuable source of information regarding the popularity of documents, however is also quite difficult to obtain. Visitation counts can generally only be obtained from internet providers, proxy servers, web browsers or operating systems, and due to the potential sensitive nature of this type of private information it is generally kept confidential. Richardson et al. (2006) used this type of information, combined with other features, in order to get improvements over PageRank; they were able to acquire this data from the MSN toolbar, which recorded the documents that were requested by users.

4.2.8 Document Structure and Layout

One other important aspect of document importance (that is often overlooked from a IR point of view) is the structure and layout of a page, and how this relates to the quality of a document. We may consider documents that follow a certain text layout (possibly accompanied by relevant images) to be a higher quality document than one with the same content in a less well organised fashion. Hill and Scharff (1997) discuss the readability of web sites with different foreground/background combinations; this may also be thought of as a quality metric. We may also consider the ease of navigation within a web document, as well as any other aspects that allow for the easy of access and processing of information from a user perspective.

If we can identify these traits in documents, these may be used in a query-independent manner to improve the quality of the search results. If for instance the user enters a sufficiently broad query (which is likely to have many relevant documents), why not provide that user with a high quality, aesthetically pleasing, and easy to navigate document. In other words, if the system is confident in fulfilling the user's information need, then it may also consider providing them with a document that presents this information in the best possible manner, in order to further reduce the cognitive load on the user. Therefore the system will not only have the difficulty in deciding how to define these measures, and how they are applied, but also how these should be integrated into the search process, in such a way as to not degrade its retrieval performance.

4.2.9 HTML Correctness

There may be a certain level of detail contained within the HTML structure of a web document, that may provide an insight into the level of quality of that document, in a query-independent way. Perhaps if a document is compatible with the HTML specifications (The World Wide Web Consortium, 2007), this gives an indication of a higher quality document. Alternatively, if a document contains errors, such as wrongly named tags, this could perhaps indicate that the document is of low quality.

Also the inclusion of broken, or dead links, that no longer point to a valid target, may also indicate that little or no care has been taken with the upkeep of the document, and so implying a lower level of quality than a document which links to active documents on the web.

Possibly, if more care is taken with the creation of the HTML, then there may also be care taken with the contents of the document. This may indicate that the source is reputable, and should be treated as such by the IR system.

4.2.10 Document Currency

Document currency deals with how current, or up-to-date a document is. In its simplest form this can take into account the date each document was last updated. Depending on the information need this can be something that may be of benefit in providing the user with more suitable information. For instance, if a user is seeking information on a very recent news topic, then a document which was last modified today would tend to be more likely to be useful than a document last modified two years ago. Therefore, depending on the search task the user is involved in, we may want to weight this measure differently. A more advanced approach could track the actual changes made, rather than basing the currency purely on the modification date (as the changes may be such that the main content may not have significantly changed).

4.2.11 URL Information

Some important information can be extracted from the URL of a web document. The hierarchal structure that is inherent in the URL is often ignored, even though this is quite important and is often viewed as being closely related to the link structure of the web (Eiron and McCurley, 2004; Xue et al., 2005). Due to this hierarchal structure that is used to organise documents, the documents on the root of directories can be seen as being more general (in terms of content) than those found at the lower levels. Similar to the Web's linkage structure, there is a certain amount of latent human judgement that can be extracted from the URL of a document.

One simple query-independent measure, derived from a web document, is its URL depth. Consider the URLs:

"*www.nasa.gov/index.html*"

and

"www.nasa.gov/multimedia/imagegallery/index.html"

We may view the former (with a URL depth of 1) as being more authoritative (within the overall context of the Web) than the latter (with a depth of 3). Although the second document may indeed be more relevant for a more concise query concerned with images and multimedia content from NASA, the first page should be considered more authoritative. For this reason we may use this to assign an importance measure to each document, derived from their depth within the URL. Another variation on this would be to use the length of the URL in characters, rather than dealing with the URL depth, based on the number of "/"'s.

4.2.12 Term-Specific Sorting

Unlike other query-independent measures, which are also static and constant for all terms in the collection, there are other measures with are again query-independent, however each document may have a different value (or score), depending on the term. For instance, a simple measure of significance for a document, on a term by term basis is the number of times that term occurs in the document.

Term Frequency

A simple measure to judge the significance of a document (for a specific term), is the number of times that the particular term occurs within each document. As explained in Chapter 3, Persin (1994) and Persin et al. (1996) used this measure to sort the postings for each term.

BM25

A more complex way to measure the importance of a document (on a term by term basis) is to use a term weighting approach, such as BM25. As discussed in Chapter 2, for an adhoc retrieval and ignoring any repetition of terms in the query (as is the case for the vast majority of web queries) this function can be written as:

$$BM25(q,d) = \sum_{t \epsilon q} log\left(\frac{N - df_i + 0.5}{df_i + 0.5}\right) \times \frac{(k_1+1)tf_i}{k_1((1-b) + b\frac{dl}{avdl}) + tf_i} \tag{4.2}$$

where df_i is the number of documents in the collection that contain the term i, and k_1 , k_3 and b are parameters. This calculates a similarity score between a query and a document. With this formulation the scores for all query terms are summed to give a final score, however to calculate a weight or score on a single term i, for a document i, without considering any query terms that may be used, we may use the following formula:

$$BM25(q,d) = log\left(\frac{N - df_i + 0.5}{df_i + 0.5}\right) \times \frac{(k_1+1)tf_i}{k_1((1-b) + b\frac{dl}{avdl}) + tf_i} \tag{4.3}$$

When applied to each document (in each term's postings list separately) it provides a pre-calculated BM25 ranking for each posting list. For a single term query this will provide the same ranking of documents as that generated by BM25 at query time. The only missing element from calculating a full query-dependent score (for multi-term queries) is the summation of each query term's scores. This of course is something that can only be done at query execution time, as the query terms are unknown. Therefore it is as close as we can get to a query-dependent estimation of the query-dependent BM25 score.

Also, if we are only concerned with calculating a score, that is only to be used as a means by which to sort postings, and this is done on a term by term basis, there is then no need to include any global term information, such as df_i (the number of documents in the collection that contain the term i). This leaves the simplified calculation of:

$$BM25(d,i) = \frac{(k_1+1)tf_i}{k_1((1-b) + b\frac{dl}{avdl}) + tf_i} \tag{4.4}$$

which can give a relative measure of importance, for each document, in each postings

list.

4.3 Sorting Inverted Index Using Query-Independent Evidence

Similar to the approaches described in Chapter 3, we advocate the use of a sorted index. All these types of sorted index use (to some extent) a term weighting based measure of query-independent evidence, which is likely to be related to the way in which the retrieval method being used performs:

- Term frequency sorted: sorts the postings lists based on the within-document term frequency of each posting, which is a very simplistic term weighting method.
- Impact sorted: sorts the postings lists based on an impact value, which is a term weighting based value.
- Access count sorting: the access counts themselves are a direct result of the retrieval process used by the search engine to generate the access counts, and so would be most likely to work optimally if the same retrieval process is employed on the access count sorted index, as was employed to generate those access counts.

What we wish to investigate is, the usefulness of different query-independent measures for sorting an inverted index. We then want to combine different query-independent measures together in order to provide a richer representation of each of the documents and hopefully a better indication of their query-independent quality. If we are able to produce a measure that is successful at indicating a documents query-independent quality we believe that this should provide us with a useful means by which to sort an inverted index, so that as we choose to process less postings for each query it should be the lower-quality documents that are eliminated and so

any loss in retrieval effectiveness will be minimised. Our goal is then to produce the sorting which can minimise the drop-off in retrieval accuracy as we process less postings in order to retrieve documents in response to a user's query. In this way we hope to maximise both the system's *efficiency* as well as its *effectiveness*.

4.3.1 Index Creation Process

To create a sorted index based on the discussed query-independent measures, the approach is fundamentally the same, whether sorting based on the static query-independent measures (such as PageRank), or using a term-specific sorting. The only difference is the stage (during indexing) at which the sorting measure is generated:

- **Static measure:** when using a static measure (where each of the documents in the collection have just one single value, which does not change), these scores are generated once (at a separate stage to indexing). These scores are then fed into the indexer (at indexing time), allowing each postings list to be sorted using a particular static measure. This process is illustrated in Figure 4.3.

- **Term specific measure:** when creating a sorted index using term specific weights, these weights are calculated on a term-by-term basis during the indexing process, as detailed in Figure 4.4 (for a more detailed view consult Figure 4.5).

The indexing system firstly processes the documents, so that the term information from all documents are extracted from all documents, as shown in stage 1 in Figure 4.5. This information is eventually stored in the form of postings lists for each term and so as a first step towards this we extract all terms, as well as other information associated with that term that is needed for the final inverted index, including the document where it occurred, the number of times it occurred within the document, as well as possibly positional information (if it is required for the retrieval being provided by the system.) This information is stored in temporary

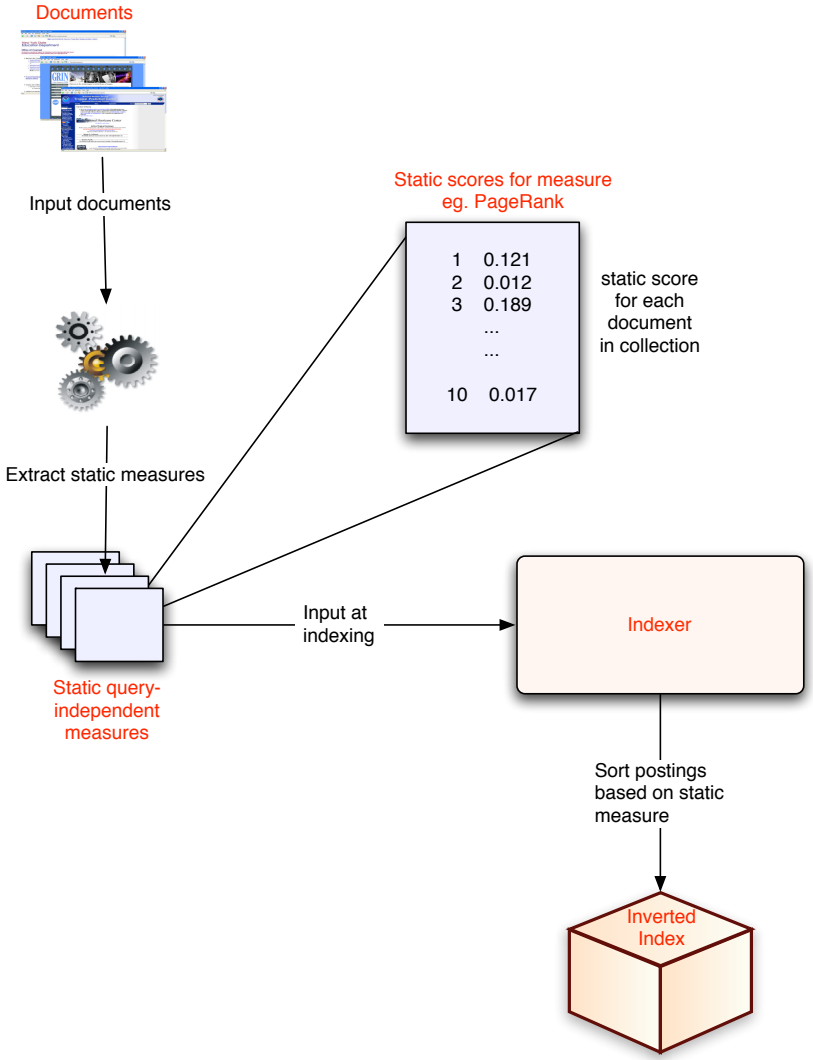


Figure 4.3: Creating a sorted index using static query-independent evidence

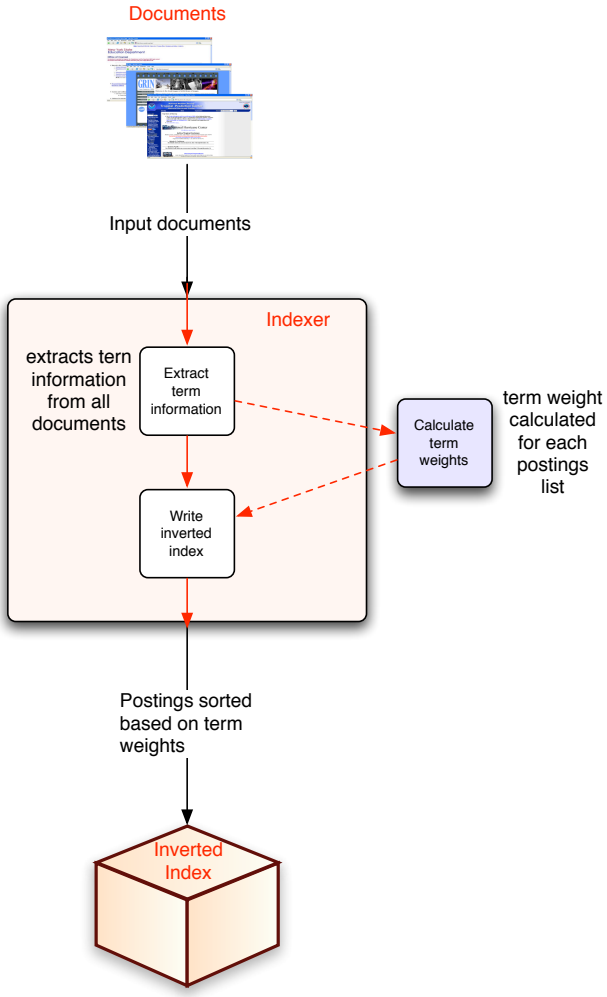


Figure 4.4: Creating a sorted index using term specific query-independent evidence (detailed)

files, that are then sorted (as in stage 2) so that all information for each term can be easily grouped together (as in stage 3).

At this stage a conventional inverted index can be constructed by taking the information associated with each of the terms separately and creating postings lists that are ordered by the document identifiers. However we can also choose to sort each of the postings lists based on the same static document weights (instead of the document identifier), or we can calculate term weightings for each term separately and sort based on these (as shown in stage 4 of Figure 4.4).

This inverted index, which is sorted using a particular query-independent measure appears similar to a conventional inverted index, except that the postings within the postings lists are sorted in descending order of the query-independent measure that we have chosen.

Again taking our previous example of a conventional postings list (from Chapter 2 in the form <d, $f_{d,t}$>:

<15,4><17,1><104,2><112,19><142,8><157,2><204,3><248,4><304,1>

and look at how this is changed if it is ordered by an example query-independent measure such as PageRank. Giving the documents in the postings list the following PageRank scores (shown in the form: document identifier (score)): 15 (0.01), 17 (0.3), 104 (0.02), 112 (0.03) 142 (0.08), 157 (0.05), 204 (0.1), 248 (0.07), 304 (0.06). If we order the postings list based on these scores it becomes:

<17,1><204,3><142,8><248,4><304,1><157,2><112,19>)<104,2><15,4>

The query-independent scores that are used to sort the index are not stored in the index, they are only used for the sorting process. This sorted index approach has the disadvantage that it does not allow the d-gaps between the document IDs to be stored (instead of their actual IDs), as the postings are not necessarily in order of their document IDs, although with this type of index the most important postings (as defined by the sorting measure, e.g. PageRank) are at the top of each postings list. Then, provided that the measure we use to sort the index produces a good indicator of the documents' importance, then we can choose to only process a

limited number of these postings from the top of each query terms' postings list. The more accurate the measure used to sort the index is at predicting the likely relevance of a document, the less the drop in performance will be, even as we process only a small number of postings from each postings list.

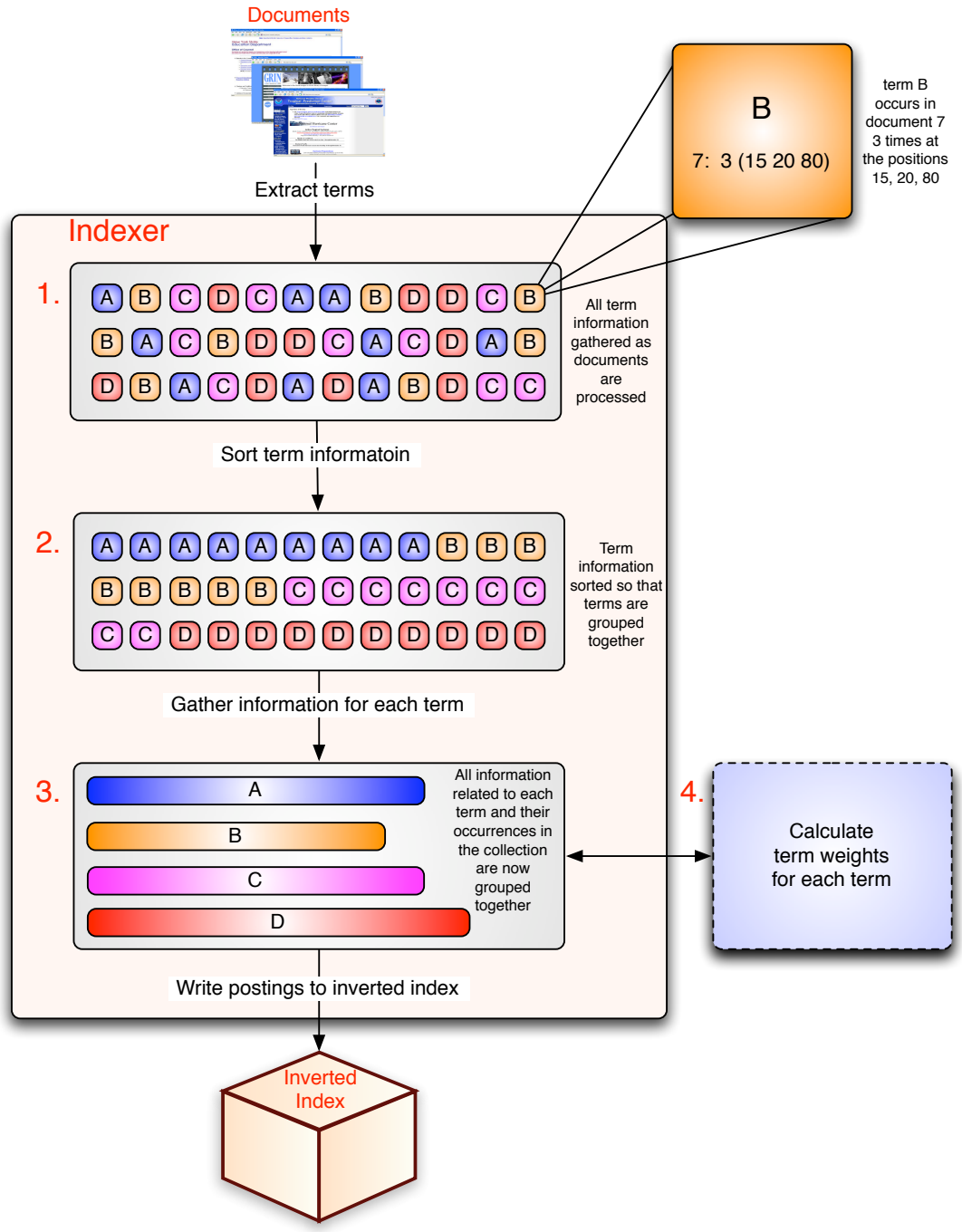


Figure 4.5: Creating a sorted index using term specific query-independent evidence

4.4 Summary

In this Chapter we outlined the main objectives of this thesis, to investigate the usefulness of using various different query-independent measure in order to effectively sort the postings lists within an inverted index – in such a way as to promote the high quality documents in the collection, so that if we choose to process less postings at query-time, then less of a drop in query-effectiveness will be incurred. We discussed the use of quality in information retrieval and suggested a number of different query-independent measure that may be useful to identifying, and promoting high quality documents. We also discussed in detail the way in which these sorted indexes are created.

In the following Chapter we investigate how to combine query-independent measures together in order to provide an improved single measure from several different measures. Then in Chapter 6 we anlayse the effectiveness of different query-independent measures (as well as the combined measures) in sorting the postings lists.

Chapter 5

Combining Sources of Evidence

"The whole is greater than the sum of the parts", Aristotle (informally attributed)

Data fusion is concerned with the combination of different sources of evidence. Data fusion has been applied in various different domains (L. Valet, 2000), in our case the data we are concerned with fusing is the different sources of retrieval evidence. This combination may take results from multiple search engines (as is the case for *meta-search engines*), or from within a single search engine architecture, where there may be ranked results generated from multiple representations of the same document, such as document text, titles, anchor text or linkage structure. Each of these may produce a quite different ranking of documents, and as with combining any sources of information, the goal is to gather all these sources together and use them to produce a more accurate final result. A basic overview of this process is illustrated in Figure 5.1.

Part of the gain achieved by using fusion in retrieval, is due to the increased recall resulting from different documents being returned by using: different document representations (Croft and Harper, 1979b; Das-Gupta and Katzer, 1983), different weighting schemes (Lee, 1995), as well as different search systems (Harman, 1993; Voorhees and Harman, 2000). Although having introduced more documents from different sources it is then the function of the fusion process to identify the documents which are most likely to be relevant. The main difficulty when fusing multiple sources

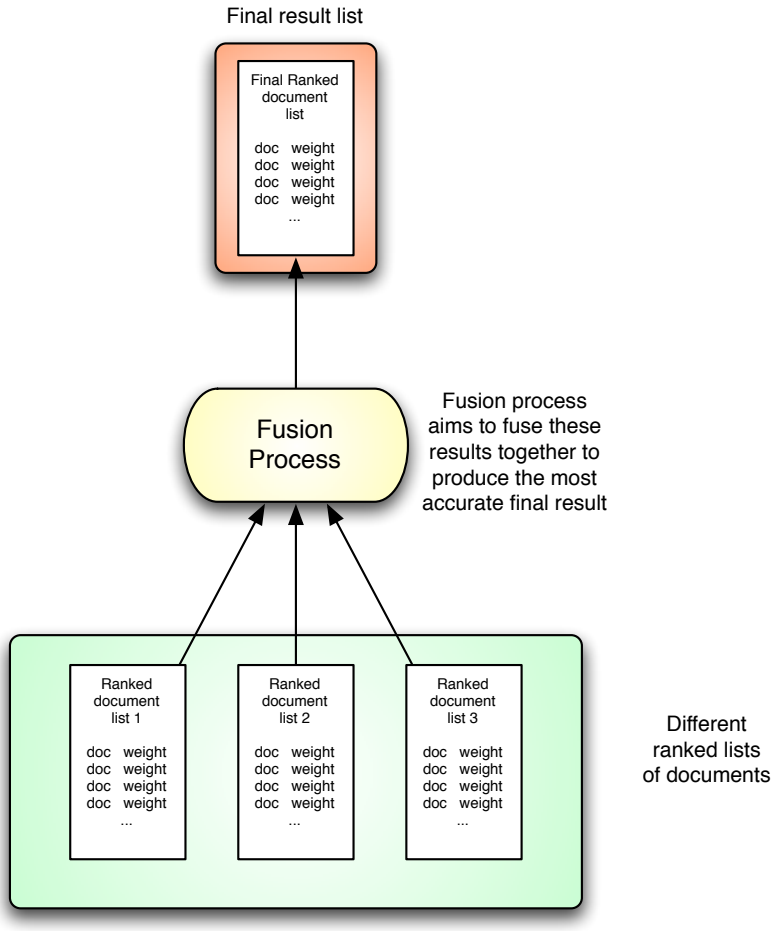


Figure 5.1: Basic overview of the fusion process

of evidence together (in retrieval) is in choosing how these sources are to be combined together in order to produce the best results.

In this chapter we examine different methods that have been applied to data fusion for information retrieval. We also show how these can then be applied to combining query-independent sources of evidence, such as those discussed in Chapter 4.

5.1 Score and Rank Based Fusion

"There are two main classes of meta-search fusion algorithms: ones that use scores from systems and ones that do not" (Ogilvie and Callan, 2003). In this section we outline some of the most popular methods for combining retrieval rankings, based on the individual sources' scores and ranks.

5.1.1 Similarity Merge

Fox and Shaw (1995) proposed a number of fusion methods based on the unweighted min, max or sum of each document's normalised score and later Lee (1997) considered the case where the rank has been used in place of the score. The two most successful of the proposed methods of combination introduced by Fox and Shaw (1995) are CombSUM and CombMNZ, which calculate a combined score for a document d, from a number different data sources:

CombSUM:

$$Score(d) = \sum_{i=0}^{n} Score_i(d) \tag{5.1}$$

where n is the number of data sources that are to be combined.

CombMNZ:

$$Score(d) = (\sum_{i=0}^{n} Score_i(d)) \times k \tag{5.2}$$

where k is the number of times where $Score_i(d) > 0$

These basic forms of combination have been used extensively in IR for the purposes of combining various forms of information. These types of combination are generally used to combine only the top n documents (and their corresponding scores) from the different results sets that are being combined. For this reason the CombMNZ method may penalise documents that do not occur in one or more of the result lists. However, for the purpose of generating a combined static score, where all documents in the collection have a score, then this approach will produce equivalent results to CombSUM (as all documents in the collection have a corresponding score and so none are penalised). For this reason we consider only the CombSUM method in our experiments in Chapter 6

5.1.2 Score Normalisation

When combining different sources together using scores as a means to determine the most effective source, this may be prone to error. For instance, suppose we wish to combine two ranked lists, one whose scores are in the range 0 to 0.8 and the other whose scores are in the range 10 to 20,000, as shown in Figure 5.2. Clearly using an approach such as CombSUM (shown in equation (5.1) above) the second source will dominate the resulting combination, solely because of its higher scores. If the scores of these two non-homogenous sources are generated using different approaches then the difference in these two score ranges may be as a result of this, and may not be an indication of the superior source. It is because of this that the sources being combined should be *normalised* in some way, in order to provide a "level playing ground" for all of the sources.

One such way to normalise the scores is to use the *min-max* approach, which shifts the minimum score to 0 and scales the maximum to 1; the new scores $\{s'_i\}$ are calculated from the original scores $\{s_i\}$ as follows:

$$s'_i = \frac{s_i - min\{s_i\}}{max\{s_i\} - min\{s_i\}} \tag{5.3}$$

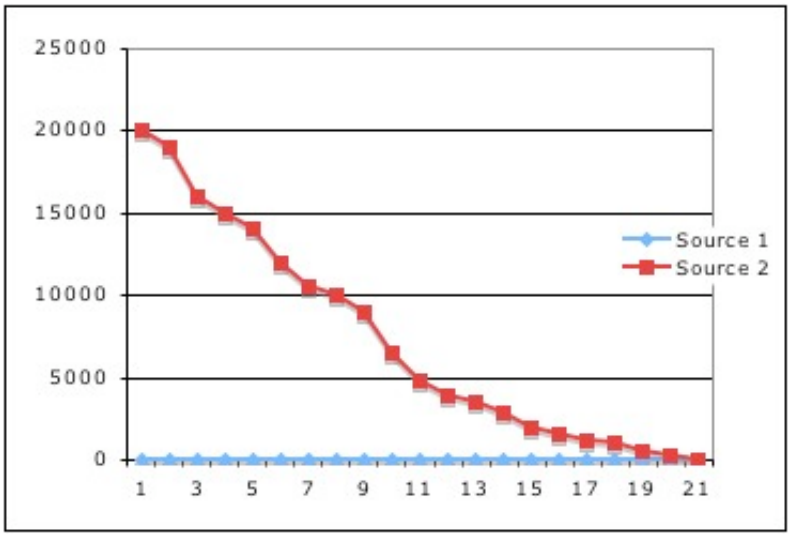


Figure 5.2: Unnormalised scores from non-homogeneous sources

This would have the effect of leaving all scores within the range 0 to 1 and so allowing a more meaningful combination between certain sources. Figure 5.3 shows the effect of this normalisation on the same two sources.

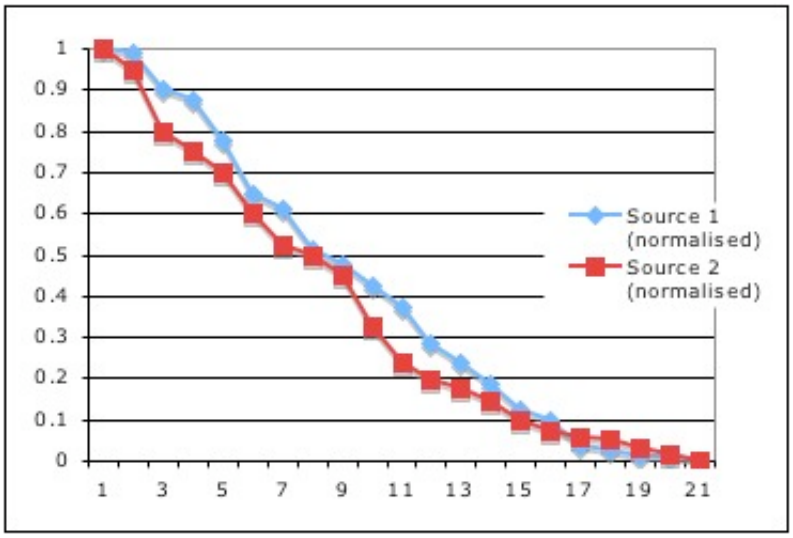


Figure 5.3: Min-Max normalisation on two non-homogeneous sources

Although in general this min-max approach is one of the most popular normalisation methods, there are a number of alternatives, and Montague and Aslam (2001) provides a more comprehensive study of the use of score normalisation for metaseach.

5.1.3 Linear Combination

As an alternative to approaches such as CombSUM, where each of the sources receives equal importance, Bartell et al. (1994) and Vogt and Cottrell (1999) propose the use of weights, in order to promote more "expert" sources. These weights can be generated from an initial training phase, so that an optimal combination may be achieved using this training data. We refer to this as *Linear Combination*, and this is calculated as follows:

Linear Combination:

$$Score(d) = \sum_{i=0}^{n} Score_i(d) \times w_i \tag{5.4}$$

where w_i is the weight associated with the data source i.

5.1.4 Rank Fusion

In addition to the methods previously described, there also exist various methods for combining ranked lists based on the documents' rank positions within each of the lists. These methods assume that the scores of the systems are not directly comparable, and so use the ranks instead.

Among the most straightforward of these rank-based combination approaches is the *reciprocal rank* approach. Using this approach, the score of a document i is calculated from a number of ranked lists ($j = 1 \ldots$ n) as follows:

$$Score(d_i) = \frac{1}{\sum_j rank(d_{ij})} \tag{5.5}$$

A simple extension of this approach is to weight the sources being combined (similar to linear combination using scores), based on an initial training phase that may generate suitable weights. This approach (which we refer to as weighted rank (wrank)) generates a combined score from the ranks of all the sources, taking each source's weight into account:

$$Score(d_i) = \frac{1}{\sum_j (rank(d_{ij})/w_j)} \tag{5.6}$$

where w_j is the weight associated with source j.

One advantage of using ranked-based combination is that score normalisation (as discussed previously) is no longer required (as the sources are combined based on the ranks), making ranked lists more comparable. However one potential disadvantage of a rank-based combination approach is that the scores themselves hold information that may potentially be of use for combining the sources of data more effectively.

5.2 Dempster-Shafer combination

Dempster-Shafer's Theory of Evidence is a formal framework for the combination of independent sources of evidence. The theory was originally proposed by Dempster (1968) and extended by Shafer (1976). Its main appeal is its explicit representation of ignorance in the combination of evidence, which is expressed by *Dempster's combination rule* and in this way extending the classical probability theory. In this section we outline Dempster-Shafer's Theory of Evidence and its application in combining query-independent measures of importance.

The *frame of discernment* Θ represents the set of all possible elements $\{\theta_1, \theta_2 \ ... \ \theta_n\}$ in which we are interested in. The power set of Θ (denoted 2^Θ) contains all possible propositions. Each of these propositions is assigned a *probability mass function* (denoted m) where

$$\sum_{A \subseteq \Theta} m(A) = 1 \tag{5.7}$$

Here A is any element of 2^Θ and m(A) is the amount of the total belief committed exactly to A.

If $A \subseteq \Theta$ and $m(A) > 0$, then A is called a *focal point*. The function m(A) measures the amount of belief that is exactly committed to A, not the total belief

that is committed to A. Each mass m(A) supports any proposition B that is implied by A. Therefore the belief that a proposition A is true is gained by adding all the masses m(B) allocated to propositions B that imply A. This *degree of belief* is defined as follows:

$$Bel(A) = \sum_{B \subseteq A} m(B) \tag{5.8}$$

The *uncommitted belief* m(Θ) is a measure of the probability mass that remains unassigned. This is the measure that is used to model "ignorance" or conversely the "confidence" in the evidence, this is said to be its *uncertainty*, and is defined as follows:

$$m(\Theta) = 1 - \sum_{A \subset \Theta} m(A) \tag{5.9}$$

Two bodies of evidence within the same frame of discernment (provided they are independent) can be combined using *Dempster's Combination Rule*. If m_1, m_2 are the two probability mass functions of the two bodies of evidence defined in the frame of discernment Θ, that we wish to combine, the probability mass function m defines a new body of evidence in the same frame of discernment Θ:

$$m(A) = m_1 \oplus m_2(A) = \frac{\sum_{B \cap C = A} m_1(B) * m_2(C)}{1 - \sum_{B \cap C = \emptyset} m_1(B) * m_2(C)} A, B, C \subseteq \Theta \tag{5.10}$$

This returns a measure of agreement between the two bodies of evidence.

5.2.1 Applying Dempster-Shafer to Query-Independent Similarity Measures

Here we wish to apply the *Dempster-Shafer's Theory of Evidence* to combine various query-independent measures of retrieval, and we outline how it is applied specifically to this area, as well as computational savings that can be made in its calculation.

The measures of importance that we are dealing with are defined for each document in the collection, therefore the frame of discernment Θ is defined as $\{d_1, d_2...d_n\}$, where d represents each of the documents in the collection. We must normalise each of the query-independent measures, so that they fulfill the *probability mass function*, as defined by equation 5.7. For this we must also know the *uncertainty* value m(Θ) associated with each measure so that the measure is normalised to 1 - m(Θ), to satisfy equation 5.9. For example if we wish to combine the two query-independent measures, *access counts* ($m_{ac}(d)$) and *URL length* ($m_{ul}(d)$), we may choose the m(Θ) values for each source, based on their individual performances. If for instance this gives us uncertainty values of $m_{ac}(\Theta) = 0.4$ and $m_{ul}(\Theta) = 0.6$, we then normalise the $m_{ac}(d)$ values to sum to 0.6 and normalise the $m_{ul}(d)$ measures to sum to 0.4

To then combine these measures $m_{ac}(d)$ and $m_{ul}(d)$ we use *Dempster's Combination Rule*, as defined in equation 5.10. It is possible to simplify this equation, as this currently takes account of all elements in the set 2^{Θ}. However, as we have non-zero basic probability assignments for only the singleton subsets of Θ, i.e. each of the documents $\{d_1, d_2...dn\}$, as well as the uncertainty in the body of evidence $m(\Theta)$, we can reduce the complexity in the combination, similar to Jose et al. (1998) and Plachouras and Ounis (2002). The modified equation 5.10 can be re-written as:

$$m_{ac,ul}(\{d\}) = \frac{m_{ac}(\{d_i\}) * m_{ul}(\{d_i\}) + m_{ac}(\{d_i\}) * m_{ul}(\Theta) + m_{ac}(\Theta) * m_{ul}(\{d_i\})}{1 - \sum_{B \cap C = \emptyset} m_1(B) * m_2(C)} \tag{5.11}$$

Since the denominator in equation 5.12 is a normalising factor and is independent of $\{d_i\}$ (Jose, 1998), we follow the lead of Plachouras and Ounis (2002) and Y. Alp Aslandogan (2000), by simplifying this formula to:

$$m_{ac,ul}(d) \propto m_{ac}(d) * m_{ul}(d) + m_{ac}(d) * m_{ul}(\Theta) + m_{ac}(\Theta) * m_{ul}(d) \tag{5.12}$$

This produces a single (combined score) for each document in the collection. This by itself can now be treated as a new query-independent source of evidence.

5.3 Support Vector Machines

Support Vector Machines (SVMs) are a set of machine learning algorithms that have been used in the areas of *classification* and *regression*. SVMs were first suggested by Vapnik in the 1960s for classification, and were formally introduced by Boser et al. (1992), and since then SVMs have have been applied in a wide range of areas including: hand-written character recognition (Cortes and Vapnik, 1995; Scholkopf et al., 1996); face recognition (Osuna et al., 1997); and text categorisation (Joachims, 1998). In this section we describe the basic operations of SVMs and suggest how they may be utilised in order to combine query-independent sources – for a more detailed analysis of SVMs we suggest Vapnik (1995), Vapnik (1998) and Cortes and Vapnik (1995).

Focusing on the use of SVMs for two-class classification, here the SVM learns by example to assign labels, e.g. either positive (+1), or negative (-1) to data. In order to do this the SVM is firstly given a set of l *training* examples in the form $(\mathbf{x}_i, y_i)$, where i = 1, 2, . . ., l, each $\mathbf{x}_i$ is a p-dimensional real vector (list of p numbers), and $y \in 1, -1$. The task for the SVM is to learn the mapping from $\mathbf{x} \rightarrow y$; choosing from the set of all possible hypotheses, the one that minimises the risk of error in the classification of a new (unseen) example, minimising this error will lead to better generalisation.

5.3.1 Separating Hyperplane

The human eye is very good at pattern recognition; we can quickly identify two distinct classes in Figure 5.4. In this case it is easy to draw a separating line between the two classes of training data. When labelling new data, the classification process simply decides which side of the separating line that the new point goes on, and

then it is labelled into that class. In the case of Figure 5.4, where the data is represented in two dimensions, the data may be separated using a line, however in higher dimensional space a *hyperplane* is required to separate the data.

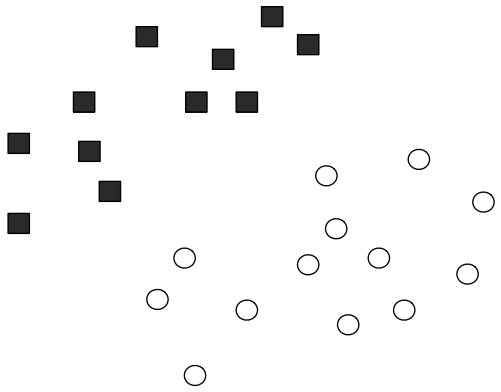


Figure 5.4: Two separate classes of data

5.3.2 Maximum-Margin Hyperplane

Considering again the problem of drawing a separating line between the two sets of data in Figure 5.4, as illustrated in Figure 5.5 there may be an infinite number of different separating lines drawn between the two sets of data.

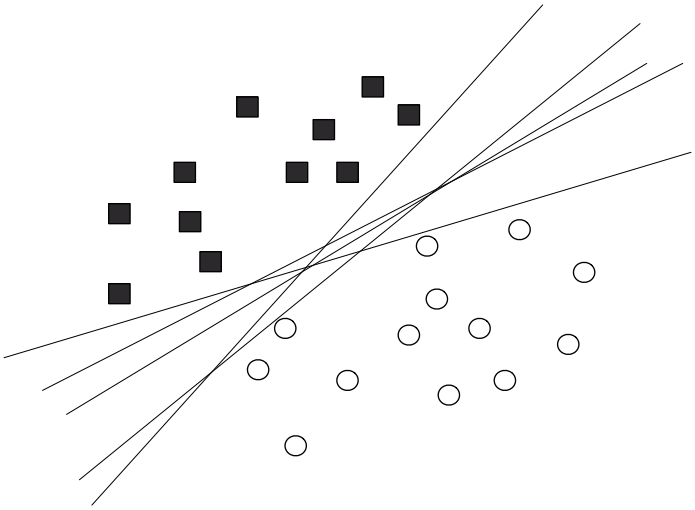


Figure 5.5: Multiple separating lines

However, what the SVM aims to find is the optimal separating *hyperplane* (H)

between classes, by focussing on the training cases that are placed at the edge of the class descriptors; these training cases are referred to as *support vectors*. A simple example is shown in Figure 5.6, where the SVM attempts to maximise the margin between two different sets of data that are linearly separable in a two-dimenstional space. Although there are a infinite number of lines that can be drawn between the two set of data, the SVM attempts to find two parallel lines H1 and H2 (i.e. the dotted lines in Figure 5.6), each of which borders one set of data, such that the distance between the two lines is maximised. The boundary between these two classes are defined by vectors that lie on, or near the two parallel lines, these are known as *support vectors*. The line that lies between the two lines H1 and H2 forms the classification boundary, and is known as the *optimal hyperplane* (H). By selecting this hyperplane it maximises its ability to predict the correct classification of unseen examples.

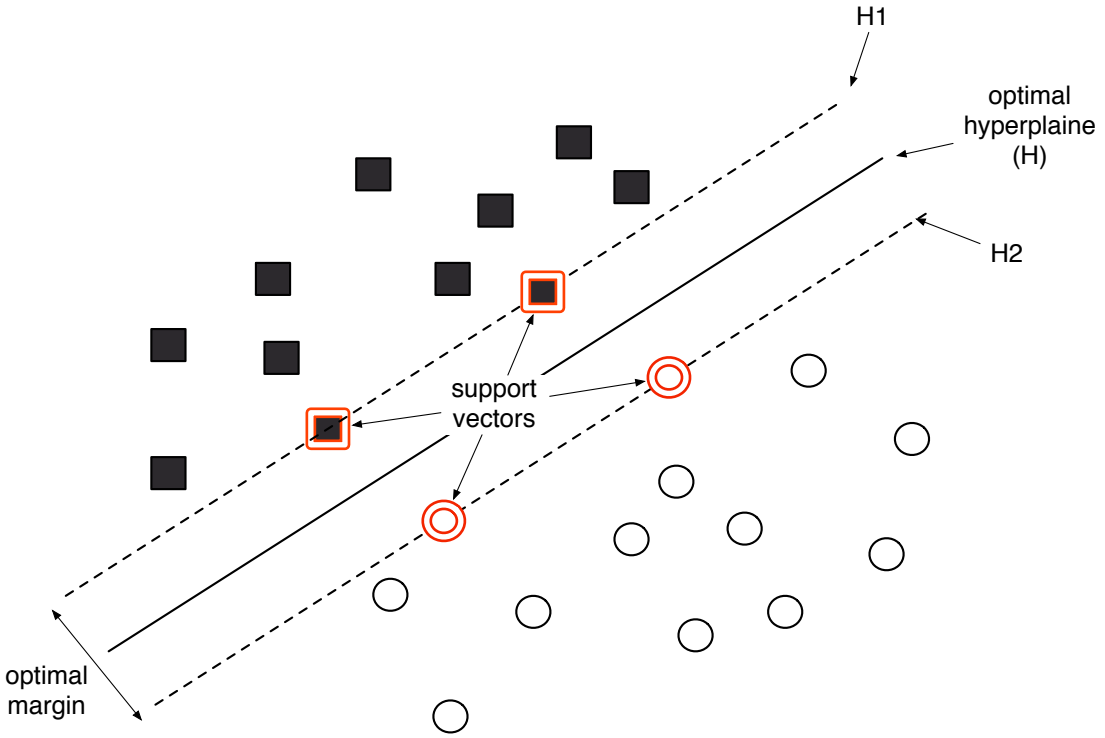


Figure 5.6: SVM separating sets of data

The optimal hyperplane is described by the following:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{5.13}$$

where **w** and b are parameters of the hyperplane. The parallel hyperplanes H1 and H2 that lie on the decision boundary are defined by:

$$\mathbf{w} \cdot \mathbf{x} + b \geq 0 \qquad for\ y_i = 1 \tag{5.14}$$

$$\mathbf{w} \cdot \mathbf{x} + b \leq 0 \qquad for\ y_i = -1 \tag{5.15}$$

These can be combined into the following inequality:

$$y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0 \quad for\ all\ i \tag{5.16}$$

The optimal hyperplane (H) can be found by minimising the following:

$$\frac{1}{2}\|\mathbf{w}\|^2 \tag{5.17}$$

subject to:

$$y_i(\mathbf{w}.\mathbf{x} + b) \geq 1, i = 1, ..., l. \tag{5.18}$$

5.3.3 Soft Margin

Ideally the SVM should separate the groups of feature vectors completely into non-overlapping classes, however, in certain cases this may not be possible, or it may result in a model that does not generalise well to new data – which is referred to as *overfitting*.

Intuitively we would like the SVM to be able to handle certain atypical cases in the training data and to allow them to fall on the "wrong side" of the hyperplane. The SVM allows this by adding a *soft margin*. Although it may be beneficial to allow a certain amount of misclassification to occur, we do not want the SVM to

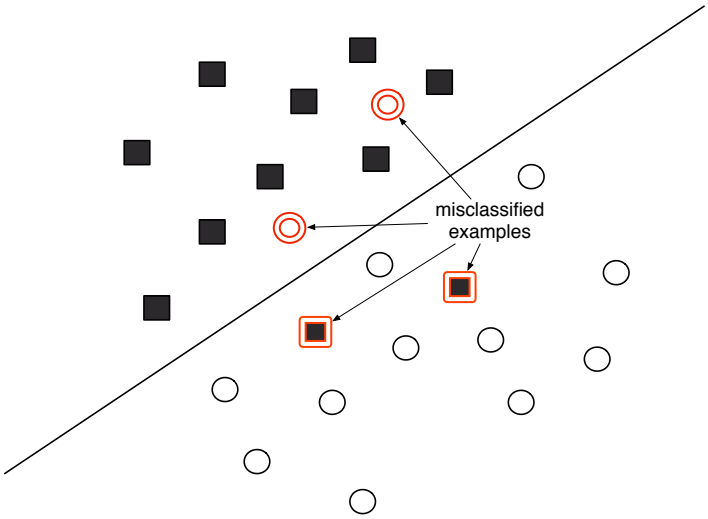


Figure 5.7: Misclassified examples

allow too many misclassifications – in order to handle this the SVM employes the slack variables ξ_i (which measures the amount of misclassification), as well as a cost parameter C (which dictates the level of flexibility that is allowed during separation). Depending on the value chosen for C, a certain level of errors may be acceptable to allow a more generalised model: increasing the C values increases the penalty for misclassified training examples. Figure 5.7 shows an example where four points have been misclassified to allow a more general separation of the two classes.

With the incorporation of this C parameter and the slack variables ξ_i, the SVM's training phase involves the minimisation of the following:

$$\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{5.19}$$

subject to:

$$y_i(\mathbf{w}.\mathbf{x} + b) \geq 1 - \xi_i, i = 1, ..., l. \tag{5.20}$$

where $\xi_i \geq 0$. This can then be solved using the Lagrange multipliers.

5.3.4 Kernel Functions

In general, a *kernel function* maps data from a low-dimensional space to a higher dimensional space, if the correct kernel function is chosen, this mapping may allow separation to occur. There exists a large number of different kernel functions, however, in practice there are only a small number that are widely used for a variety of different types of data, the most popular of these being:

- polynomial
- radial basis function (RBF)
- sigmoid

For our experiments in Chapter 6 we make use of the RBF kernel, which maps training samples non-linearly into a higher dimensional space, so that the SVM can handle cases were the classes are not linearly separable (we also use the linear kernel for comparison purposes). In general the RBF kernel is a popular kernel function as it behaves like the sigmoid kernel for certain parameters (Hsu and Lin, 2002; Keerthi and Lin, 2003), also the polynomial requires more parameters than the RBF kernel (and so requiring more tuning).

5.3.5 Applying SVMs to Query-Independent Evidence

As discussed in Chapter 2 the binary independence model, introduced by Robertson and Sparck Jones (1976), viewed IR as a classification problem. They considered retrieval in terms of classifying documents into two classes: relevant or irrelevant. Their approach essentially classified documents into these two classes at query-time, basing their classification on the query. Adopting this viewpoint of classifying documents as relevant and irrelevant, however, doing so in a query-independent manner, we may now utilise SVMs in order to solve this binary classification problem.

Providing suitable training examples for both relevant and irrelevant documents, we may extract the scores for these documents from suitable query-independent

features for estimating quality documents (e.g. PageRank and info-to-noise ratio, as discussed in Chapter 4). The SVM can then use these training examples to generate a model to can be used to estimate a document's likelihood to belong to the class of relevant or irrelevant documents.

If we can provide suitable examples of both relevant and irrelevant documents, as well as providing suitable query-independent quality estimate scores for each of these documents the SVM's classification model should provide us with an effective way of estimating a document's likelihood to be relevant or not. In order to carry out our experiments using SVMs we use the SVM^{light} software package (Joachims, 1999), which is an implementation of the SVM described in Vapnik (1995).

Figure 5.8 illustrates how we generate a prediction of relevance and non-relevance for each document in the collection, it does this by breaking the process into three key stages:

1. **SVM Training File:** firstly we generate typical examples of relevant and non-relevant documents (further discussed in Chapter 6). With these we build up a training file for the SVM, which consists of the scores for each of the query-independent measures for each of the training documents.

2. **Classification Model:** the *SVM learn* process then "learns" a model to classify relevant and non-relevant documents, based on the example documents and their query-independent scores.

3. **Classification Predictions:** the classification model is then used by the *SVM classify* process, which generates a predictions file, consisting of a likelihood score for each document in the collection belonging to either the *relevant* or *non-relavant* class.

In Chapter 6 we experiment with different ways in which to produce the training example documents, as well as different models for predicting the classifications. Once we have this predictions file (as outputted by the *SVM classify* process) we

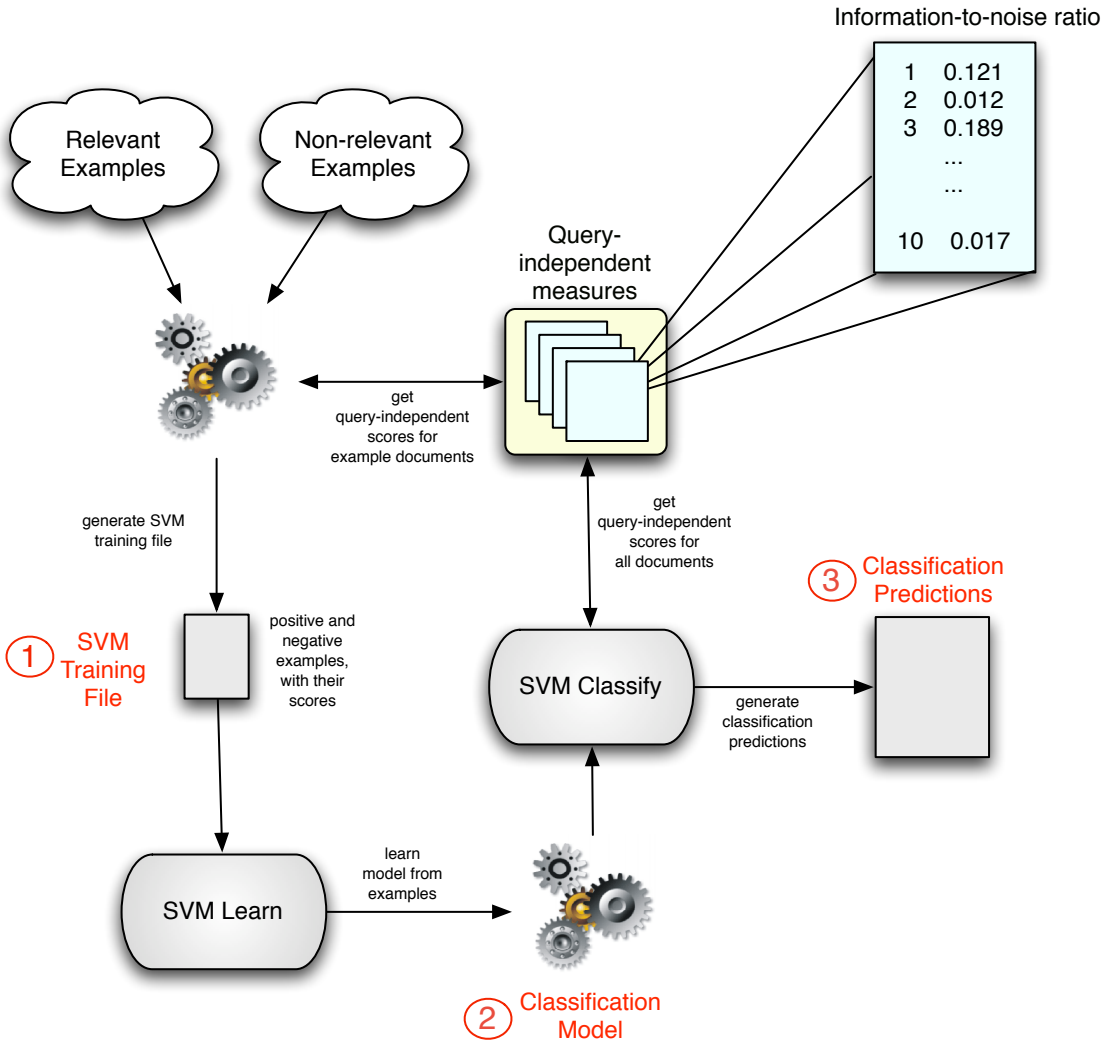


Figure 5.8: Generating SVM predictions from query-independent features

can easily translate this to a query-independent measure: if we classify the relevant class of documents as the positive example class and the non-relevant documents as the negative example class (as outlined previously as the classification labels, $y \in (1, -1)$), then documents with a higher (positive) score are more likely to be relevant than those with a lower (negative) score (according to the SVM's model). This predictions file can then be easily transformed into a query-independent measure that is the combination of all the features (e.g. PageRank, info-noise-ratio) that were used in the *SVM learn* process.

5.4 Combining query-independent sources of evidence

Having discussed various ways in which query-independent sources may be combined, we now discuss (independently of the actual fusion method chosen) how, as well as at what stage these sources are combined.

Similar to the creation of the query-independent measures (as discussed in chapter 4), the combination of these measures is also carried out at different stages, depending on the type of query-independent measure. In this section we describe how this combination process is performed, depending on the type of query-independent measure: static and term-based.

5.4.1 Static Measures

Much the same way as these static measures are created, which is done once, separate from the index creation process (demonstrated in chapter 4), the combination of these static measures is also done once, also in a standalone process separate from the indexing process. Figure 5.9 demonstrates the separate process that creates the initial static measures that are to be used in combination.

Each of these static measures contains a single (static) score for each document in the collection and so their combination also results in a list of static scores associated with each document in the collection, which in itself can also be though of as another static measure. A simple example of combining three static measures (PageRank, URL depth and Info-to-noise ratio) is shown in Figure 5.10, which shows the three static measures being combined to output a new static measure. This new combined list of scores can now be treated as a new static measure, and used to sort an inverted index entry in the same way as any other static measure.

When combining these static scores it is only the type of fusion process used that changes. For example the fusion may use either the CombSUM, or a Dempster-Shafer type combination, which will potentially change the actual static scores as-

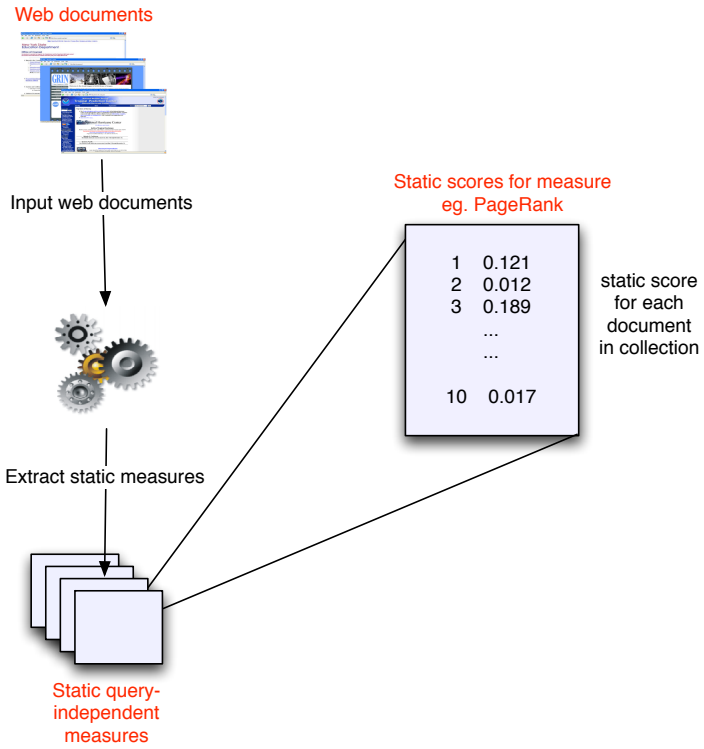


Figure 5.9: Static measure creation

signed to the individual documents, however this is all that will differ.

5.4.2 Term-Specific Measures

Again, similar to the creation of these term-specific measures, which are generated on a term by term basis at the time of indexing, likewise the combination with these measures is also done at this stage. Although it is possible to combine different term-specific measures together, the combination that we wish to investigate is the combination of term-based measures with static query-independent measures.

Figure 5.11 shows how term-specific fusion occurs within the context of the indexing process. Here during the indexing process the term-specfic weightings are calculated on a per term basis, and each term has associated weights (or scores) for each of the documents that that term occurs in. These scores are combined with a chosen static measure and combined in the *term-specific fusion process* to produce a combined list of scores associated with the document that the term occurs in. These

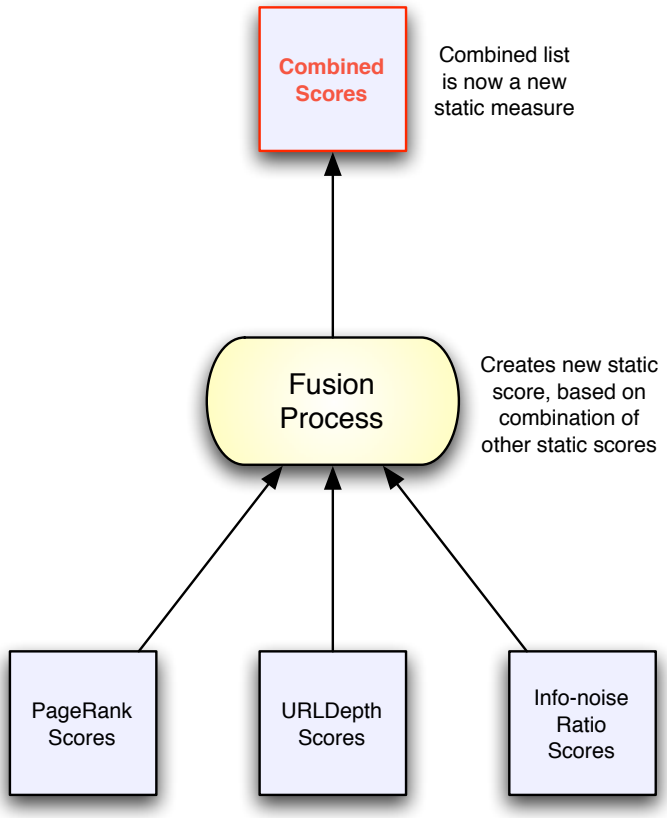


Figure 5.10: Static fusion example

combined scores are then fed back to the indexer, which sorts the posting list for that term using these scores, and the same process is followed for all terms.

5.5 Summary

In this chapter we have discussed various methods that may be used to combine the query-independent measures that we introduced in Chapter 4, the aim of which is to produce a combination of a number of query-independent sources that is in itself a more confident measure than any of the individual measures on their own.

The following chapter presents experiments, carried out in order to investigate the usefulness of different query-independent measures, as well as investigating the gain achieved (if any) from the combination of sources.

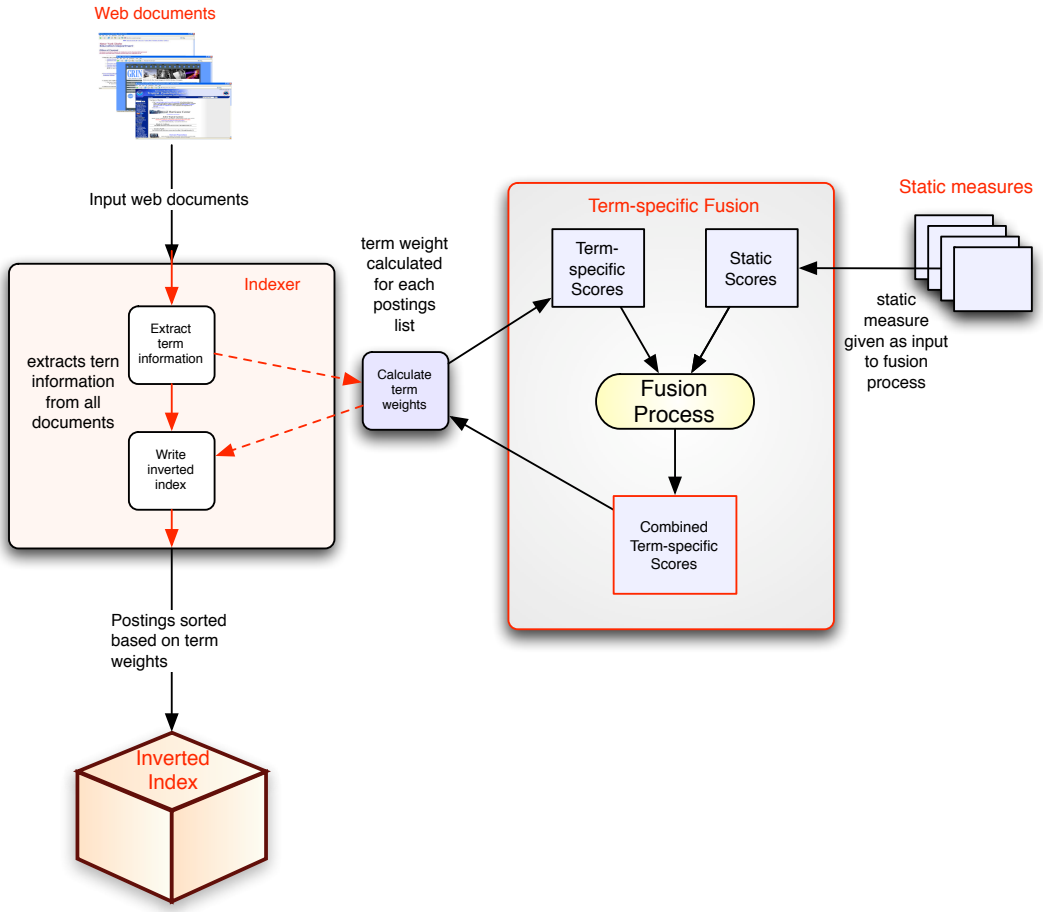


Figure 5.11: Term-specific fusion

Chapter 6

Experiments

"It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong", Richard Feynman (1918-1988)

6.1 Introduction

In the previous chapters we introduced the area of information retrieval, then more specifically discussed the idea of sorting the inverted index using query-independent measures. In order to evaluate the usefulness of this idea, in this chapter we experiment with the use of a sorted index with various types of sortings, and see how these have an affect on the performance of the system.

We then combine these measures together in an effort to increase the overall retrieval performance.

6.2 Overview of Experiments

First of all we describe the experimental setup that we used to carry out our experiments, including the search engine, the document collection and queries used. The main experiments are then presented in the same chronological order in which they were conducted:

- In section 6.4 we experiment with different forms of early termination, and investigate their effect on query execution.
- In section 6.5 we compare the effectiveness of different query-independent measures, both static and term-based.
- Section 6.6 examines how to evaluate the effectiveness of these different sorted indexes.
- Section 6.7 demonstrates the effect of combining different query-independent measures using a number of combination methods described in Chapter 5.
- Section 6.8 examines the trade-offs that are made by eliminating postings with this sorted index approach.
- Section 6.9 investigates the impact that a sorted index has on the effectiveness of the system, as well as the efficiency.

Finally we summarise our results and our main experimental findings in section 6.10.

6.3 Experimental Setup

6.3.1 Físréal Search Engine

Our experiments are carried out using the Físréal search engine (Blott et al., 2004; Ferguson et al., 2005b), which we developed in order to index and retrieve the documents from the SPIRIT collection (a collection of almost 95 million documents from the general Web) (Jones et al., 2002).

All experiments were carried out using a single search engine architecture, in order to alleviate any additional factors that might come into play while using a distributed architecture – this was done in order to isolate the effects of using a sorted inverted index.

The indexing and retrieval processing was carried out a Pentium 4, 2.6GHz PC with 1.5GB of RAM.

6.3.2 Test Collection and Queries

The experiments within this thesis are carried out on the GOV2 test collection, as described in section 2.3, using the queries from the TREC terabyte track from 2004, 2005 and 2006. We use this test collection as, at over 25 million documents, it is the largest document collection available within the TREC framework that provides relevance judgements over a number of ad hoc type queries. At this size, the collection presents a realistically sized collection, and provides a reasonable challenge to a single search engine machine when processing user queries, so that processing less postings per query may be of benefit. For smaller TREC collections such as Wt10g and Wt2g this may not be so beneficial, as the number of documents to be processed for each term would be relatively small anyway. Also as the size of these collections increases, particularly with crawls from the Web, so does the lack of quality of those documents. So although this collection may not contain the large amounts of low quality and spam documents that are to be found throughout the Web, the collection does naturally contain some lower quality documents, and if identified as low quality by our proposed sorting measures, these documents may be suppressed towards the bottom of their respective postings lists, allowing them to be filtered.

6.3.3 Baseline Sorting

In order to compare various sorting metrics, we provide a *baseline* measure to sort the postings within each postings list in the inverted index: this provides a yardstick to compare other sorting measures against. For this baseline measure we use the BM25 formula to sort the postings (as described in section 4.3). We have chosen this as it represents the current state-of-the-art in information retrieval in providing

a probability of relevance between documents and query terms. Therefore it should provide a high performing baseline measure to compare results against. The actual formula used is:

$$bm25(q,d) = \sum_{t \epsilon q} log\left(\frac{N - df_i + 0.5}{df_i + 0.5}\right) \times \frac{(k_1+1)tf_i}{k_1((1-b)+b\frac{dl}{avdl}) + tf_i} \tag{6.1}$$

where df_i is the number of documents in the collection that contain the term i and k_1 , k_3 and b are tuned to 1.2, 1000 and 0.25 respectively. This formula is used to sort both the postings in the index, as well as the documents at retrieval time.

In addition to the use of BM25 as a means to compare sorting metrics to, we also provide a random sorting to give a clearer indication of each measure's performance, not only to the upper bounded measure, but also show how it performs relative to a random sorting. This random sorting is generated by assigning each of the documents in the collection, with a unique and randomly generated number. By taking these to be the scores associated with each of the documents, the postings can then be sorted using this random measure, in an identical way to the generation of the other query-independent sorted indexes (as discussed in section 4.3).

6.4 Early Termination Experiments

Firstly we investigate approaches that allow the early termination of the search process. We carry out this as our premier experiment, as it aims to provide a method that can be used in order to reduce the number of postings that need to be processed for a given query, while at the same time providing the most effective tradeoff with retrieval efficacy. This will equip us with a means by which to allow postings to be eliminated in a uniform manner using various different types of sorted indices in later experiments, and allow those to be evaluated in a consistent manner that will promote the differences among these contrasting sorting metrics and so allow us to evaluate these more effectively.

To illustrate how these different approaches work we use the example postings lists as shown in Figure 6.1, which shows three different postings lists consisting of document identifiers and their corresponding within document term frequency.

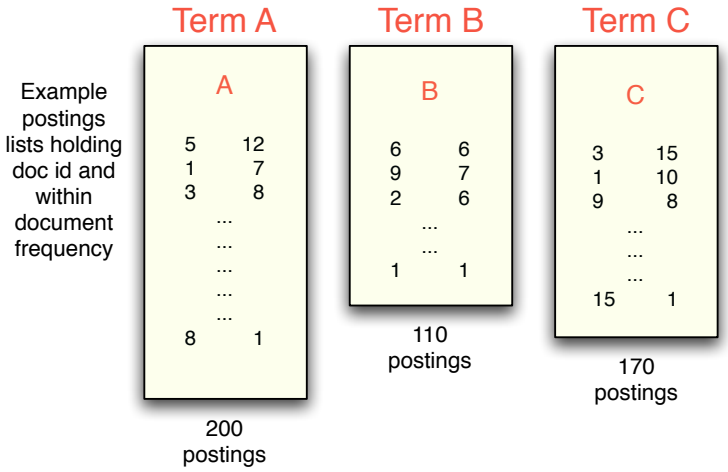


Figure 6.1: Sample postings lists

6.4.1 Maximum postings size cut-off

This approach processes a *maximum number of postings* from each query term's postings list. It has the benefit that a larger percentage of rarely occurring terms are processed, which have a major impact on ranking, due to their high inverse document frequency (idf) score. At the same time only a small percentage of all the postings for a commonly occurring term may be processed, and this may also be beneficial, as frequently occurring terms contribute less to the overall document ranking, due to their lower idf score. We originally referred to this as the *top subset* size approach in (Ferguson et al., 2005a; Blott et al., 2004; Ferguson et al., 2005c) and we have found that this is also the same approach that is utilised by Garcia et al. (2004), although referred to as *maxpost*. This process is illustrated in Figure 6.2, which shows that a specified maximum number of postings is selected, regardless of the size of the list.

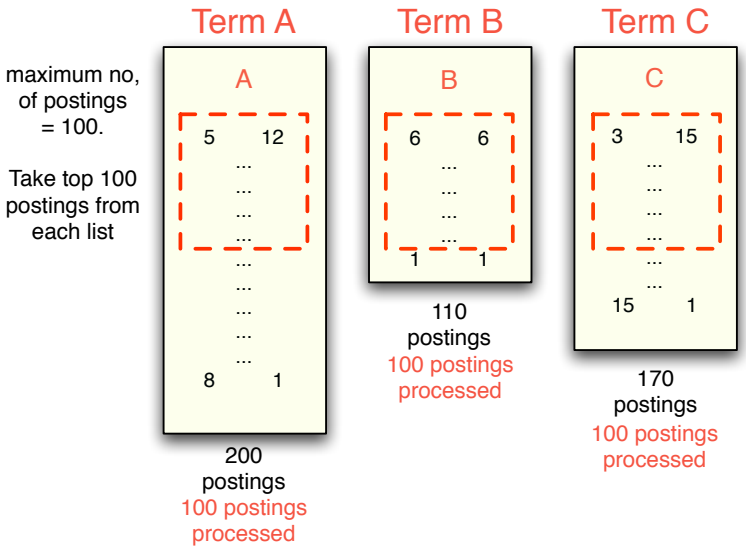


Figure 6.2: Eliminating postings using a maximum posting size

6.4.2 Percentage Size Cut-off

Where the previous approach specifics a maximum size, this percentage-based technique selects the number of postings to process from each postings list by taking a percentage of the total number of documents that that term occurs in. For instance if a term a occurs in 10,000 different documents, with a 10% cut-off, this term would have the first 1,000 postings from its postings list processed. Therefore the actual number of postings that are processed is dependent on the total number of postings for that term. This process is demonstrated in Figure 6.3, here we can see that unlike the fixed size based approach, the number of postings processed is influenced by the size of each list.

6.4.3 Score Based Cut-off

An alternative method of choosing a cut-off point for posting selection it to choose the cut-off point based on the score that it has been sorted by. For instance if the postings have been sorted using BM25, using a cut-off of 0.2, then any posting that does not contribute a score of at least 0.2 is not considered for further evaluation. Using this approach, when the first posting that falls below this threshold is found

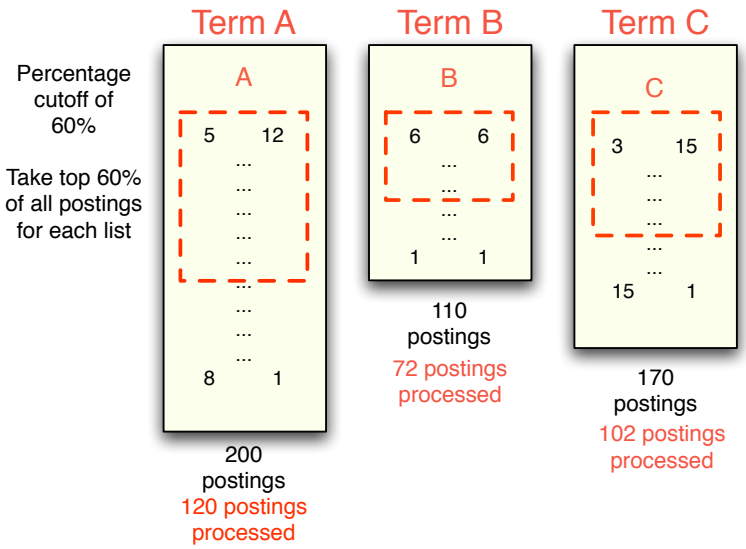


Figure 6.3: Eliminating postings using a percentage based approach

then the rest of the postings within that posting list can also be eliminated, as the postings should contribute the same or less (as the postings have been sorted in descending order). Although it must be noted that when using this approach we may need to calculate different threshold scores for indexes that are sorted based in different measures.

If we use the BM25 formula, as shown in equation 6.1, which incorporates the inverse document frequency element, therefore similar to the fixed size based approach the shorter lists should be given higher weighting, as they produce higher BM25 scores, and because of this they will have a greater percentage of their postings processed. Figure 6.4 gives a graphical representation of this score-based cut-off scheme.

6.4.4 Cut-off Comparisons

In order to compare these different query-time cut-off strategies we measure the overall number of postings that they process and then compare this with the retrieval effectiveness in terms of mean average precision (MAP) and precision at 10 documents (P10), so that we can see the trade-off that is being made between the

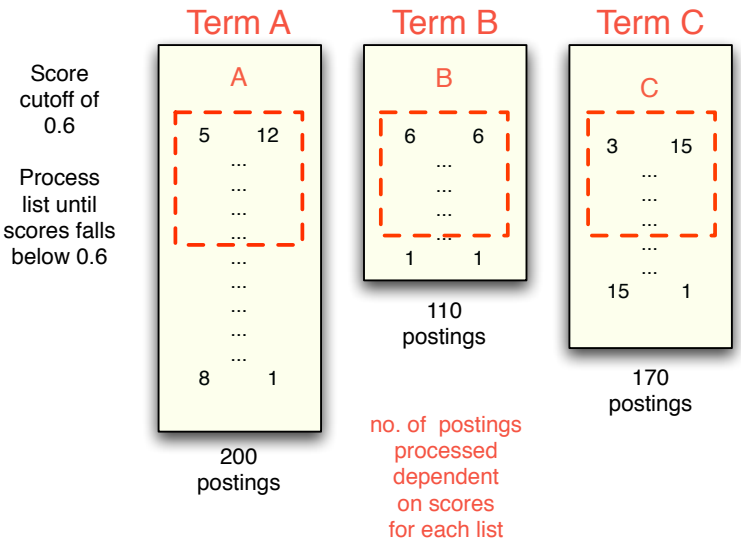


Figure 6.4: Eliminating postings based on their score

number of postings that are processed and query effectiveness. For this we ran 100 ad hoc queries from the TREC 2004 and 2005 terabyte track (topics 701-800) using the different cut-off strategies. Figures 6.5 and 6.6 compare the the performance of these methods based on their MAP and P10 scores respectively.

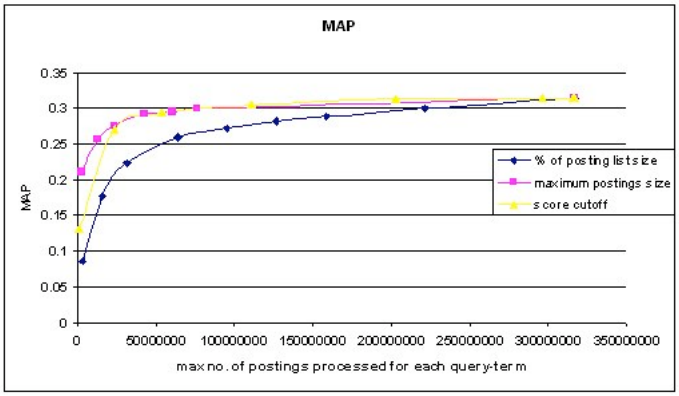


Figure 6.5: Comparisons of different cut-off methods (MAP)

These results show that the *maximum postings size* approach works best for both P10 and MAP (particularly for MAP). The score-based cut-off approach performs ahead of the percentage-based approach for MAP, however the reverse is the case for P10. For this reason we proceed with this maximum postings size approach to

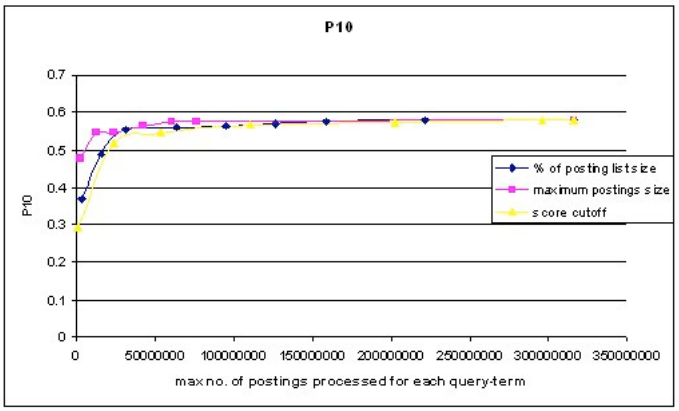


Figure 6.6: Comparisons of different cut-off methods (P10)

compare different query-independent methods of sorting the inverted index (in the following section).

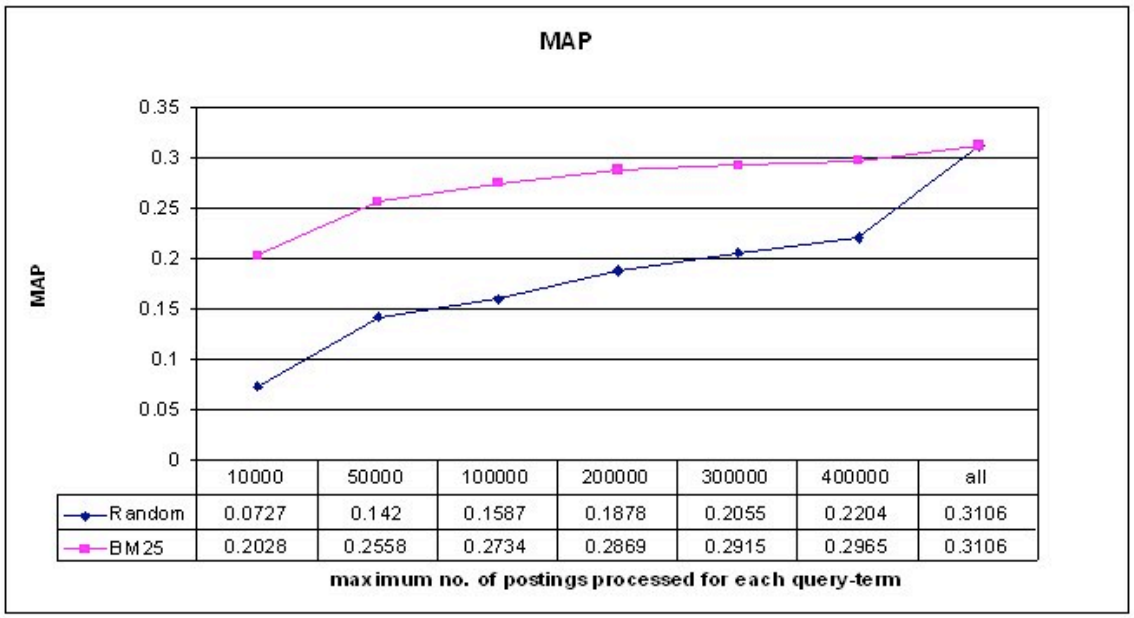
6.5 Query-Independent Sorting

In this section we shall present different query-independent sorting measures that have been introduced in Chapter 4. Here we show how these measures perform individually, as a means of sorting these postings lists.

Firstly we present the performance of both the BM25 sorted index, as well as an inverted index whose postings are sorted in a random order. These will allow for more meaningful comparisons for the other sorted indexes that we will evaluate. For these as well as the other sorted indexes we present the performance of each of the indexes in terms of MAP and P10 as we increase the *maximum postings size*. As we have chosen to use the *maximum postings size* cut-off approach (described in section 6.4), we present the performance figures as we increase the maximum number of postings processed for each query-term, from 10,000 up to the point where all postings are processed for all query-terms. The exact maximum number of postings at the different intervals are as follows: 10,000, 50,000, 100,000, 200,000, 300,000, 400,000 and then finally with all postings being processed. We feel this effectively allows a sorting measure to show its performance as the number of postings are

increasing, and so allowing for comparison of different sorted indexes based on their performance versus the number of postings that the require to process.

We evaluate the efficacy of BM25 as well as the other query-independent measures in this section using the topics 701-850 from the TREC terabyte ad hoc search task. Firstly, Figures 6.7 and 6.8 show the performance of the baseline BM25 sorting, and as with the other measures we will also include the performance of a random sorting, in order to show the relative performance of these and other methods of sorting.



	10000	50000	100000	200000	300000	400000	all
Random	0.0727	0.142	0.1587	0.1878	0.2055	0.2204	0.3106
BM25	0.2028	0.2558	0.2734	0.2869	0.2915	0.2965	0.3106

Figure 6.7: MAP performance of BM25 sorting

From Figures 6.7 and 6.8 we can see that the BM25 sorting offers much better performance at the different cut-off points when compared to a random means of sorting (as would be expected). Secondly we can see that high P10 scores can be achieved having only processed relatively few of all postings. Figure 6.9 shows the percentage of all postings that are being processed at each cut-off point. Looking at the cut-off point of 200,000 postings, Figure 6.9 shows that less than 14% of all postings are evaluated, comparing this with the MAP performance, as shown in Figure 6.7, where at a cut-off point of 200,000, this results in only a 0.0238 drop off in MAP performance. Comparing this with the P10 performance at the same

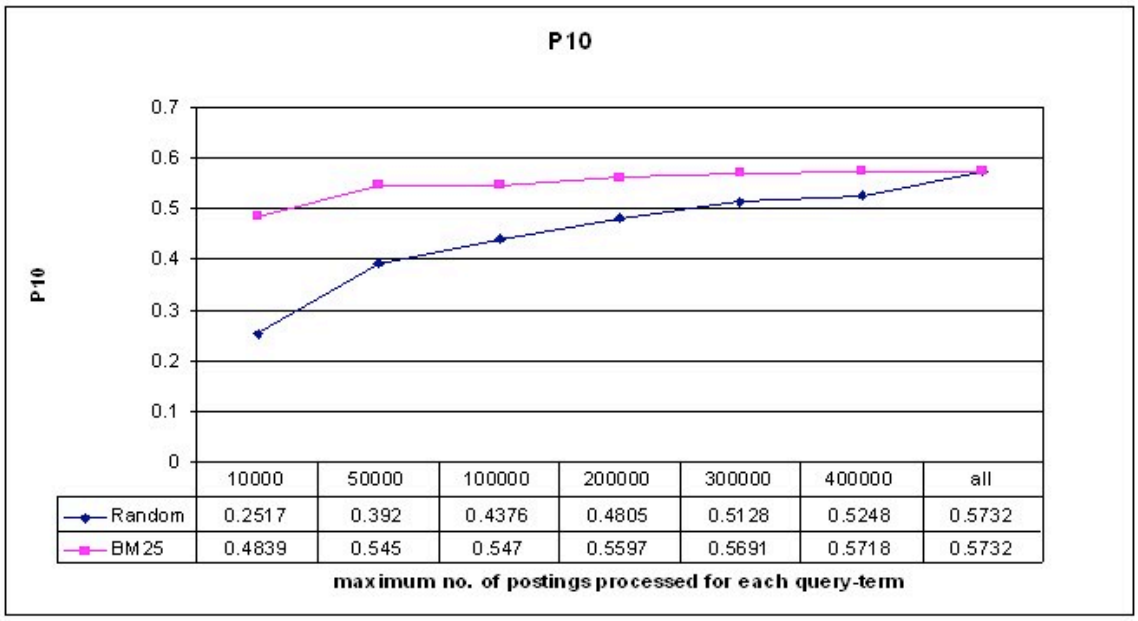


Figure 6.8: P10 performance of BM25 sorting

cut-off point, this results in an even smaller degradation in performance of 0.0135 – we would consider this to be a very small drop in performance, when considering that less than 14% of the available postings for the query terms were evaluated.

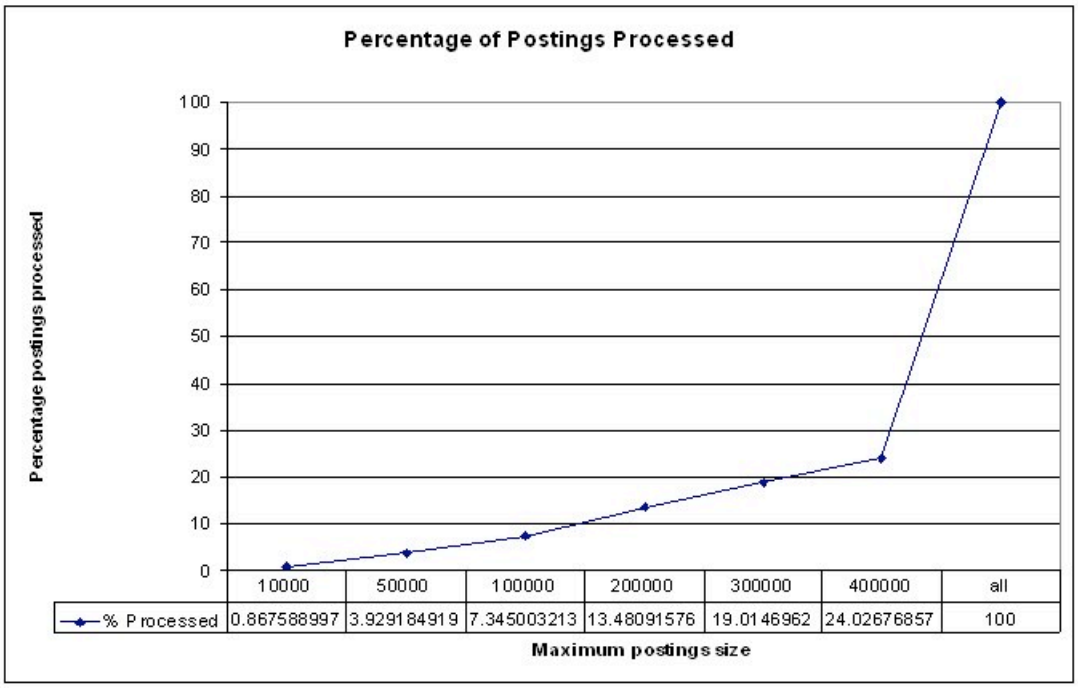


Figure 6.9: Percentage of postings processed at each cut-off point

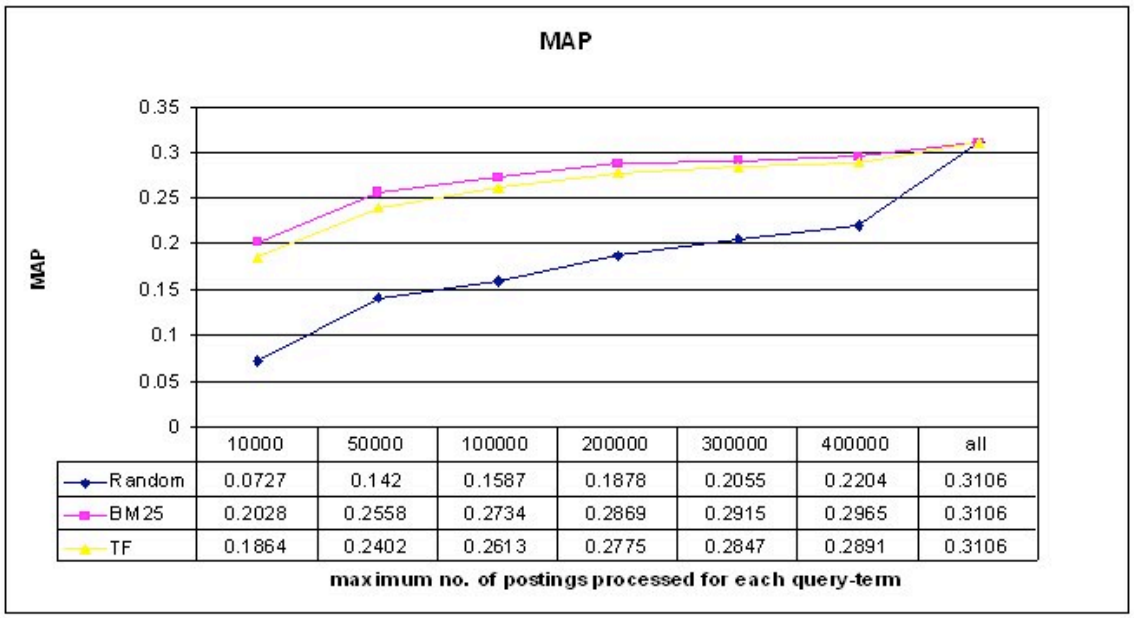
As there is a significant saving in terms of the number of postings being evaluated, we must look at the trade-off being made in terms of the retrieval accuracy and decide if this trade-off is worthwhile. For web search in particular, users are most often only concerned with looking at the top 5-10 documents returned and want results returned quickly, we would therefore suggest that for a general web user they would prefer to receive prompter results from a search engine, and would be willing to accept a small degradation of 0.0135 in P10 for example. We do realise this this of course may not be suitable for all types of applications, such as a search where very high recall is of critical importance. This could obviously become an issue with this approach as there may be a large number of documents may not be considered during retrieval.

In the following sections we sort the inverted index based on various different query-independent measures and investigate their usefulness in allowing effective promotion of relevant documents and an effort to further decrease the trade-off that needs to be made between the number of postings that are processed and the accuracy of the results.

6.5.1 Term-Specfic Sorting

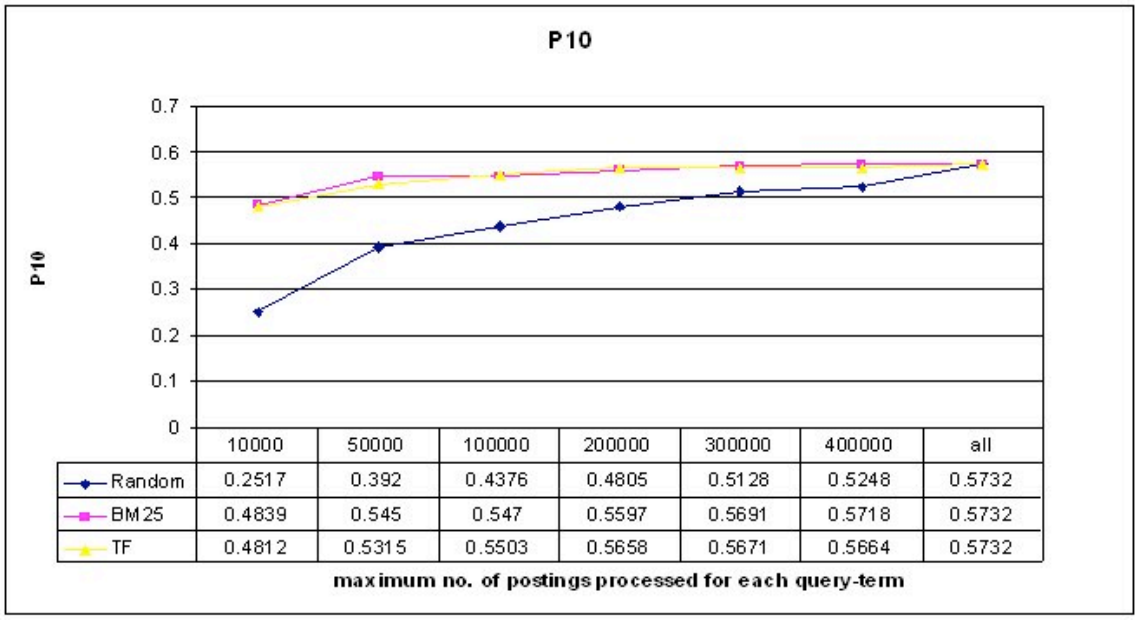
As discussed in section 4.3 these term-specific methods of sorting postings lists provide scores for documents on a term-by-term basis, rather than providing a single static score for each document. The BM25 sorting, as previously discussed, is an example of a term-specific means of sorting these posting lists. In order to compare the performance of this against other term-specific method we choose firstly to look at the performance of an index sorted based on the within document term frequency (TF), this is the number of times that the term occurs within a document. This was the type of index proposed by Persin (1994) and Persin et al. (1996). The performance of this measure is shown in Figures 6.10 and 6.11.

From Figure 6.10 we can see that BM25 outperforms the TF means of sorting, this should be expected, as the BM25 formula considers more information in calcu-



	10000	50000	100000	200000	300000	400000	all
Random	0.0727	0.142	0.1587	0.1878	0.2055	0.2204	0.3106
BM25	0.2028	0.2558	0.2734	0.2869	0.2915	0.2965	0.3106
TF	0.1864	0.2402	0.2613	0.2775	0.2847	0.2891	0.3106

Figure 6.10: MAP performance of TF sorting



	10000	50000	100000	200000	300000	400000	all
Random	0.2517	0.392	0.4376	0.4805	0.5128	0.5248	0.5732
BM25	0.4839	0.545	0.547	0.5597	0.5691	0.5718	0.5732
TF	0.4812	0.5315	0.5503	0.5658	0.5671	0.5664	0.5732

Figure 6.11: P10 performance of TF sorting

lating it's measure. However with P10 (as shown in Figure 6.11) the difference is less evident, and overall their performance is quite similar. Overall, for such a simple measure TF provides a relatively effective means of sorting an inverted index.

Although with this type of sorting we would expect that long documents would have an unfair advantage in being promoted towards the top of the postings lists as

longer documents obviously have a greater probability of having terms with higher (within document) term frequencies. However a simple normalisation of the TF (NTF), such as dividing by the document length (shown in equation 6.2) degrades the performance, as can be seen in Figures 6.12 and 6.13, as this perhaps is liable to penalise long documents too severely and is obviously not as effective as the normalisation employed by the BM25 algorithm, which saturates the influence of the term frequency element, as well as controlling the influence of the document length by the use of its b parameter.

$$NTF = TF/docLength \tag{6.2}$$

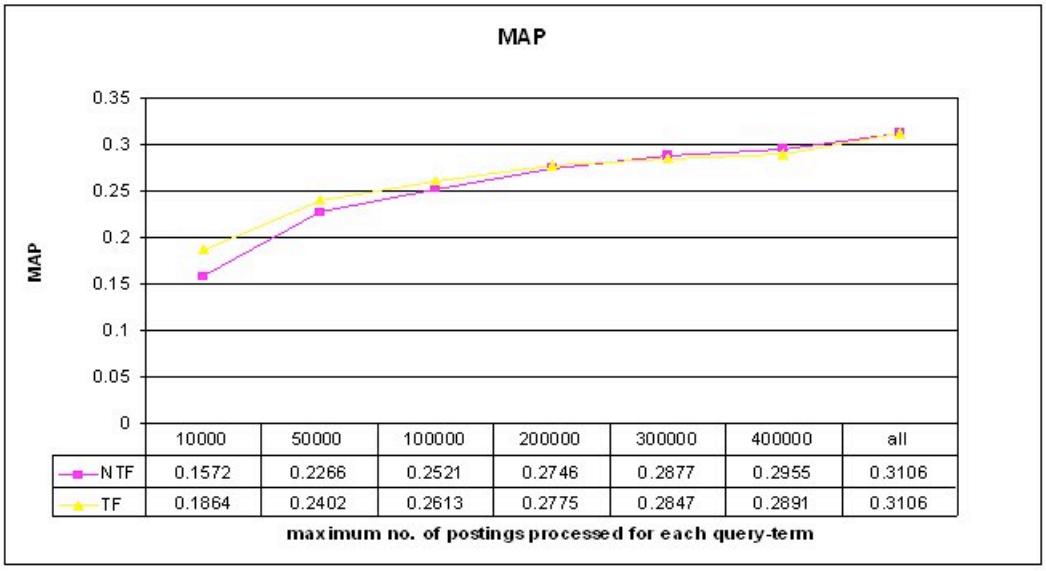


Figure 6.12: MAP performance of NTF sorting

6.5.2 Global BM25 Scores

Although the previous BM25 sorting is carried out on a term by term basis, we can however provide a static BM25 scoring of all the documents in the collection. These scores are calculated in a similar way to which the term-specific scores are generated, however we also keep track of the scores accumulated by each document and increment their score after each term is processed. For this we use the BM25

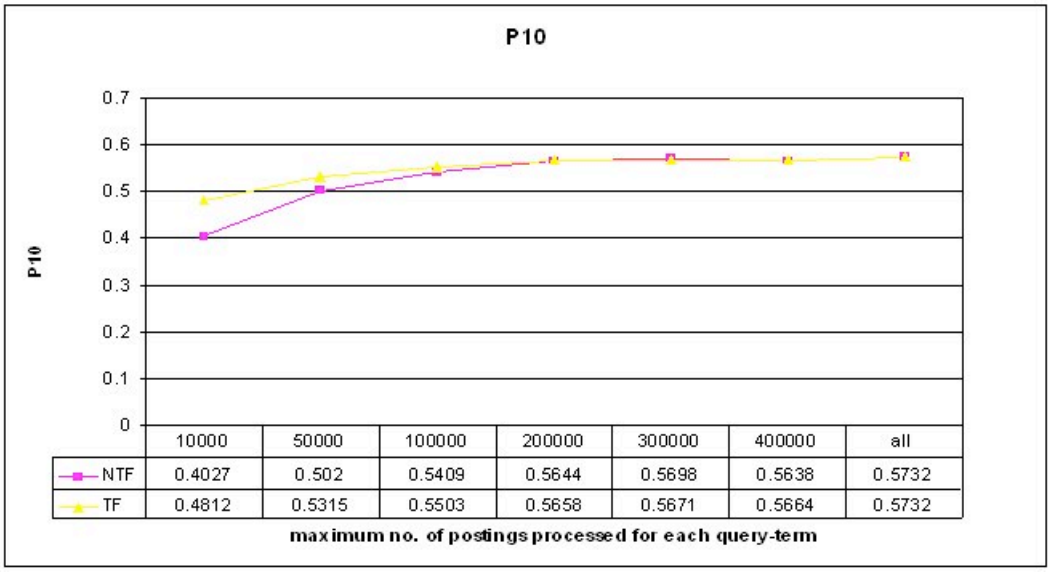


Figure 6.13: P10 performance of NTF sorting

formula, as in equation 6.1, which takes into account the relative importance of each of the terms, unlike the scores calculated on a single term basis (where it is unnecessary), as it is only being used to order documents specific to that term, and so its global importance is not of any consequence.

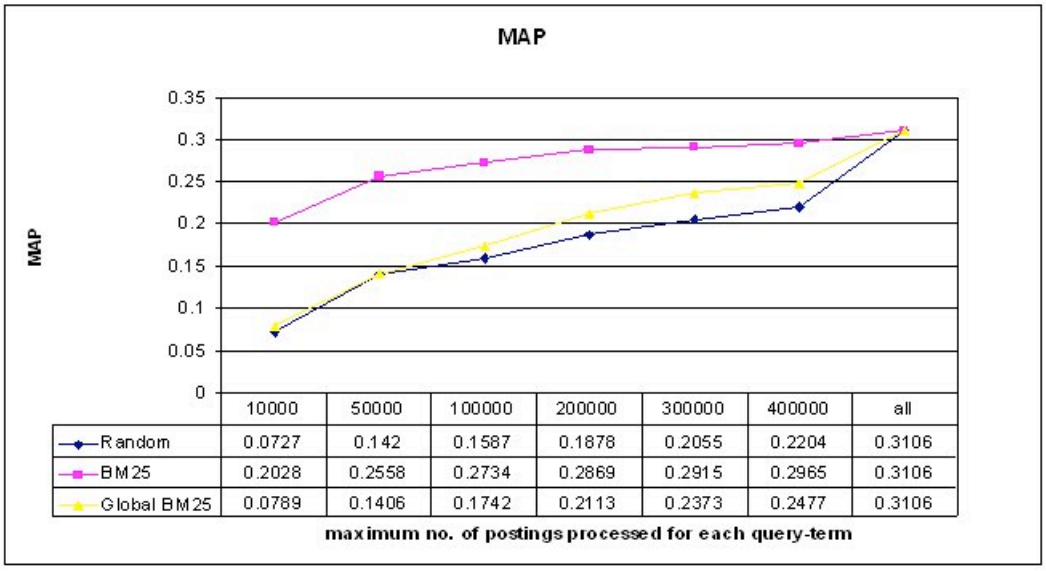


Figure 6.14: MAP performance of global BM25 sorting

As can be seen from Figures 6.14 and 6.15, this global BM25 sorting does not perform as well as the term-specific BM25 sorting, this is not surprising as the

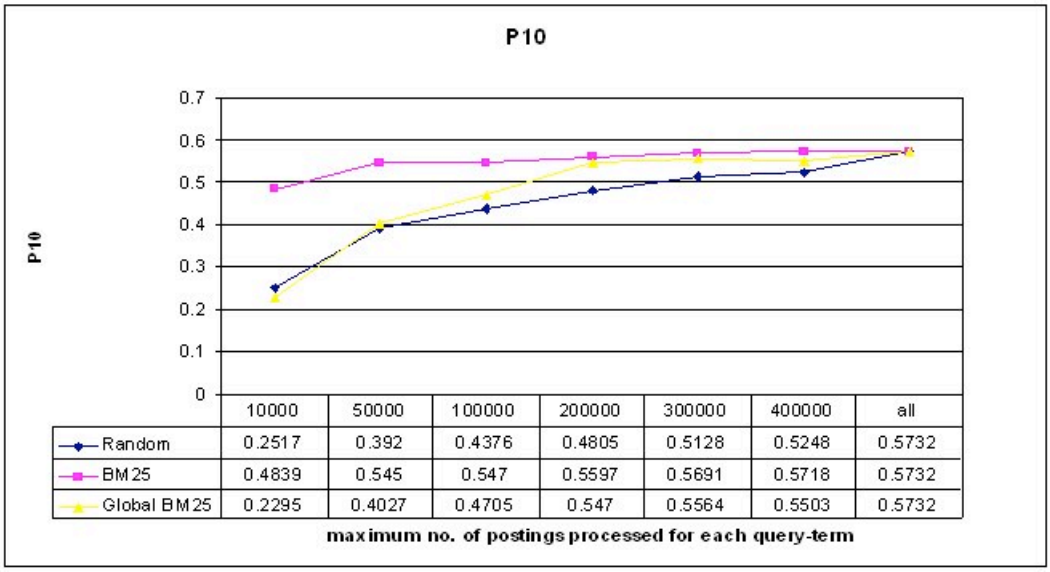


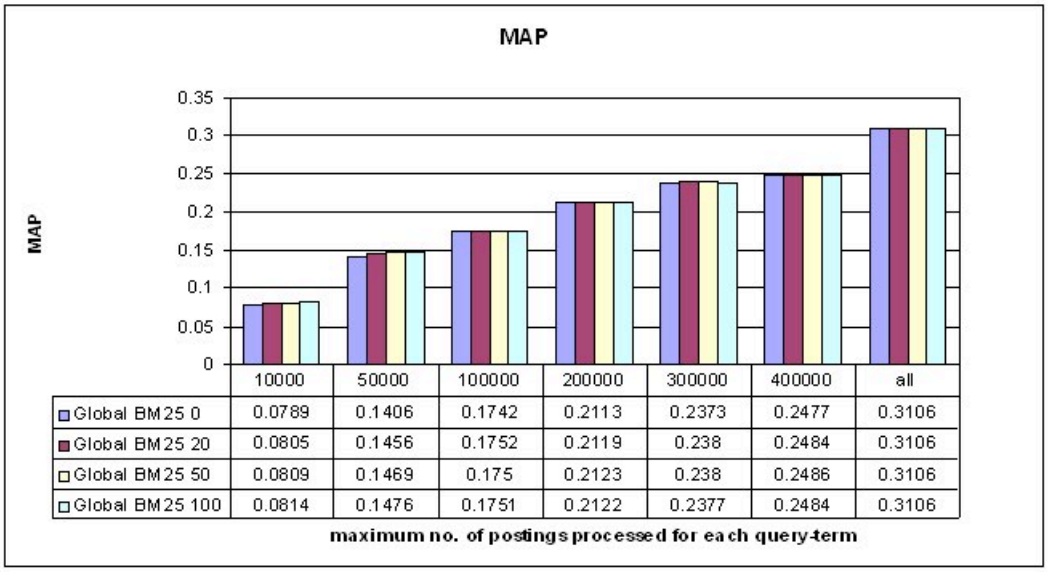
Figure 6.15: P10 performance of global BM25 sorting

ordering of each of the postings lists is done based on static scores that are the same for all terms and we would suggest that a single static feature would find it difficult to improve upon a similar term-specific ordering. Nevertheless these global BM25 scores provide us with useful baseline for static scores that we will make use of for further experiments.

One of the reasons for the underachievement of this global BM25 measure we believe may be due to its upweighting of infrequently occurring terms in the collection – due to the idf component of the BM25 formula. The idf component gives a higher weight to terms that occur less frequently within a collection, this is certainly beneficial when we do not want to give high weights to very frequently occurring documents, however this does give abnormally high weights to very infrequently occurring terms within a collection (which are more likely correspond to misspellings, rather than a meaningful term). As we discussed previously (in Chapter 3) and shown in Figure 3.9 the most important terms are those that occur a moderate number of times, and as shown in Figure 3.9, there are two thresholds, where the collection frequency of a term becomes: too frequent and too infrequent to be useful. Usually the case where the terms occur too frequently are taken into account with

the removal of a certain list of "stopwords" (as we have done with our search system). However, identifying a pre-defined list of infrequently occurring terms that are to be removed is somewhat more difficult to generate (in a collection-independent manner at least). In order to investigate the effect of this on the generation of our global BM25 measure, we no longer considered terms that did not occur more than a specified number of times within the collection. Varying this threshold would then give give us a clearer indication of the effect of including of infrequent terms on the generation of a static BM25 measure.

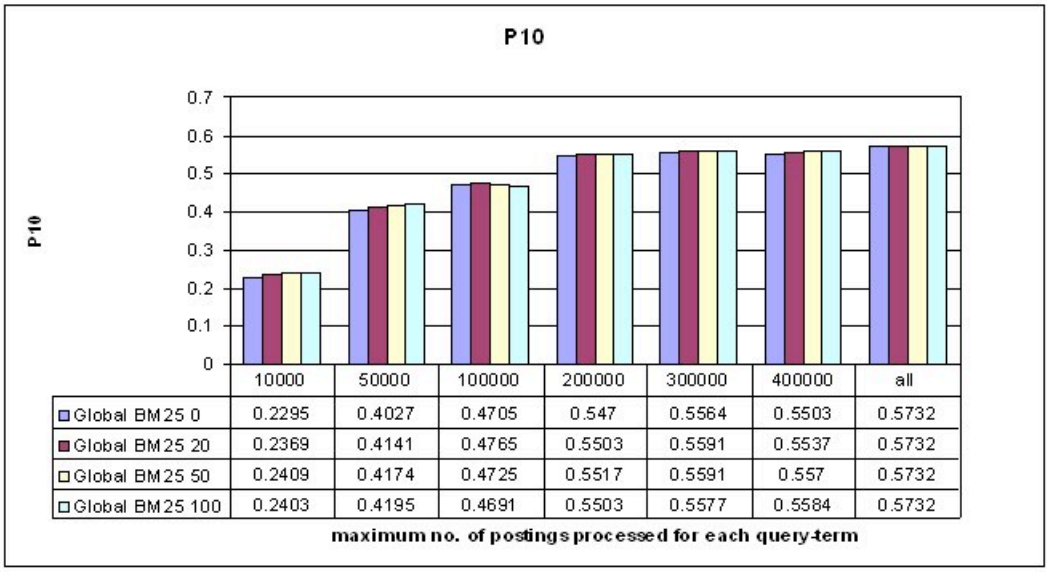
We firstly choose to increase this threshold (t) from frequency 0 (original global BM25 measure), then 20, 50 and finally 100 (here a term must occur at least t times in order to be included). Figures 6.16 and 6.17 show the performance of these new global BM25 measures, in terms of MAP and P10 respectively. Examining these results at a reasonable cut-off point of 200,000 documents per postings list, we can see that there is a minor improvement between the original measure and the result where only terms with a global collection frequency of over 50 are considered: with an increase in MAP of 0.5% and an increase of 0.8% in P10.



	10000	50000	100000	200000	300000	400000	all
Global BM25 0	0.0789	0.1406	0.1742	0.2113	0.2373	0.2477	0.3106
Global BM25 20	0.0805	0.1456	0.1752	0.2119	0.238	0.2484	0.3106
Global BM25 50	0.0809	0.1469	0.175	0.2123	0.238	0.2486	0.3106
Global BM25 100	0.0814	0.1476	0.1751	0.2122	0.2377	0.2484	0.3106

Figure 6.16: MAP performance of global BM25 sorting with term frequency threshold

Leading on from this approach where we eliminate the contribution of the rarest



	10000	50000	100000	200000	300000	400000	all
Global BM25 0	0.2295	0.4027	0.4705	0.547	0.5564	0.5503	0.5732
Global BM25 20	0.2369	0.4141	0.4765	0.5503	0.5591	0.5537	0.5732
Global BM25 50	0.2409	0.4174	0.4725	0.5517	0.5591	0.557	0.5732
Global BM25 100	0.2403	0.4195	0.4691	0.5503	0.5577	0.5584	0.5732

Figure 6.17: P10 performance of global BM25 sorting with term frequency threshold

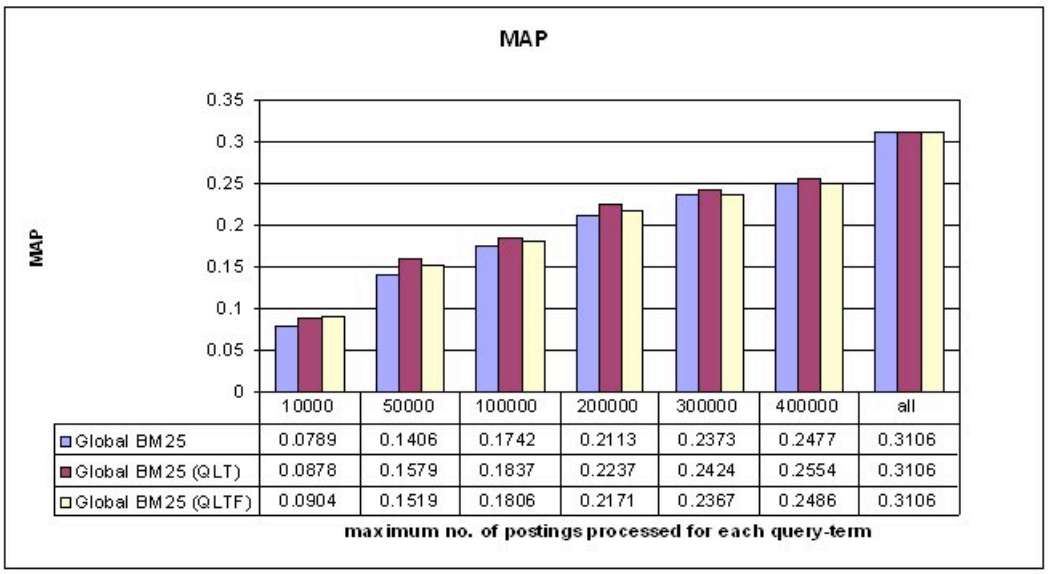
terms in the collection from contributing to the generation of the global BM25 measure, we considered at what point would it be considered reasonable to impose this threshold. so that unimportant terms are not considered and yet important rare terms are still considered. If we consider the terms that the users are searching for as the most important terms in the collection, we may then use these terms to impose a similar threshold as before. In order to do this we used a large query log (of 2.4 million queries) from the *Excite* search engine from December 1999. We used this query log by gathering all the terms contained within the query log, then generated two alternative global BM25 measures based on the term statistics of these queries.

- **global BM25 Query Log Terms (QLT):** with this approach we generate the global BM25 scores by only including terms that have been issued in the query log. This is a similar approach to the previous approach of using a threshold – except that rather than the threshold being imposed based on the global term frequency, it is imposed base on the occurrence of the term in the query log.

- **global BM25 Query Log Term Frequency (QLTF):** as an alternative

to this approach we decided to also take into account the frequency that the term occurs in the query log. Having removed the most commonly occurring terms in the query logs (i.e. stopwords) we then consider the frequency that each term occurs within the query log when calculating the term importance (i.e. idf component of the BM25 calculation).

The performance of these two measures are shown in Figures 6.18 and 6.19. Again if we examine these results at a cut-off point of 200,000 documents per postings list (as above), we can see that there is a quite significant improvement between the original measure and the QLT approach: with an increase in MAP of 5.9% and an increase of 2% in P10.



	10000	50000	100000	200000	300000	400000	all
Global BM25	0.0789	0.1406	0.1742	0.2113	0.2373	0.2477	0.3106
Global BM25 (QLT)	0.0878	0.1579	0.1837	0.2237	0.2424	0.2554	0.3106
Global BM25 (QLTF)	0.0904	0.1519	0.1806	0.2171	0.2367	0.2486	0.3106

Figure 6.18: MAP performance of global BM25 sortings using query log threshold

6.5.3 Linkage Analysis

As discussed in section 4.2, linkage analysis provides certain query-independent measures of document popularity, based on the hyperlink linkage structure between documents. For these experiments we have chosen to use both the indegree and PageRank measures as a means of evaluating the effectiveness of these linkage analysis

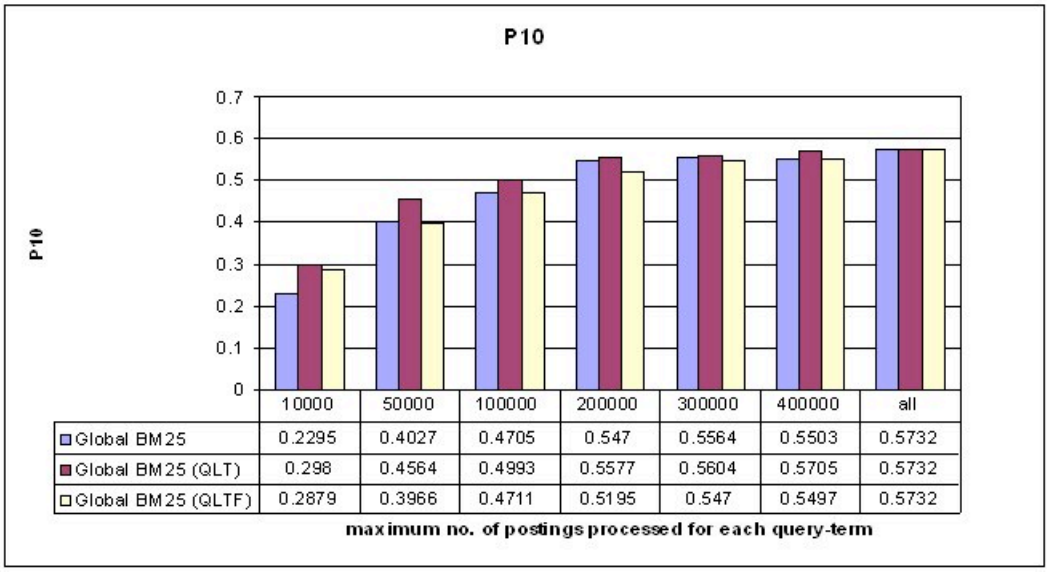


Figure 6.19: P10 performance of global BM25 sortings using query log threshold

approaches in order to effectively sort an inverted index.

For the indegree measure we have simply taken a count of the number of documents that link to a document as its indegree score. The PageRank scores used were calculated over 50 iterations of the PageRank algorithm (which is consistent with the number of iterations that was found by Page et al. (1998) to be sufficient for convergence to occur on this size of collection). Figures 6.20 and 6.21 show the relative performances of these two measures in terms of MAP and P10.

These figures show that indegree performs below that of random sorting, while the PageRank sorting performs above both measures for MAP and P10. This clearly shows the advantage of using PageRank over the simpler linkage measure of indegree to filter documents by.

As mentioned in section 4.2, measures such as PageRank and indegree display a *power-law distribution*, and so their scores are often normalised in order to combine these measure more effectively. However in the case of producing a means by which to sort an inverted index, normalising by taking the log of the score for example has no effect on the global ordering of the documents (only the values change) and so performs the same as using the raw score to sort the index.

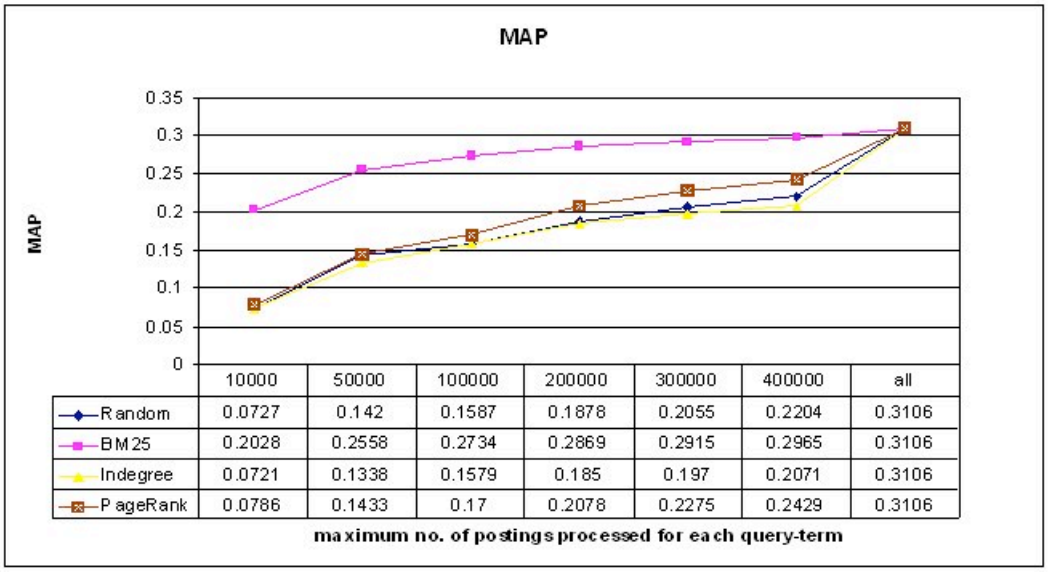


Figure 6.20: MAP performance of linkage analysis sorting

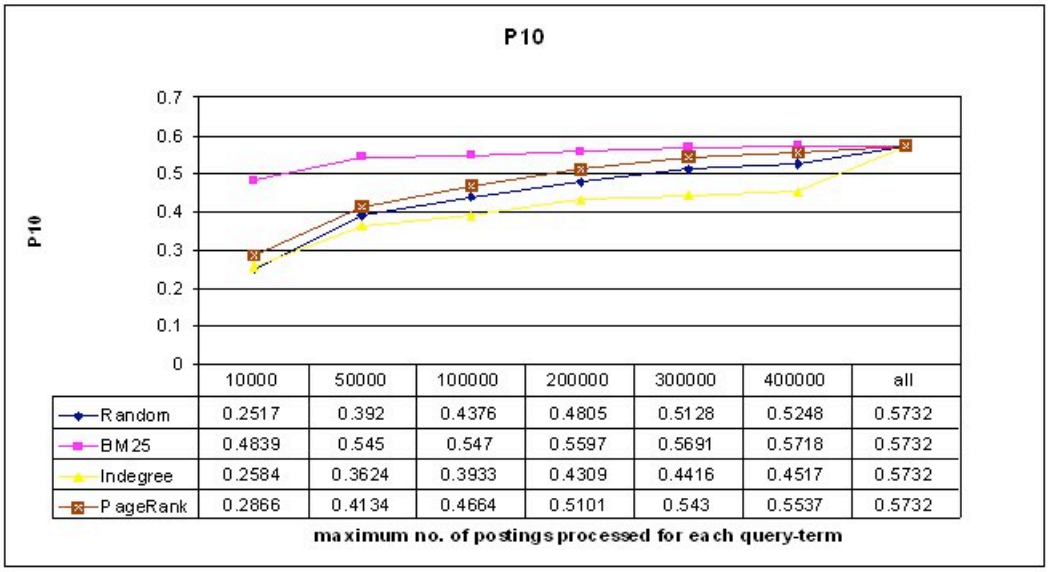
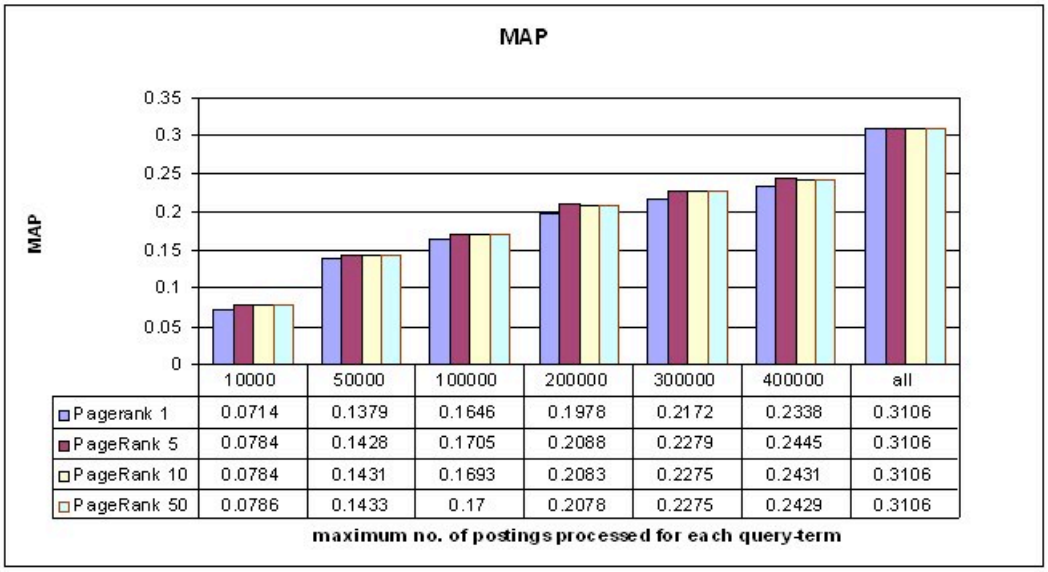


Figure 6.21: P10 performance of linkage analysis sorting

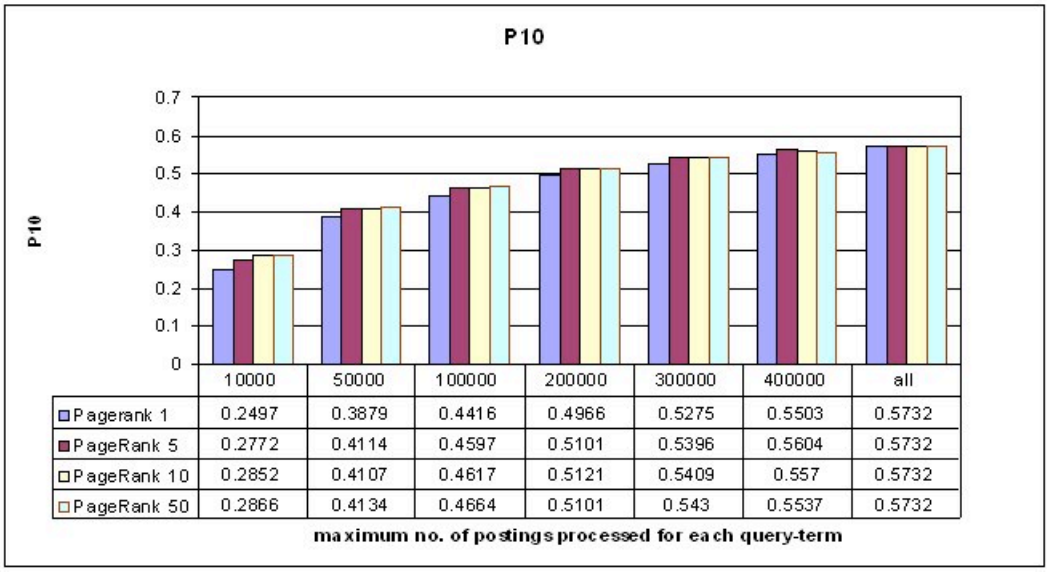
As it is only the ordering of the documents that we want to maintain, it may also be possible to perform fewer iterations of the PageRank calculation, which can be quite time consuming to generate. Consequently we compare how the PageRank sorting differs when calculated over a fewer number of iterations; Figures 6.22 and 6.23 show the performance of PageRank after 1, 5, 10 and 50 iterations.

Surprisingly it is only the PageRank after 1 iteration that suffers most notice-



	10000	50000	100000	200000	300000	400000	all
Pagerank 1	0.0714	0.1379	0.1646	0.1978	0.2172	0.2338	0.3106
PageRank 5	0.0784	0.1428	0.1705	0.2088	0.2279	0.2445	0.3106
PageRank 10	0.0784	0.1431	0.1693	0.2083	0.2275	0.2431	0.3106
PageRank 50	0.0786	0.1433	0.17	0.2078	0.2275	0.2429	0.3106

Figure 6.22: MAP performance of PageRank sorting after varying the number of iterations performed



	10000	50000	100000	200000	300000	400000	all
Pagerank 1	0.2497	0.3879	0.4416	0.4966	0.5275	0.5503	0.5732
PageRank 5	0.2772	0.4114	0.4597	0.5101	0.5396	0.5604	0.5732
PageRank 10	0.2852	0.4107	0.4617	0.5121	0.5409	0.557	0.5732
PageRank 50	0.2866	0.4134	0.4664	0.5101	0.543	0.5537	0.5732

Figure 6.23: P10 performance of PageRank sorting after varying the number of iterations performed

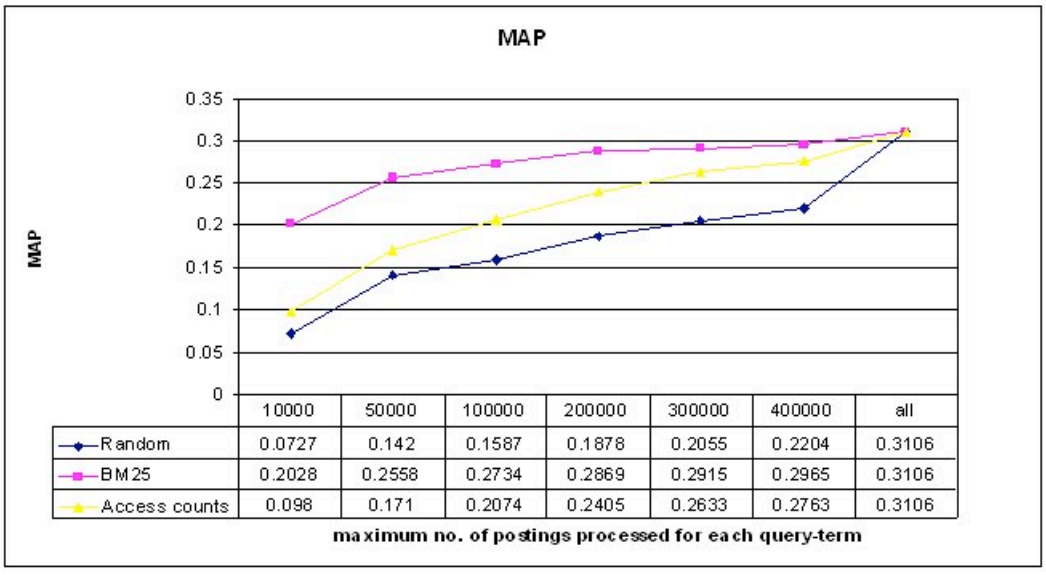
ably, the others, and in particlular the PageRank after 10 iterations performs very similarly to the PageRank after 50 iterations, which takes much longer to generate. Although we have shown that time savings may be made with the reduction of the number of iterations in order to produce a PageRank suitable for the ordering of documents in the inverted index, throughout the thesis we will continue to work

with the PageRank scores as calculated after 50 iterations, for the sake of clarity.

6.5.4 Access Counts

In this section we experiment with the use of access counts, as discussed in section 4.2 as a means to sort the postings by. Garcia et al. (2004) generated a set of access counts by running 1.9 million queries from an *Excite* search engine query log, and incrementing a document's *access count* each time that document was returned within the top 1,000 documents returned in response these queries.

Similarly to Garcia et al. (2004), we provide a baseline access count measure that has been generated from a different *Excite* query log of 2.4 million queries (from December 1999), again incrementing the access count of any document returned within the top 1,000 documents for each of the queries. Figures 6.24 and 6.25 show the performance of this access count measure – which performs relatively well, particularly for precision at 10 documents, considering it is a static query-independent measure.



	10000	50000	100000	200000	300000	400000	all
Random	0.0727	0.142	0.1587	0.1878	0.2055	0.2204	0.3106
BM25	0.2028	0.2558	0.2734	0.2869	0.2915	0.2965	0.3106
Access counts	0.098	0.171	0.2074	0.2405	0.2633	0.2763	0.3106

Figure 6.24: MAP performance of access count sorting

In addition to the default access count measure proposed by Garcia et al. (2004), we would also like to investigate how well this access count approach performs given

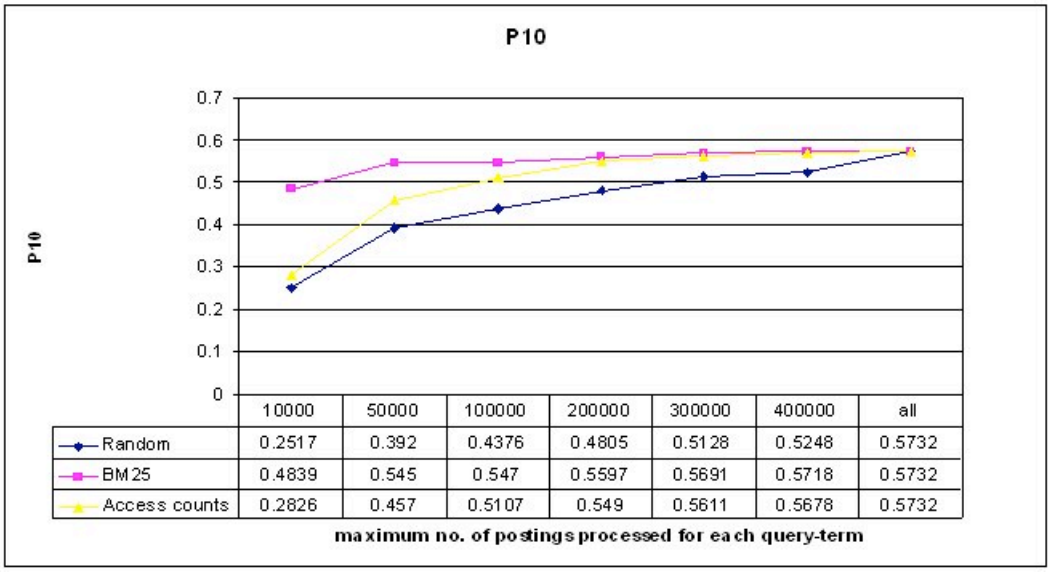


Figure 6.25: P10 performance of access count sorting

less queries, i.e. how many training queries are required in order to provide a consistent performance? For this we created different sets of access counts that were generated by using different numbers of queries; these access counts were generated after 250,000, 1 million, as well as our original 2.4 million queries (note: these smaller query log "subsets" were selected in chronological order from the start of the original query log). Using these we can then compare the performance of this access count approach having different numbers of training queries to evaluate how much this effects performance. Figures 6.26 and 6.27 show the effect this has on the performance in terms of MAP and P10.

These figures show that there is little or no gain made by issuing larger number of queries (of this same type of general web queries at least). There is no significant difference in performance after issuing at least 250,000 queries, which translates to a large saving in the time taken to run this training phase to generate the access count scores.

The other aspect of this access count approach that we wanted to investigate was the choice of the 1,000 boundary for incrementing the access counts for each query. For the typical user query, all of the top 1,000 documents returned by a

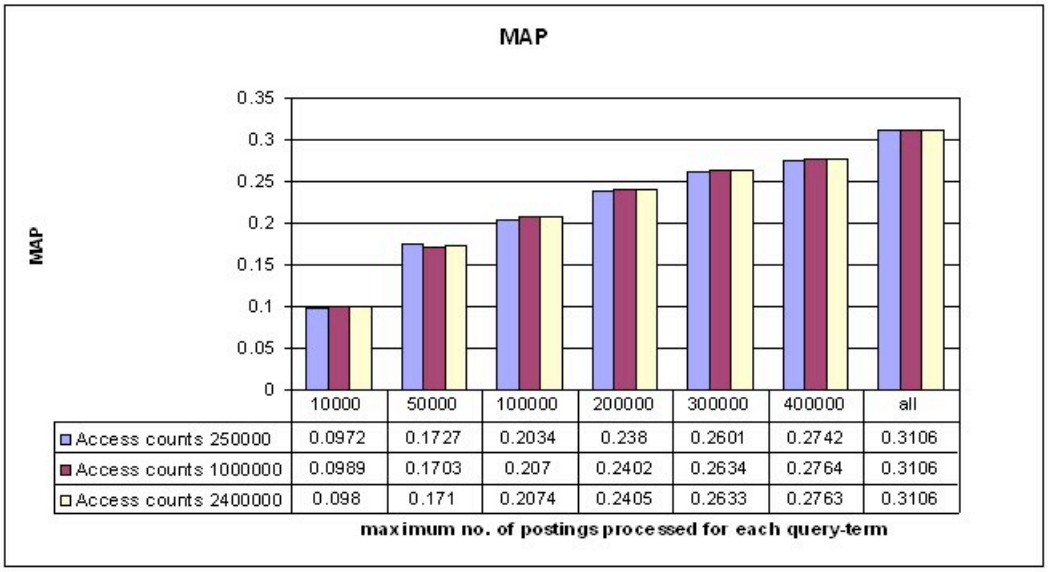


Figure 6.26: MAP performance of access count sorting, with different numbers of training queries

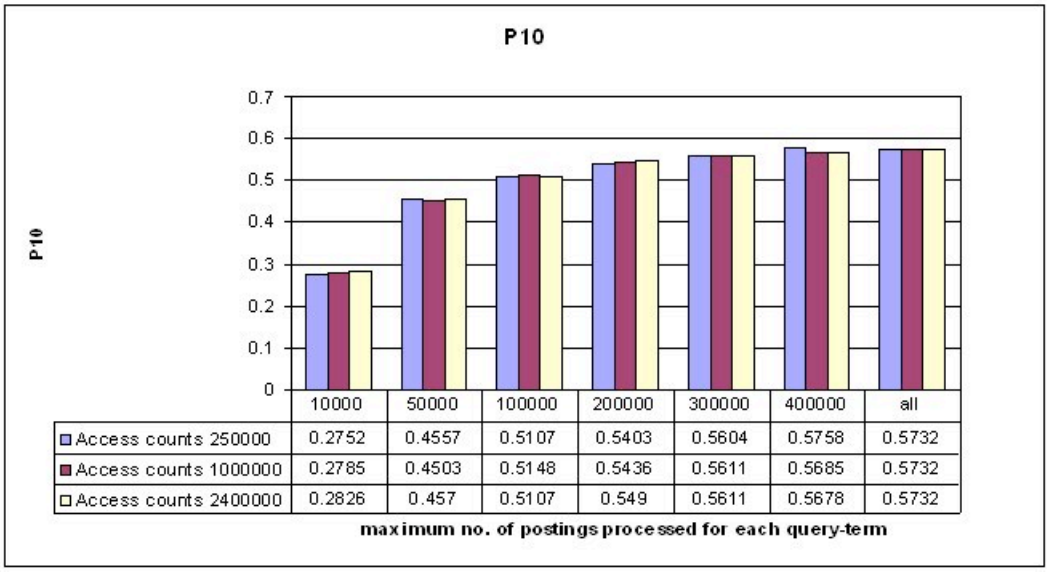
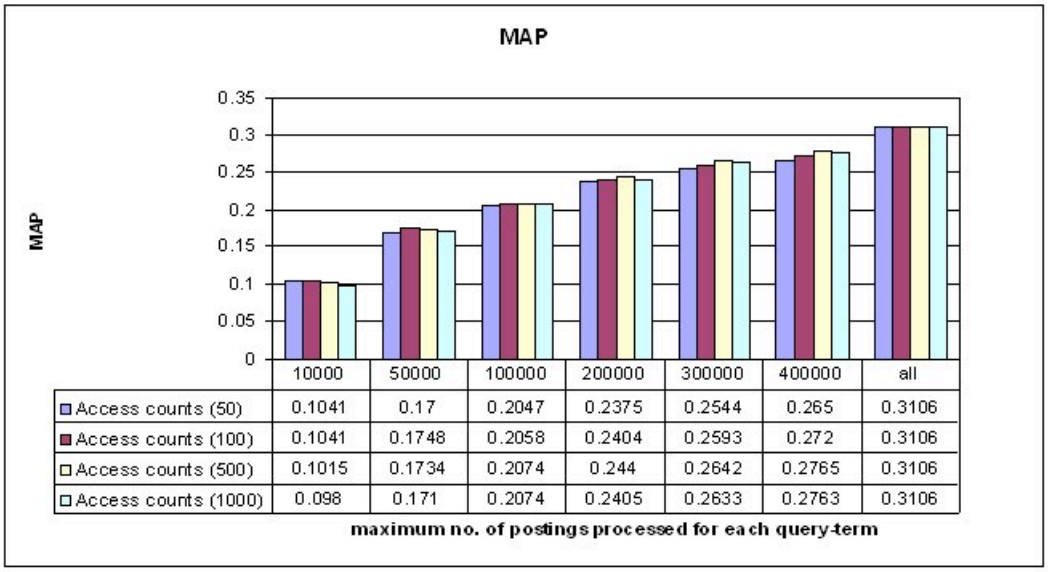


Figure 6.27: P10 performance of access count sorting, with different numbers of training queries

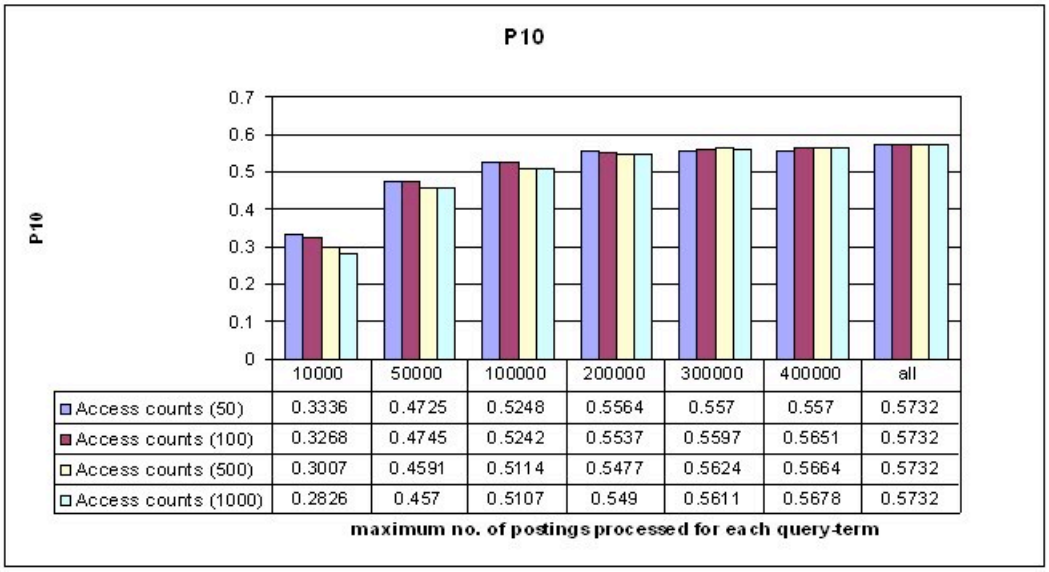
search engine are unlikely to be relevant to the user's query, therefore, perhaps a more conservative number of the top ranked documents should have their access counts incremented in response to the training queries. In order to investigate this further we chose to increment the access count of each document as they occurred within a higher cut-off point than the original 1,000 point in the result list. Figures

6.28 and 6.29 show the performance of these access count measures, as generated at cut-off points of the top 50, 100, 500, as well as the original 1,000, for comparison.



	10000	50000	100000	200000	300000	400000	all
Access counts (50)	0.1041	0.17	0.2047	0.2375	0.2544	0.265	0.3106
Access counts (100)	0.1041	0.1748	0.2058	0.2404	0.2593	0.272	0.3106
Access counts (500)	0.1015	0.1734	0.2074	0.244	0.2642	0.2765	0.3106
Access counts (1000)	0.098	0.171	0.2074	0.2405	0.2633	0.2763	0.3106

Figure 6.28: MAP performance of access count sorting, with different access count thresholds



	10000	50000	100000	200000	300000	400000	all
Access counts (50)	0.3336	0.4725	0.5248	0.5564	0.557	0.557	0.5732
Access counts (100)	0.3268	0.4745	0.5242	0.5537	0.5597	0.5651	0.5732
Access counts (500)	0.3007	0.4591	0.5114	0.5477	0.5624	0.5664	0.5732
Access counts (1000)	0.2826	0.457	0.5107	0.549	0.5611	0.5678	0.5732

Figure 6.29: P10 performance of access count sorting, with different access count thresholds

These figures show that the access count threshold of 1,000 documents is not necessarily the best for producing the most accurate results: in particular for the P10 measure, the lower access count threshold values perform better. This is not

necessarily surprising as the access count approach is essentially marking as relevant all documents within the chosen threshold of the returned result list as being relevant for the particular query. This would most likely mark many documents that are not relevant as being relevant, and the larger the threshold the more likely that more non-relevant documents are marked as relevant. Therefore choosing a smaller threshold value should result in the less non-relevant documents being marked as relevant, however, if less documents are being marked for each query this will also mean that a large number of documents will not receive any access counts and it is worth noting that this has an adverse effect on the recall of the system, as is shown in Figure 6.30.

Having experimented with the way in which these access count values may be effected (and improved upon) for the remainder of our experiments we proceed with using the original access count measure generated using the default threshold of 1,000 documents (as proposed by Garcia et al. (2004)) and generated using the full query-log of 2,400,000 queries.

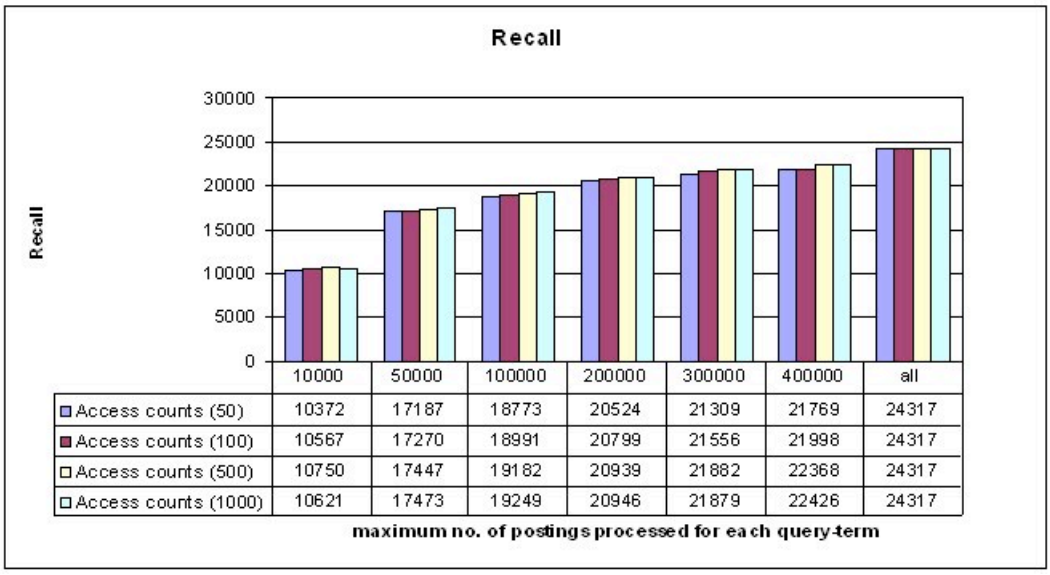
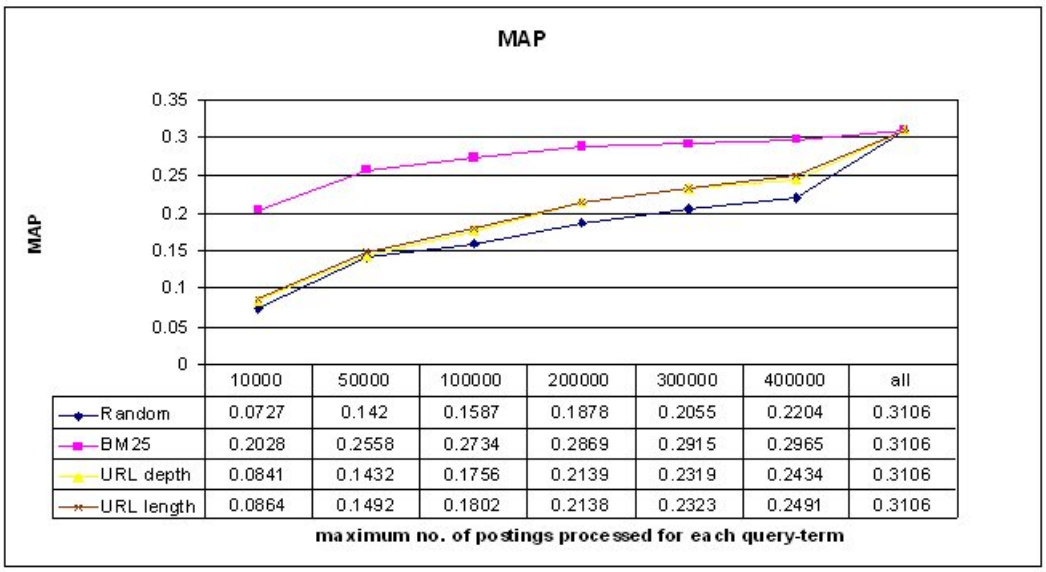


Figure 6.30: Recall performance of access count sorting, with different access count thresholds

6.5.5 URL Information

As outlined in Chapter 4 there is a certain amount of implicit information contained with the URL of a web document. We may view documents with shorter URLs to be more general in terms of content than documents with a longer URL, and so more likely to be relevant to more users' information needs. For this reason we highlighted two simple ways that may be used to estimate this importance: *URL depth* and *URL length*.

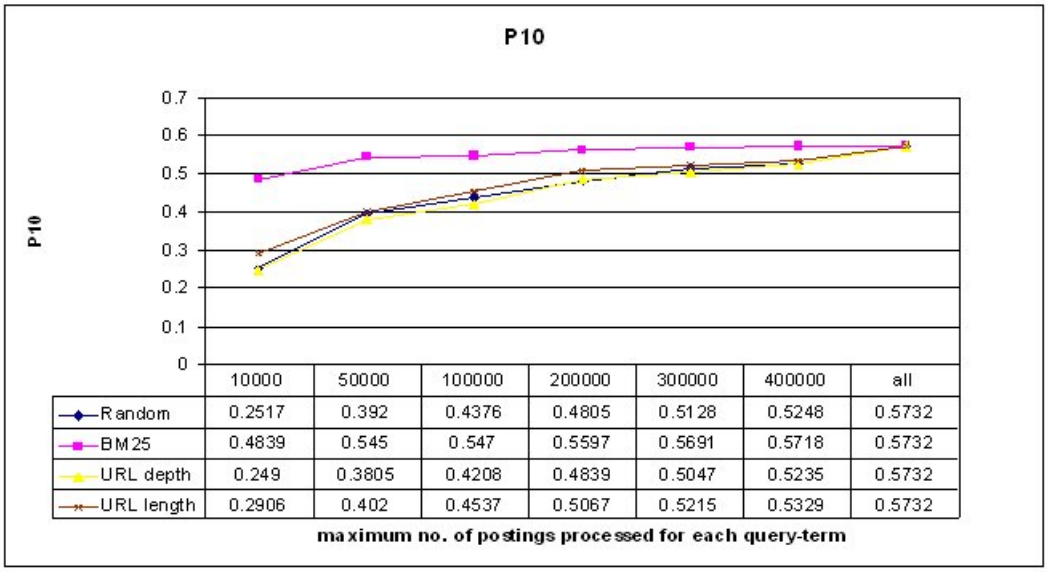
Note that as for these measures it is the documents with a lower score we wish to promote, we therefore use the inverse of these values – sorting the postings lists in descending order of these inverse values. Figures 6.31 and 6.32 show the performance figures for both URL length and depth compared with the the BM25 and randomly sorted indexes.



	10000	50000	100000	200000	300000	400000	all
Random	0.0727	0.142	0.1587	0.1878	0.2055	0.2204	0.3106
BM25	0.2028	0.2558	0.2734	0.2869	0.2915	0.2965	0.3106
URL depth	0.0841	0.1432	0.1756	0.2139	0.2319	0.2434	0.3106
URL length	0.0864	0.1492	0.1802	0.2138	0.2323	0.2491	0.3106

Figure 6.31: MAP performance of URL length and depth

From these figures we can see that for MAP the two measures are quite comparable, however, for P10 the URL length measure performs better (with the URL depth measure falling below random performance on average).



	10000	50000	100000	200000	300000	400000	all
Random	0.2517	0.392	0.4376	0.4805	0.5128	0.5248	0.5732
BM25	0.4839	0.545	0.547	0.5597	0.5691	0.5718	0.5732
URL depth	0.249	0.3805	0.4208	0.4839	0.5047	0.5235	0.5732
URL length	0.2906	0.402	0.4537	0.5067	0.5215	0.5329	0.5732

Figure 6.32: P10 performance of URL length and depth

6.5.6 Information-to-noise ratio

As discussed in Chapter 4, it is believed that a document with a lower information-to-noise ratio score is generally of lower quality than a document with a higher score. It therefore seems like an ideal candidate to sort the postings within an inverted index in order to promote more high quality documents. Again to investigate this we sorted an inverted index using this measure, and again we compare the results against the BM25 and random sortings for MAP and P10 in Figures 6.33 and 6.34 respectively.

These figures show that the infomation-to-noise ratio performs relatively well for both MAP and P10 measures. In fact for P10 at a maximum number of 200,000 postings processed per query-term there is only a drop-off of 2.6% (which is quite good for a static measure) compared with the BM25 measure which generates term-specific scores.

6.5.7 HTML Correctness

In order to investigate how useful the correctness of a document's HTML is at indicating the quality of that document, we firstly processed each document – looking

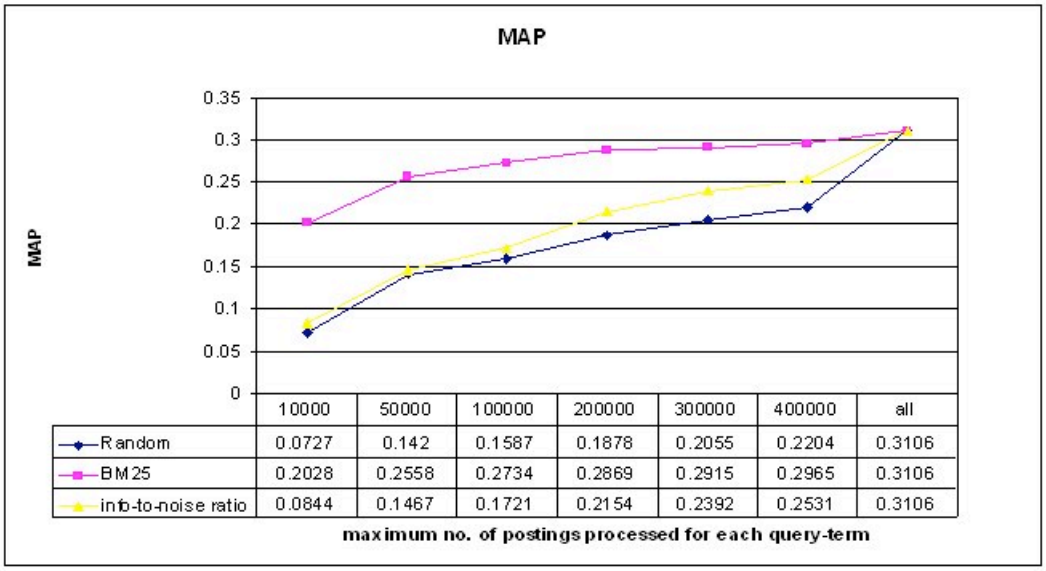


Figure 6.33: MAP performance of information-to-noise ratio

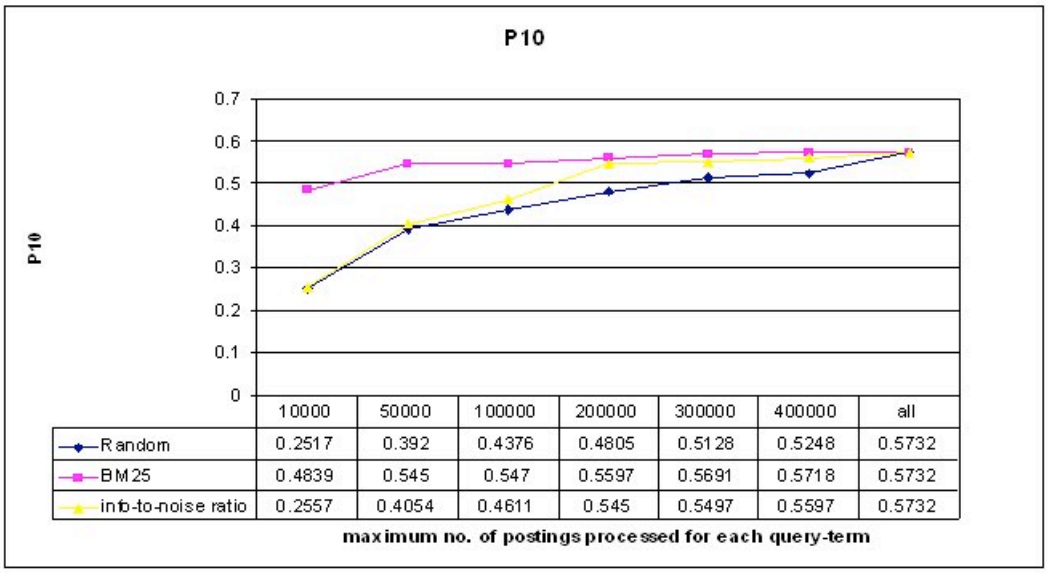
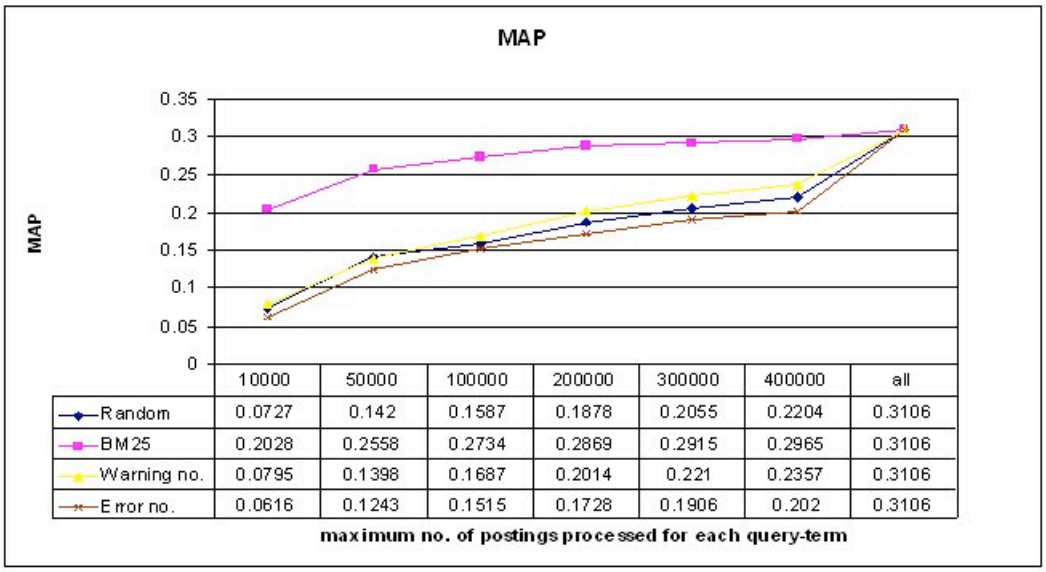


Figure 6.34: P10 performance of information-to-noise ratio

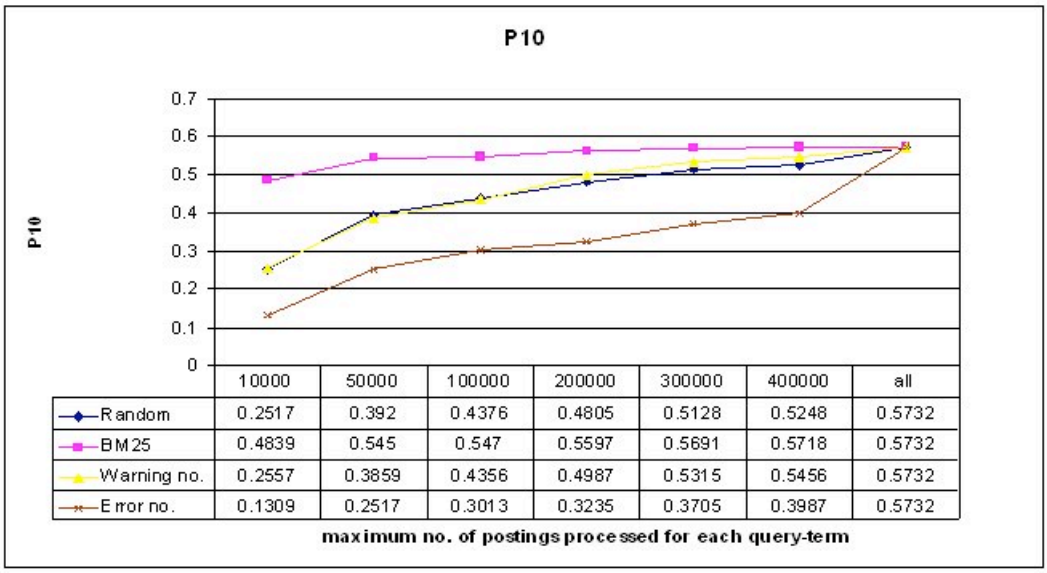
for compliance with the HTML specifications as defined by the World Wide Web Consortium (W3C). This process generated two lists, the number of errors and the number of warnings for each document in the collection: here an error corresponds to a serious error such as a tag being defined in the document that is not in the W3C specifications, whereas a warning corresponds to something less severe such as no closing tag for a specific element. Using these two lists we sorted the inverted index

using the inverse of these two measures, the results of which are shown in Figures 6.35 and 6.36.



	10000	50000	100000	200000	300000	400000	all
Random	0.0727	0.142	0.1587	0.1878	0.2055	0.2204	0.3106
BM25	0.2028	0.2558	0.2734	0.2869	0.2915	0.2965	0.3106
Warning no.	0.0795	0.1398	0.1687	0.2014	0.221	0.2357	0.3106
Error no.	0.0616	0.1243	0.1515	0.1728	0.1906	0.202	0.3106

Figure 6.35: MAP performance of HTML error no. and warning number.



	10000	50000	100000	200000	300000	400000	all
Random	0.2517	0.392	0.4376	0.4805	0.5128	0.5248	0.5732
BM25	0.4839	0.545	0.547	0.5597	0.5691	0.5718	0.5732
Warning no.	0.2557	0.3859	0.4356	0.4987	0.5315	0.5456	0.5732
Error no.	0.1309	0.2517	0.3013	0.3235	0.3705	0.3987	0.5732

Figure 6.36: P10 performance of HTML error no. and warning number.

These figures show that the number of warnings within a document provides a more effective way to sort an inverted index. This may be due to the fact that a large number of documents contain no errors (over 22 million documents), so because of

this there is nothing to discriminate among this large number of documents. Also many documents may not contain warnings, however the number of documents that contain no warnings is much less than that those with no errors (350,000 documents) and so this allows more discrimination to be made between high and low quality documents, as more documents have at least one warning.

On considering this further, its seems intuitive that on ranking a document in order of the number of mistakes in the HTML a more accurate way to estimate a document's quality would be to also consider the amount of HTML that is present in the document: if a document contains a large amount of HTML markup and contains 3 mistakes then this should be ranked higher than a short document that contains only a small amount of HTML markup and yet also contains 3 mistakes. In order to take this into consideration we calculate the amount of HTML markup that the document contains and divide the length of this into the number of errors or warnings that that document contains. Using this approach gives us two more measures: *error rate* and *warning rate* (Figures 6.37 and 6.38 show the performance of these two measures). Surprisingly these two modified measures perform noticeably worse than the original measures. We had originally anticipating that incorporating more information into the calculation would aid in its performance, however this was not the case.

6.5.8 Document Length

In calculating the likelihood that a document is relevant to a query, the length of the document is quite often going to have an impact on this (taking for example ranking methods such as BM25), although the fact that one document is longer than another is not necessarily an indication that one is of higher quality than another. However, if a document contains more information than another it is more likely to be returned as relevant, purely on the basis that there are more words contained in the document. Although we do not expect this to perform relatively highly we wish to also use the document length as an additional measure so that we may see if it

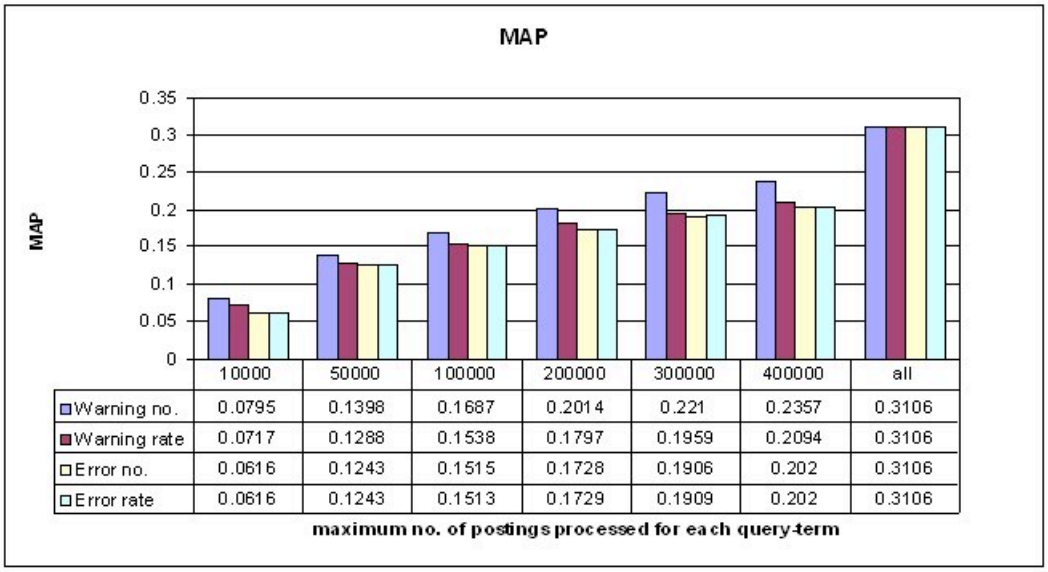


Figure 6.37: MAP performance of HTML error rate. and warning rate.

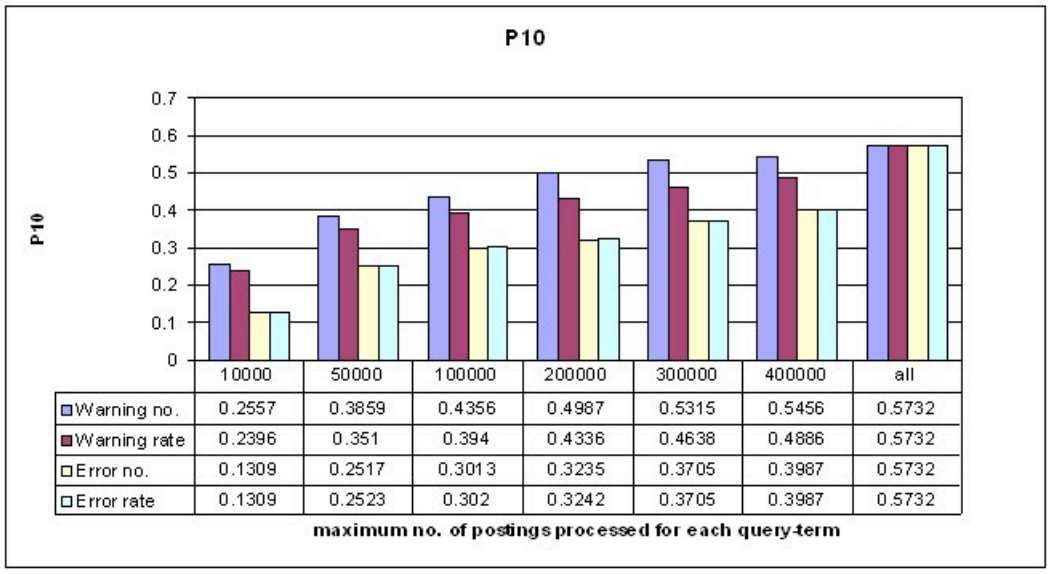


Figure 6.38: P10 performance of HTML error rate. and warning rate.

may be be combined effectively with any of the other measures in order to improve the system performance. Figures 6.39 and 6.40 show the effectiveness of using the document length to sort the inverted index. Interestingly for both MAP and P10 the document length proves to be clearly better than the random sorting measure, and so provides some encouragement that it might be useful for combining with some other measures.

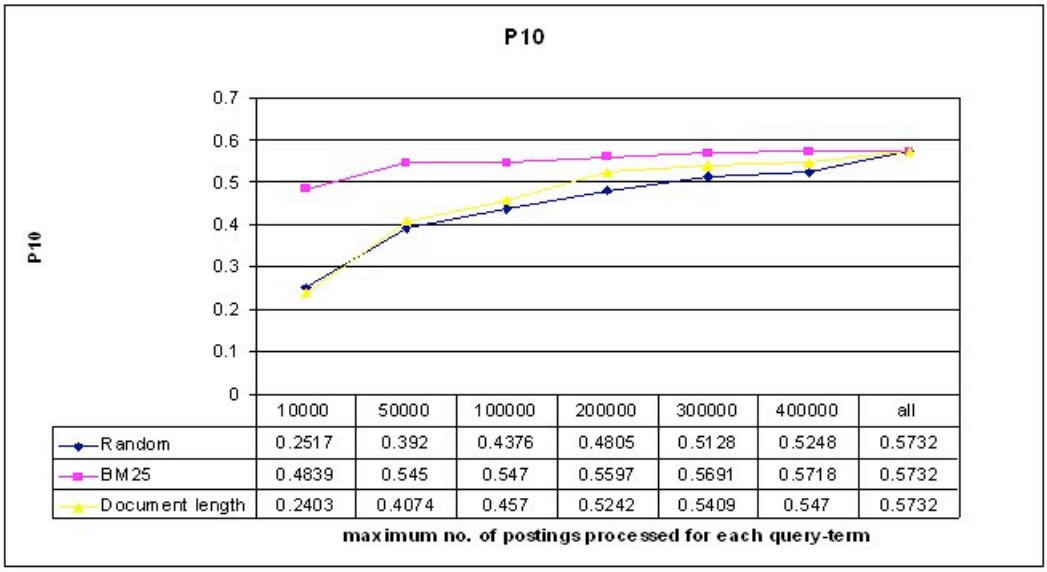


Figure 6.39: MAP performance of document length.

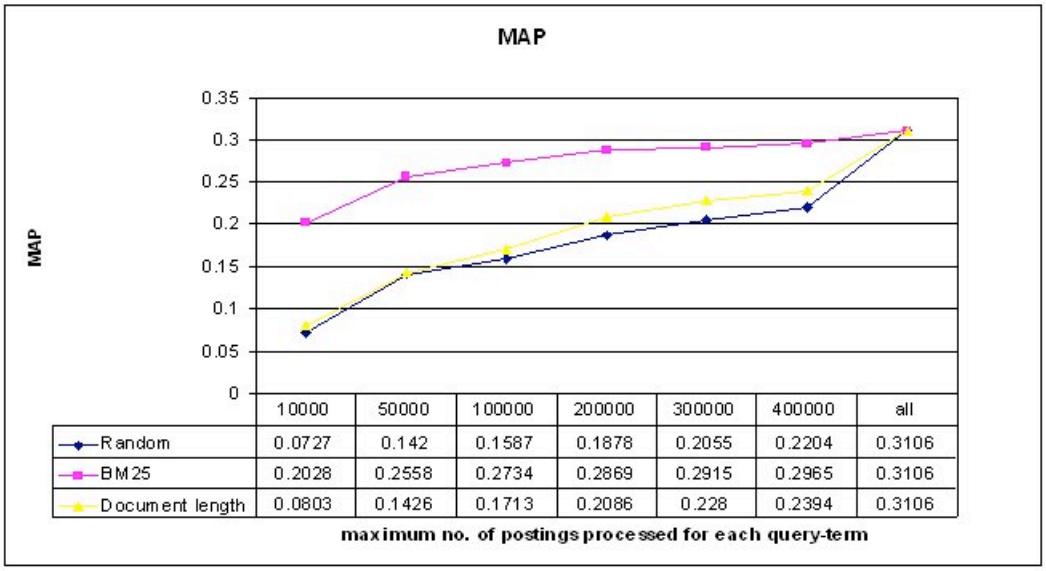


Figure 6.40: P10 performance of document length.

6.5.9 Comparing Standalone Measures

For the purposes of comparison, here we present the average (averaged from each of the cut-off points 10,000, 50,000, 100,000, 200,000, 300,000, 400,000 and all postings) MAP and P10 scores for all individual measures in Tables 6.1 and 6.2 respectively. Here we can clearly see that the term-specfic sortings all perform better than any of the static measures.

Measure	Average MAP
BM25 (term-specific)	**0.2739**
TF (term-specific)	**0.2643**
NTF (term-specific)	**0.2578**
Access counts	0.2239
Global BM25 (QLT)	0.2088
Global BM25 (QLTF)	0.2051
URL length	0.2031
Info-to-noise ratio	0.2031
URL depth	0.2004
Global BM25	0.2001
Document length	0.1973
PageRank	0.1972
Warning no.	0.1938
Random	*0.1854*
Indegree	0.1805
Warning rate	0.1786
Error rate	0.1734
Error no.	0.1733

Table 6.1: Comparison of measure using average MAP

6.6 Index Creation and Evaluation

If we consider that in order to produce an evaluation of a measure, using MAP and P10 we firstly produce a complete sorted inverted index and then run a set of queries through the search system – these are then evaluated using the query relevance judgements (QRELs) provided by TREC. This process is quite time consuming, due, in the most part to the creation of the inverted index, which incidentally also takes up a considerable amount of disk-space. This may be acceptable in evaluating a limited number of query-independent measure, such as those that were shown in the previous section, however when we are to experiment with the combination of these measures, using various different approaches, we shall produce a large number of variants that will need to be evaluated, and this simply is not feasible using the current approach. Therefore we would ideally need to evaluate each new measure without the need for the creation of an addition index.

Firstly we began to look at the current method of evaluating new measures:

Measure	Average P10
BM25 (term-specific)	**0.5500**
TF (term-specific)	**0.5479**
NTF (term-specific)	**0.5310**
Global BM25 (QLT)	0.5022
Access counts	0.5002
Info-to-noise ratio	0.4785
PageRank	0.4781
Global BM25 (QLTF)	0.4779
Global BM25	0.4757
Document length	0.4700
URL length	0.4687
Warning no.	0.4609
Random	*0.4532*
URL depth	0.4479
Warning rate	0.4205
Indegree	0.4159
Error rate	0.3360
Error no.	0.3357

Table 6.2: Comparison of measure using average P10

although this is the strategy used by Garcia et al. (2004), Ferguson et al. (2005c) and Ferguson et al. (2005a) in order to evaluate novel ways of sorting the inverted index, we began to question if this was the best way to evaluate a new measure. Using this approach a new inverted index is created that is sorted using the new measure, the search engine then uses its conventional search algorithm (such as BM25) to rank the documents in response to a query, while the effectiveness of the sorting can be evaluated by limiting the number of postings that are examined in order to produce the ranked list and comparing this with the query performance in terms of MAP and P10 for example. The problem that we see with this approach is that although the index is sorted by a given measure (X), this sorting is then re-sorted by BM25 at query-time, in order to produce the ranked set of results. The sorting of the measure X therefore becomes more and more diluted as the maximum number of postings being evaluated for each term is increased. Also, if for example the sorting produced by X is effective at promoting highly relevant documents, however, if these documents are ranked lowly by BM25, then these will be pushed down its ranked

list and so not allow for a proper evaluation of the sorting produced by X.

Our alternative approach suggests that the system looks only for the known relevant documents for particular queries in the postings lists of the query terms and evaluate how highly X ranks those documents. Using this approach eliminates the diluting of the sorting produced by X when combined with an additional sorting algorithm at query-time. We call this evaluation *Index Precision (IP)*, as it calculates a precision score by analysing the postings lists of the inverted index. This is similar to the way in which traditional precision is calculated for a list of results, except that now the sorting is calculated on the ranked postings instead. This score is calculated over each query-term's postings list as follows:

$$IP = \sum_{r=1}^{n} \frac{nrf_r}{r} \tag{6.3}$$

where n is the number of postings associated with the term being processed, r is the current position (or rank) in the postings list and nrf_r is the number of relevant documents that have been found up to the current rank r. To get the average IP score for a query we divide by the number of postings lists that were evaluated for the query. This process is illustrated in greater detail in Figure 6.41.

In addition to the benefit of providing a precision score directly correlating to how highly a sorting measure ranks relevant documents it also dramatically reduces the time required to evaluate a new sorting measure X. This reduction is achieved as we can evaluate this without the need for creating a new inverted index. This can be done by utilising a current inverted index, then at query-time re-ranking the postings list that is being processed by X – so that it can be processed as X would rank the postings. This process is shown in Figure 6.42.

Now examining the average IP scores for the same measures as before, in Table 6.3 we can see that again the BM25 measure comes out on top, but now it is more evident that there is a large difference between the term-specfic scores (in particular BM25 and TF) and the static measures. This is not necessarily surprising, since

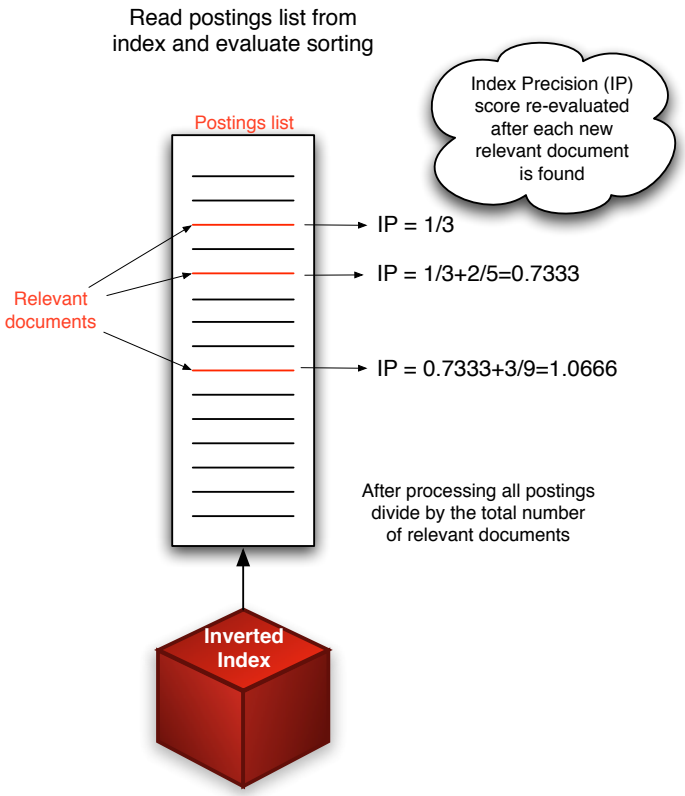


Figure 6.41: Calculating Index Precision (IP).

these term-specific sorting measures use a different sorting depending on the term, however it does illustrate the importance of the incorporation of the TF component in this term-specific way, as this alone proves to be a highly effective sorting measure.

This process of re-sorting an existing inverted index in order to evaluate a new measure now provides us with a means to quickly evaluate a new measure which is essential in order to experiment with combining different sorting measures.

Firstly in order to combine these measures we decided to eliminate a certain number of measures – as there are some measures that are variants on others and we choose to combine the stronger of these measures (based on the IP scores), for these reason we choose to eliminate the following:

- Global BM25 – due to the inclusion of the higher performing global BM25 (QLTF).

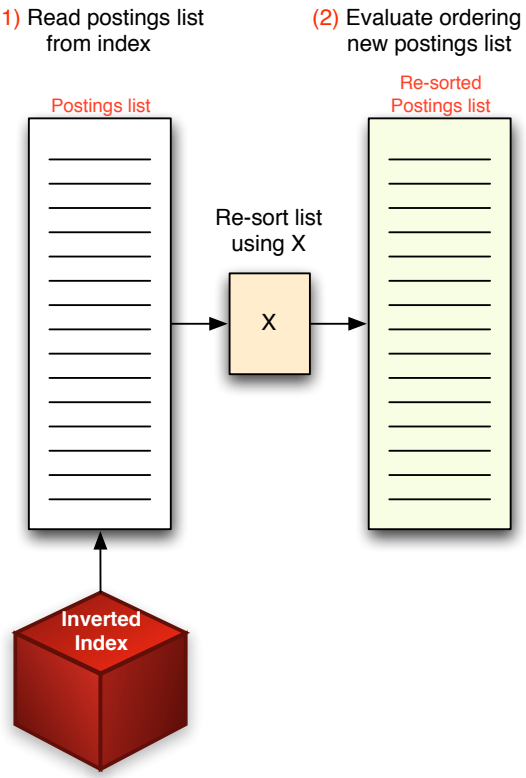


Figure 6.42: Re-sorting postings list.

- Global BM25 (QLT) – due to the inclusion of the higher performing global BM25 (QLTF).
- URL Depth – due to the inclusion of the higher performing URL length.
- Warning rate – due to the inclusion of the higher performing warning number.
- Error rate – due to the inclusion of the higher performing error number.

In our initial combination experiments we concentrate on combining only the static measures, so that we may gain a more effective static measure that may be combined with the term-specific scores. This leaves us with the following measures to combine:

- Access counts
- Document length

Measure	**Average IP Scores**
BM25	0.04349
TF	0.04000
NTF	0.02078
Access counts	0.00916
URL length	0.00879
Global BM25 (QLTF)	0.00877
URL depth	0.00853
Info-to-noise ratio	0.00824
Indegree	0.00815
PageRank	0.00786
Global BM25 (QLT)	0.00779
Warning no.	0.00774
Error no.	0.00749
Error rate	0.00749
Random	*0.00744*
Document length	0.00744
Warning rate	0.00717
Global BM25	0.00670

Table 6.3: Comparison of measure using average IP score

- Error number
- Global BM25 (QLTF)
- Indegree
- Information-to-noise ratio
- PageRank
- URL length
- Warning number

For our initial experiments we deal with combining these measures using all approaches except for the SVM combination which is significantly different from the others and so we describe it separately – for the other methods we use the same combination strategies and so we discuss these approaches together. We experi-

ment with these combination approaches in subsection 6.7.1 and experiment with combining using SVMs in subsection 6.7.2.

6.7 Combination

In Chapter 5 we discussed different ways in which we could combine various query-independent measures together, in the hope of producing a more effective single measure. To summarise from Chapter 5, the main approaches that we discussed for combining these sources are:

- Rank and Score-based Combination
  - Score-based Combination
    - Similarity Merge (specifically CombSUM)
    - Linear Combination
  - Rank-based Fusion
    - Rank Fusion
    - Weighted Rank Fusion
- Dempster-Shafer Combination
- Support Vector Machine Combination

In this section we will evaluate the usefulness of each of these methods, to combine our query-independent measures.

6.7.1 Combination Strategies

In principle we would like to combine all our query-independent measures together in every possible way, with all possible weights, however, due to the computational cost of doing this, as well as re-sorting the inverted index in order to evaluate each of these, alternatively we chose to consider a number of alternative *combination*

strategies. These were designed to get the maximum gain from the combination of the single measures, while at the same time limiting the number of combinations that needed to be evaluated – due to the computational demands involved in the evaluation process.

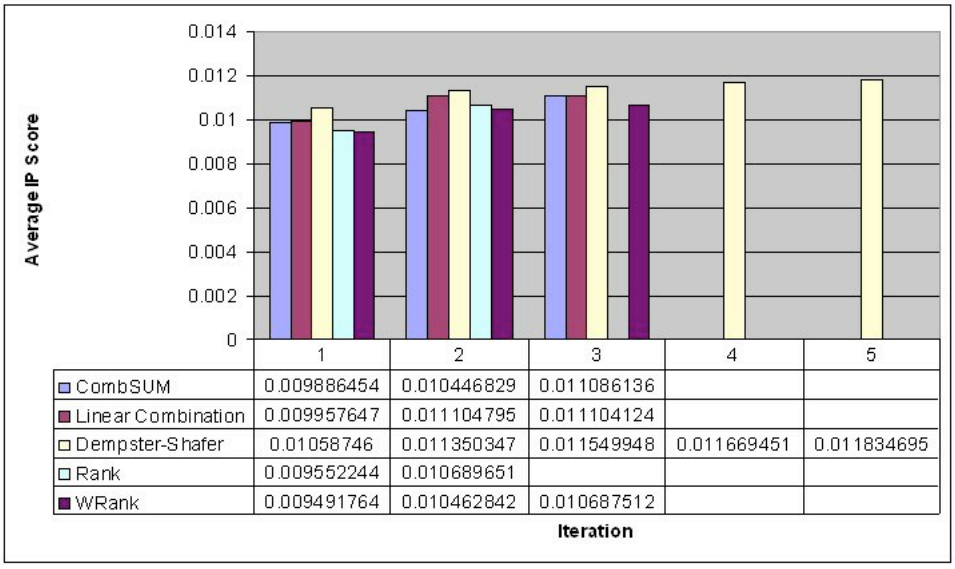
6.7.1.1 Leave-One-Out Strategy

This initial strategy attempts to identify the weakest measures that are being combined using the different combination strategies and then eliminates these measures from combination in order to find the measures that produce the most effective combination.

This involves an iterative process where we firstly combine all measures together using a given combination approach (such as CombSUM), then after each iteration we eliminate any measure(s), whose inclusion degrades the overall performance. The way we evaluate which measure(s) degrade the overall performance is to run all permutations in which we *leave out* each of the measures in turn. For example if we are combining the measure x, y, and z, we would firstly combine all three (xyz) and evaluate its effectiveness, then in the first iteration we would evaluate xy, xz and yz, from these would would then see which measures degrade the performance of the xyz combination, and then eliminate these from the next iteration. The process then stops if there is only one measure left or if none of the measures degrades the performance.

In order to test the effectiveness of this strategy (at each iteration) we run these combinations on a set of 100 queries (topics 701-800) from the TREC terabyte ad hoc task, and then evaluate their effectiveness (based on their average IP score) on a different set of 50 queries (topics 801-850), also from the same task. For any of the methods that require weights for their combination (such as the weighted rank approach) we generate weights for each individual measure based on their own performance on the same set of topics (using their IP score), then scaling the weights so that they sum to 1.

Figure 6.43 shows the performance of the the different methods of combination using this leave-one-out strategy. In this figure it can be seen that the Dempster-Shafer methods provides the best overall performance in terms of average IP score.

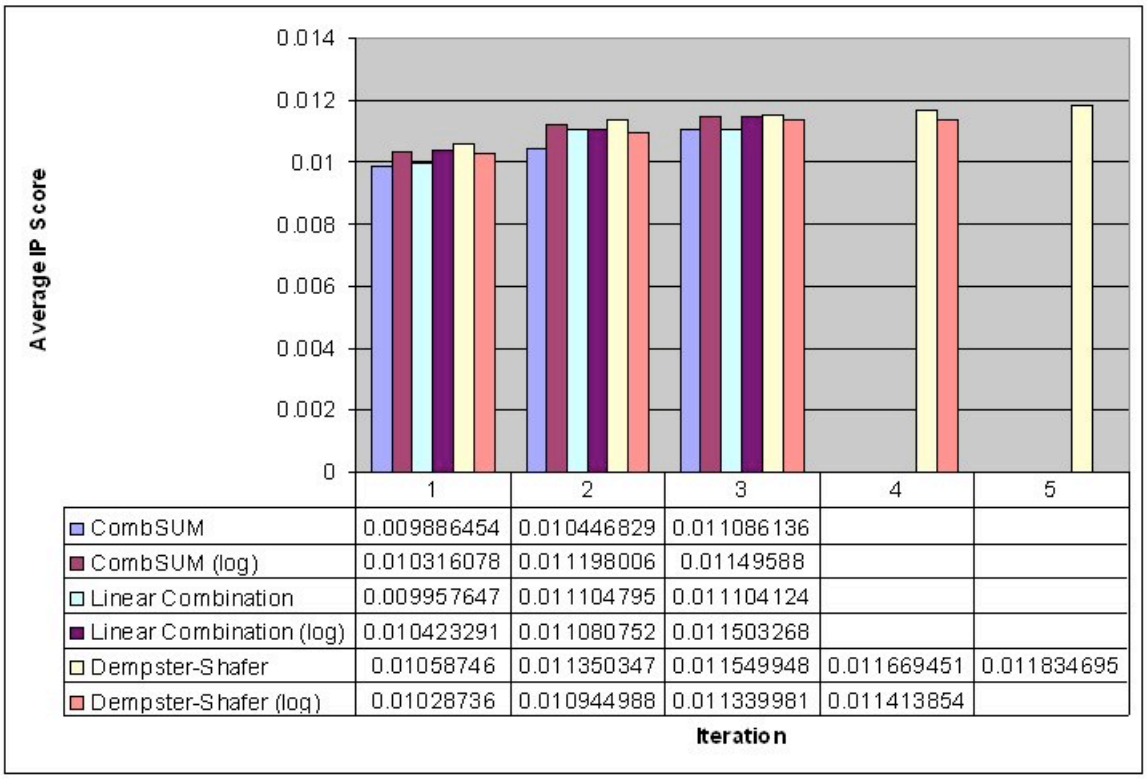


	1	2	3	4	5
CombSUM	0.009886454	0.010446829	0.011086136		
Linear Combination	0.009957647	0.011104795	0.011104124		
Dempster-Shafer	0.01058746	0.011350347	0.011549948	0.011669451	0.011834695
Rank	0.009552244	0.010689651			
WRank	0.009491764	0.010462842	0.010687512		

Figure 6.43: Combining using leave one out strategy.

As described in Chapter 4 certain measures have a power-law distribution, meaning that a small number of documents have very high scores, while the majority of the documents have a very low score. A commonly used approach for normalising these distributions is to take the mathematical log of their scores. Figure 6.44 shows the performance where we use the log values for indegree, PageRank and access counts (while the other measures remain the same). This figure shows that using these log values helps to improve the performance for the CombSUM as well as the linear combination methods, however, this deteriorates the performance of the Dempster-Shafer combination method.

Overall the Dempster-Shafer method (without using the log values) outperforms the rest of the combination methods.



	1	2	3	4	5
CombSUM	0.009886454	0.010446829	0.011086136		
CombSUM (log)	0.010316078	0.011198006	0.01149588		
Linear Combination	0.009957647	0.011104795	0.011104124		
Linear Combination (log)	0.010423291	0.011080752	0.011503268		
Dempster-Shafer	0.01058746	0.011350347	0.011549948	0.011669451	0.011834695
Dempster-Shafer (log)	0.01028736	0.010944988	0.011339981	0.011413854	

Figure 6.44: Combining using leave-one-out strategy (log).

6.7.1.2 Pair-wise Combination

Ideally we would like to exhaustively combine all measures together (which for certain methods require weights, and so this would also involve combining each of these with a number of different weights) – needless to say this would be extremely computationally expensive, especially if we consider the effort involved in re-sorting the inverted index in order to evaluate each combination.

If we are combining two source of information, one of which performs well, and the other which performs poorly, in general the lower performing measure will degrade the overall performance. For this reason we believe that for most cases, combining the lower performing measures such as *error no.* with *access counts* for example would not result in a more effective combined measure. This was also one of the reasons why we choose to firstly combine the lower performing static measures together and then combine the best of these with the term-specific measures. Therefore in another attempt to combine these measure we choose to do so by combining

them in a pair-wise fashion, from lowest performing to highest performing (as shown in Table 6.4). In this way we hope to gradually increase the performance of the combined measures and so providing a more effective measure to combine with the most effective single measures. So as an alternative to the leave-one-out approach, which entirely eliminates a measure if it degrades the combination, this approach hopes to combine the measures in such a way that more measures can contribute without degrading the overall performance. Although in general we combine from lowest to highest we choose to group similar measures together: *warning no.* and *error no.*; *indegree* and *PageRank*; *global BM25 (QLTF)* and *access counts*.

Measure	**Average IP Scores**
Access counts	0.01183
Global BM25 (QLTF)	0.01109
URL length	0.01058
Info-to-noise ratio	0.01040
Warning no.	0.00987
Document length	0.00980
Indegree	0.00978
Error no.	0.00942
PageRank	0.00890

Table 6.4: Comparison of measure using average IP score (topics 701-800)

Figure 6.45 shows the combination *steps* (from 1 to 8) that are involved. Note that when combining two measure together using a method that uses weights (such as *linear combination*) we combine the two measures together using a range of weights from 0 to 1 (in increments of 0.1) so that the sum of the two weights sums to 1 – using this approach there are 11 combinations performed in all at each step, and the highest performing combination is retained for the next *step*). For example, if combining the two measures x and y using *linear combination* (which requires weights for each source), we give varying amounts of weight to each input in order to evaluate the optimum weights for this combination, so the 11 different combinations that this produces is shown (with their appropriate weights) in Table 6.5.

Combination no.	Weight x	Weight y
1	0	1
2	0.1	0.9
3	0.2	0.8
4	0.3	0.7
5	0.4	0.6
6	0.5	0.5
7	0.6	0.4
8	0.7	0.3
9	0.8	0.2
10	0.9	0.1
11	1	0

Table 6.5: Weights used for pair-wise combinations.

Based on our experiments with the *leave-one-out* strategy, we decided to drop the *CombSUM* and *Rank* methods, as not only did these methods perform poorly, but they are essentially special cases of the *Linear Combination* and *WRank* methods respectively (where appropriate weights have been chosen). Therefore, for these experiments we concentrate on the methods Linear Combination, Dempster-Shafer and WRank. Again we run the experiments on the same TREC topics as before (701-800).

Figure 6.46 shows the results of applying this pair-wise strategy. Firstly the WRank approach is unable to gain an improvement from the combination of any of the measures, and so continues with the strongest of the two measures at each combination step, we can therefore use this as a baseline to see how much improvement that each of the other approaches can achieve. We can see from Figure 6.46 that at each of the combination steps both the linear combination and the Dempster-Shafer methods are quite comparable – with the Dempster-Shafer approach again proving to be the most effective. Overall we can see that with the final outcome there is little to distinguish between the different types of combinations (Linear Combination, *Dempster-Shafer*, *WRank*), this is essentially down to their inability to improve upon the highest performing individual measure (only Dempster-Shafer gaining an improvement over the highest performing single measures).

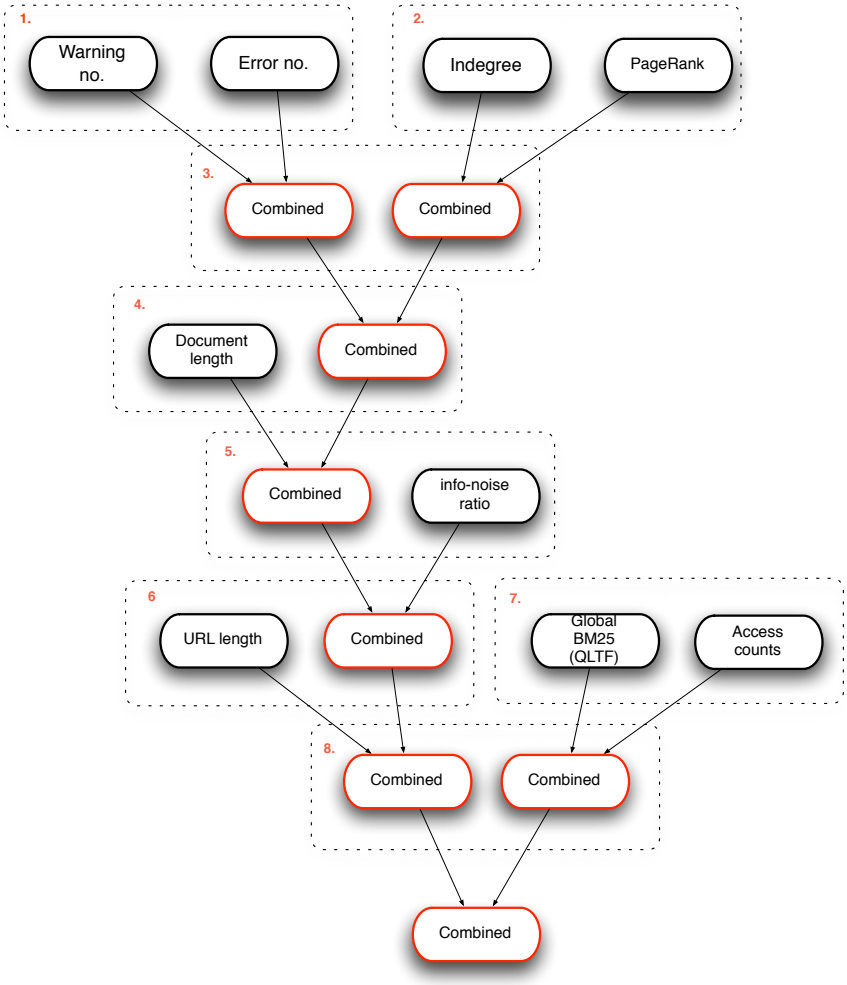
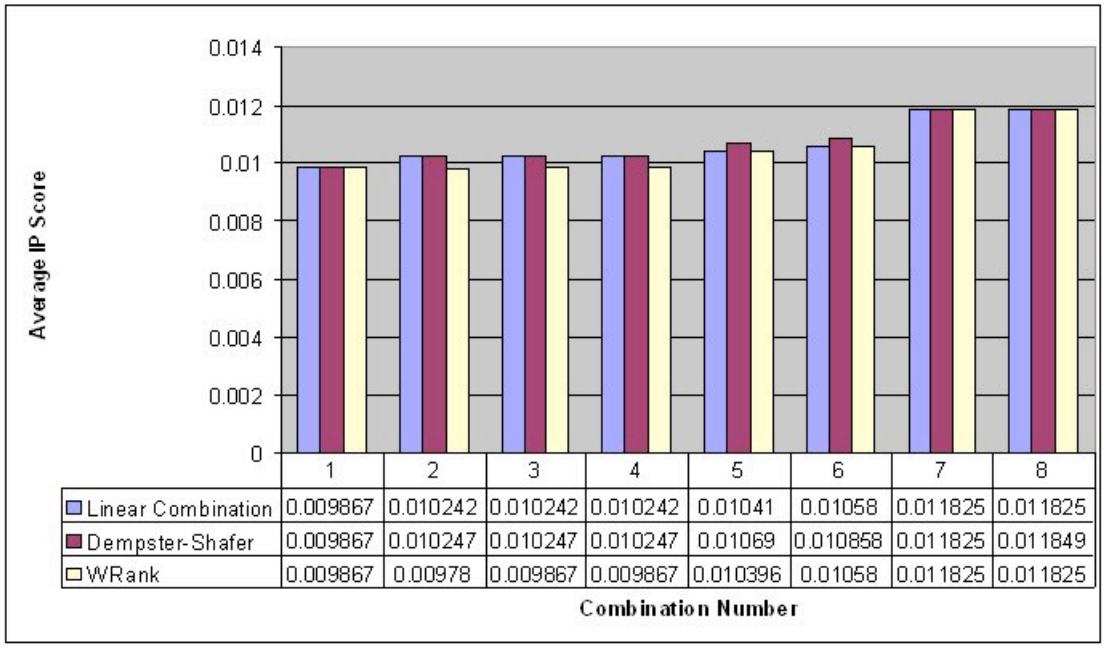


Figure 6.45: Pair-wise combination strategy.

6.7.2 SVM Combination

As discussed in Chapter 5, we may also use SVMs to combine sources of evidence. Here we shall combine the same measures as used with the other methods of combination – to allow meaningful comparisons to be made between the combination methods.

As we discussed in Chapter 5, we wish to essentially use the SVM to classify the documents into relevant and non-relevant sets. One of the major difficulties to overcome when using this approach is how to generate representative sets of relevant and non-relevant documents, in order to generate an effective SVM model to classify the documents in the collection. For this we must choose a source of



	1	2	3	4	5	6	7	8
Linear Combination	0.009867	0.010242	0.010242	0.010242	0.01041	0.01058	0.011825	0.011825
Dempster-Shafer	0.009867	0.010247	0.010247	0.010247	0.01069	0.010858	0.011825	0.011849
WRank	0.009867	0.00978	0.009867	0.009867	0.010396	0.01058	0.011825	0.011825

Figure 6.46: Pair-wise combination results.

representative high quality documents, as well as a source of low quality documents from the collection. Figure 6.47, then shows that from these two sources we then take two different sets of documents that are used to build the SVM's training and testing files.

The issue of representative high quality documents can be solved by using the known-relevant documents associated with a set of TREC topics, using this we generated a set of *positive training* examples from the TREC topics (701-750) and a different set of *positive testing* examples from a different set of TREC topics (751-800). The problem of generating a set of *negative* (i.e. low quality) documents seems a more difficult prospect. As a first attempt we could attempt to use the documents that are in returned by TREC for each topic marked as non-relevant. However, we would not necessarily consider these to be representative low quality documents as these are only included in the *query relevance judgements* as these documents were returned by at least one group's system in response to a particular topic. So although these documents are not relevant for the specified topic they should not be considered as the least likely to be relevant (of all documents) to any

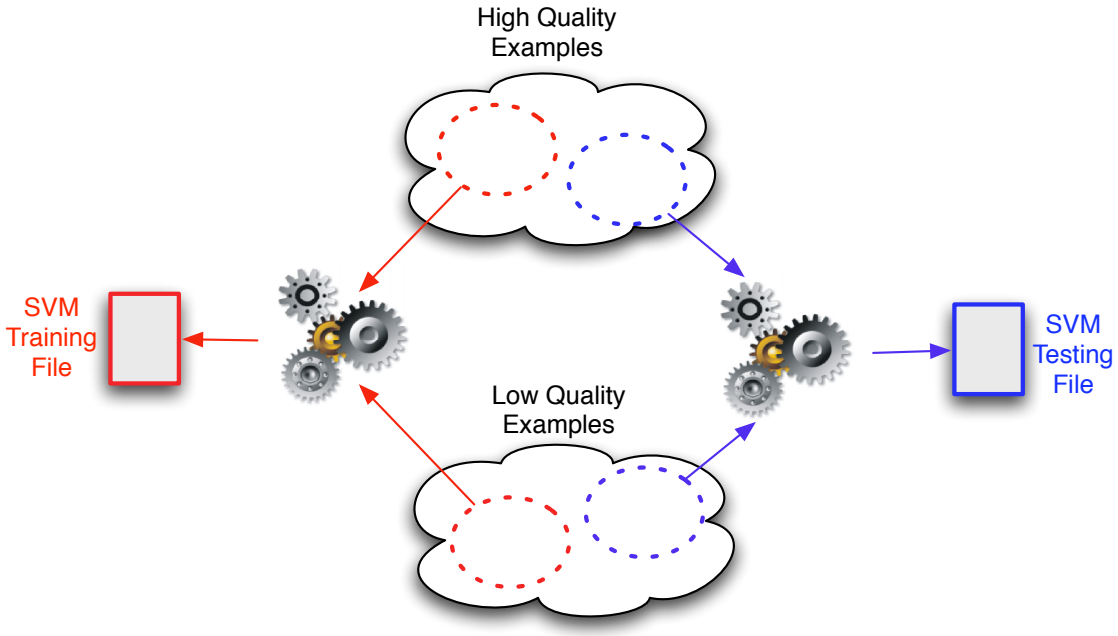


Figure 6.47: Generating SVM training and testing files.

topic. One possible way of generating more representative negative examples would be to look again at the access count measure – however, now we are looking for the the documents with the least number of access counts. There are actually a large number of documents that do not receive any access count for the 2.4 million queries that we used and so from these we randomly generate two different lists (*negative training* and *negative testing*) from this set of documents with no access counts. As an alternative to these negative examples we also generated *negative training* and *negative testing* sets in a similar way, from the lowest scoring documents in the global BM25 (QLTF) measure – which are the lowest scoring documents according to this global BM25 measure. In order to assess the effectiveness of these negative examples we also generated alternative negative sets for training and testing using a random selection of documents from the entire collection. Note that all these negative example sets are the same size as the positive example sets to give an equal distribution of positive and negative documents – in order to prevent the class imbalance problem (Probost, 2000). This leaves us with four alternative training and

testing sets, where the positive examples are the same but the negative documents differ:

- Positive + Negative QRELs (PN)
- Positive + Access counts (PAC)
- Positive + Global BM25 QLTF (PBM)
- Positive + Random (PR)

We firstly generate an SVM model by assessing the models effectiveness in correctly classifying documents in the testing set.

6.7.2.1 Classification Training

The conventional approach for training an SVM for the task of classification is to generate an SVM model using the training set of positive and negative example documents, then use this model to classify the documents in the testing sets. Based on the accuracy of the model at classifing the documents in the testing set we can assess the accuracy of the model (for classification purposes) and tune the SVM's parameters accordingly. Using SVM^{light} (Joachims, 1999) with a linear kernel allows us two parameters to tune: c which controls how strictly to enforce correct classification when generating the model; j which controls by how much positive training examples outweigh negative training examples. Figure 6.48 shows the results of using our different approaches for generating the training and testing sets and shows how accurately the SVM can predict which class (positive or negative) to put the documents from the testing sets into.
The parameter values for these models are as follows:

- PN: c=1, j=1.
- PAC: c=4096, j=1.
- PBM: c=8192, j=4.

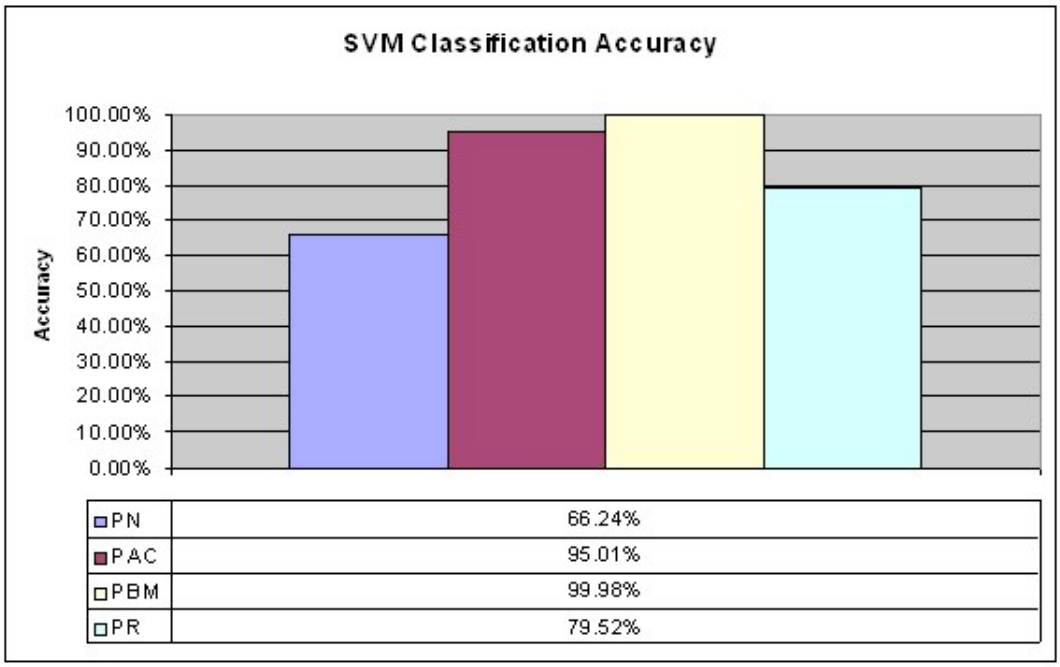


Figure 6.48: SVM Classification Accuracy.

- PR: c=2048, j=1.

As Figure 6.48 shows, the SVM which used the global BM25 (QLTF) measure to generate the negative examples provided the best classification, followed by the Access counts, followed by the random documents and finally the negative QRELs. These results are not surprising, particularly if we consider what the SVM is trying to do – separate the positive examples from the negative examples. It seems intuitive for example that there should be very little to separate between the positive examples and those marked as non-relevant in the QRELs and so this is the case, as the SVM finds it difficult to separate these. This is in contrast with both the access counts and the global BM25 (QLTF) documents, where the SVM can find a clear separation, as is highlighted with a high level of classification accuracy.

Using each of these SVM models we can then *classify* all the documents in the collection (as described in Chapter 5), which gives us a likelihood score for each document that they should belong to the positive or negative class. We then use these scores as a new query-independent measure, which we can use to re-sort an index and evaluate their IP scores.

In order to evaluate these we then calculated the average IP scores for each of these over the same topics that we used in the testing phase for the SVMs (topics 751-800). These average IP scores are displayed in Figure 6.49, and in order to make these figures more meaningful Table 6.6 displays the the average IP scores for all the measures.

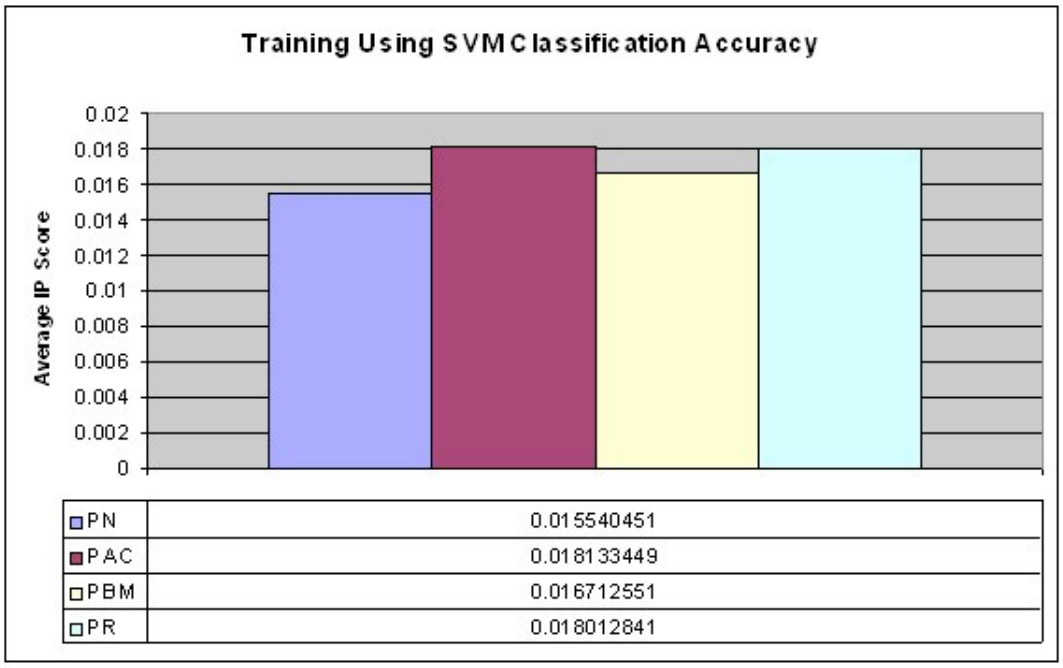


Figure 6.49: IP Scores trained using classification accuracy.

Measure	**Average IP Scores**
Access counts	0.01874
Global BM25 (QLTF)	0.01725
URL length	0.01719
Info-to-noise ratio	0.01705
PageRank	0.01698
Indegree	0.01678
Warning no.	0.01666
Error no.	0.01589
Random	0.01570
Document length	0.01564

Table 6.6: Comparison of measures using average IP score (topics 751-800)

As we can see from these average IP scores in Figure 6.49, despite the high

performance of PBM and PAC in classifying their corresponding testing examples, this does not translate into high performance in terms of sorting an inverted index.

These results may not be due to the poor performance of the SVMs themselves, but rather to our *over-training* for the task of classification. If we take for example the access counts SVM with a c value of 4096, we can see that this is highly tuned so that it imposes a high penalty for misclassified documents – this may work well in this case where the training and testing documents are composed from similar types of documents, however, ultimately we wish to classify all documents for the task of effectively sorting an index, and not for classifying similar types of documents. In order to gain more effective SVM models we should then turn our attention from tuning based on the classification performance, to tuning based on the sorting performance, and in order to measure this (sorting performance) we shall use the IP measure.

6.7.2.2 IP Training

This is similar to the previous experiment in that we shall use the same SVM training files, however, now instead of running the SVM classification process on the appropriate testing set we classify all the documents in the collection and generate a query-independent measure. We then evaluate the effectiveness of this measure by re-sorting the inverted index using this measure and calculate an average IP score over a set of topics. This is obviously a much more computationally expensive process, as we generate several query-independent measures for each approach (PAC, PBM and PR) in order to tune the SVM's parameters, however, we believe that this will give us a more effective sorting measure, as it is tuned for our specific task.
When we tune the SVM parameters using this approach, this gives the following values:

- PAC: c=16, j=1.
- PBM: c=16, j=1.

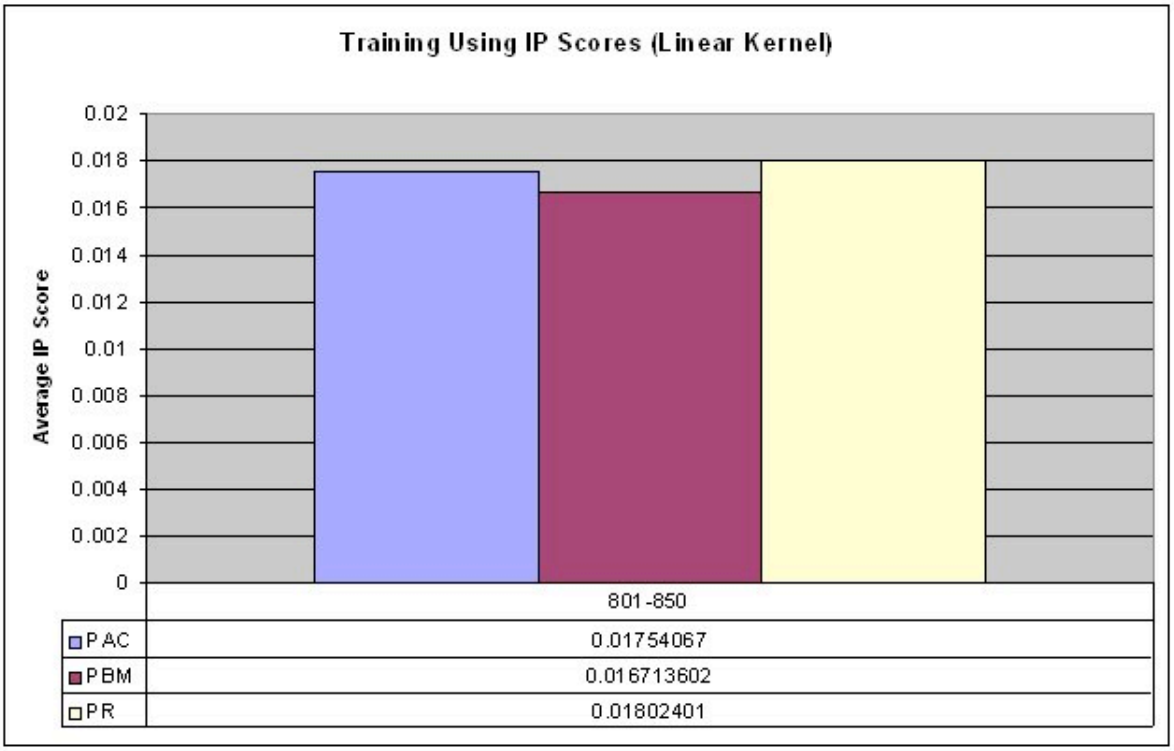


Figure 6.50: Training SVM using IP scores with a linear kernel.

- PR: c=128, j=1.

As can be seen from Figure 6.50, the performance of the different approaches differ somewhat from the previous approach of tuning parameters based on the classification accuracy. In particular the PR approach performs quite well in comparison to both the PBM and the PAC approaches.

At this stage we can see that unlike when tuning the PBM and PAC measures using the classification task, where we achieve near perfect classification accuracy, here there is room for improvement. We now choose to introduce the RBF kernel into the SVM, so that the input vectors may be mapped into a higher dimensional space – where separation between positive and negative examples may be more successful. For this we used the same training and testing files for each approach as before (PAC, PBM and PR) and tuned the parameters c, j and g using the IP score as before. Figure 6.51 shows a comparison between the three approaches using the linear kernel (as before) and now using a RBF kernel.

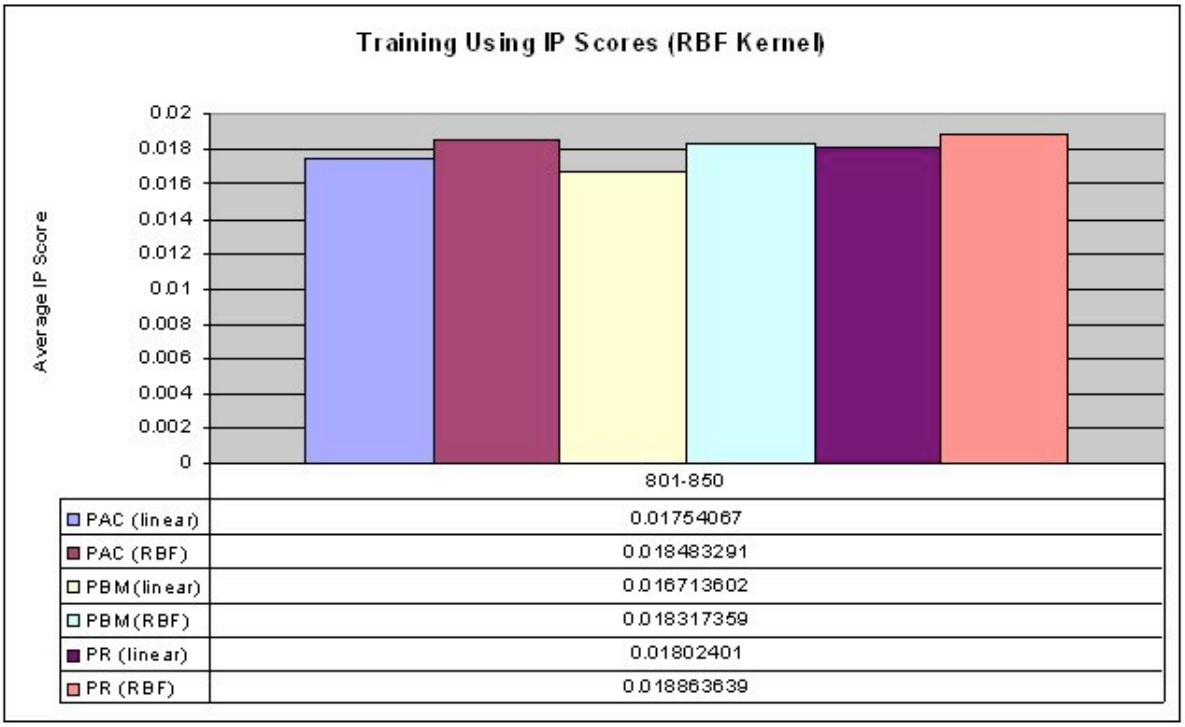


Figure 6.51: Training SVM using IP scores with a RBF kernel.

The parameter values for these models are as follows:

- PAC: c=0.001, j=2.
- PBM: c=0.001, j=1.
- PR: c=0.5, j=8.

It is clear to see from Figure 6.51 that the RBF kernel improves the results significantly over using the linear kernel. Again the PR approach performs the best, followed by PAC and then PBM.

Analysis: it may seem unusual at first that the PR approach performs the best, as the others measures (PAC and PBM) should have more accurate negative example documents in their training and testing files. This can be seen from the results of the classification experiments where the PBM (using a linear kernel) was able to classify the testing documents with an accuracy of 99.98% and with the PAC approach not far behind with 95.01%, whereas the PR approach achieved only an accuracy of 79.52%. This demonstrates to us that perhaps our chosen documents as the

negative examples for PAC and PBM may have been extremely negative examples and that is why it was so easy to accurately classify their training examples into their correct positive and negative classes. On the other hand, because we choose random documents to represent the negative examples for the PR approach, although it was more difficult to classify these documents into their correct classes, this approach produced the best way to sort the index by.

If we imagine a 2D plane (such as that shown in Figure 6.52), on which lies a collection of documents, at the top lies the high quality documents, and at the bottom lies the low quality documents. If we then attempt to identify a set of *Positive (P)* and a set *Negative (N)* examples from this collection – doing so in such a way as we did with the PAC and the PBM approaches we would expect something similar to that shown in Figure 6.53 where the positive and negative examples are towards the two extremes of the plane. This would allow for an easy separation of the two sets, and so allow for high classification accuracy, as we experienced in our experiments with the PAC and PBM approaches. However, as the two classes are so clearly separated it becomes difficult for the SVM to classify documents that are in the middle of the plane. As we explained in Chapter 5 the SVM separation concentrates on the *support vectors* that lie close to the separating plane between the two classes (positive and negative), and as the two classes are so far apart there are less support vectors for the SVM to deal with. We believe that with a selection of negative examples that are slightly higher in quality that although its classification accuracy would decrease, its ability to accurately sort an inverted index would be increased. We believe that this is the most likely explanation for the difference between the PAC and PBM approaches – PBM is more accurate at classification, suggesting that the two classes are further apart than those chosen by the PAC approach, and the PAC approach is more effective when tuned using the IP scores.

If instead we choose the negative examples using a random approach, as in our PR approach we would end up with a situation similar to that shown in Figure 6.54. With this situation the two sets are much closer together and with a much

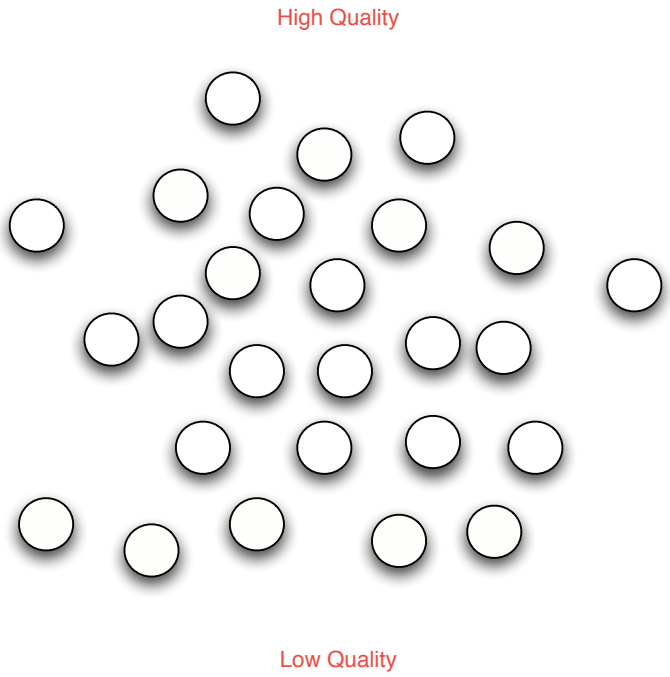


Figure 6.52: High and low quality documents.

greater chance of misclassified documents. This would obviously make the task of accurately classifying examples difficult, and this was shown to be the case. Also the reason that the PR approach performed the best may also be explained by the fact that the SVM can take into account that the negative examples may not be highly accurate, by giving higher weight to the positive examples – this is controlled by the j parameter, and for the PR approach the optimal value for this was 8 which suggests that these random negative examples were not given as much weight as the positive examples.

Although we believe that this PR approach has a distinct disadvantage, in that it is tuned very specifically to a certain set of positive and negative examples and for the (randomly selected) negative examples this is probably an untrue reflection of a correct selection of more accurate negative examples. When we evaluate the average IP scores on a set of unseen topics (801-850) we can now see that the performance of the PR approach degrades significantly in comparison to both the PAC and PBM approaches.

Figure 6.53: High and low quality documents with *positive* (P) and *negative* (N) examples.

Figure 6.55 shows what we believe to be a truer reflection of the performance of the different approaches, where they are evaluating a set of unseen topics (801-850). Now the performance of the PR approaches drops off as we expected, as we believe that it was overly tuned on a unrealistic set of negative examples, whereas the PAC and PBM approaches remain more stable.

6.7.3 Combination Analysis

We now take a look at the different types of combinations that we have experimented with and see how they compare. For this we evaluate the main approaches on a set of *unseen topics* (801-850), so that no approach has an unfair advantage.

- **Leave-one-out strategy:** from this strategy the only method that produced a score that was above the highest individual measures was the Dempster-Shafer method – this is the only methods that we will use from this strategy (referred to as LDS).

Figure 6.54: High and low quality documents with *positive* (P) and random *negative* (N) examples.

- **Pair-wise combination:** from this strategy the only method that produced a score above the highest individual measures was the Dempster-Shafer method – this is the only methods that we will use from this strategy (referred to as PDS).

- **SVM Combination:** From this we shall use the PAC measure, generated using the RBF kernel.

Figure 6.56 shows the results of the average IP scores of these three different methods. It is clear that the PAC approach is the highest performing measure, followed by PDS and then LDS. In fact the PAC method scores higher than any of the individual measures on this unseen set of topics (as can be seen from table 6.7) and is statistically significant from the highest single measure (URL length) – statistical significance of 0.03 using a paired t-test.

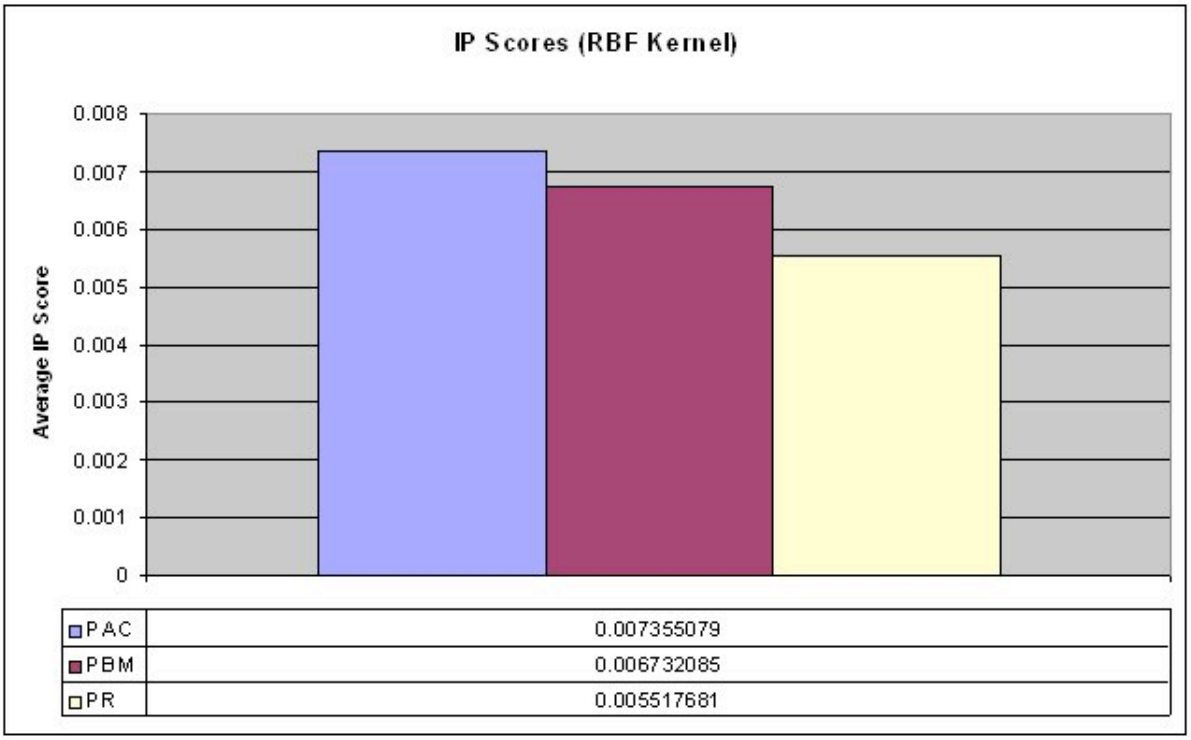


Figure 6.55: IP scores with a RBF kernel on an unseen set of topics.

Measure	**Average IP Scores**
URL length	0.00676
Indegree	0.00633
Global BM25 (QLTF)	0.00564
Access counts	0.00540
Info-to-noise ratio	0.00534
Random	0.00498
Error no.	0.00493
Warning no.	0.00482
PageRank	0.00460
Document length	0.00396

Table 6.7: Comparison of measures using average IP score (topics 801-850)

6.7.4 PageRank Re-visited

In Chapter 4 we discussed the use of the PageRank measure in order to calculate a popularity measure for the documents within a document collection, based on the linkage structure between the documents. As we mentioned previously, the calculation of PageRank is often described a "random surfer" on the Web, that

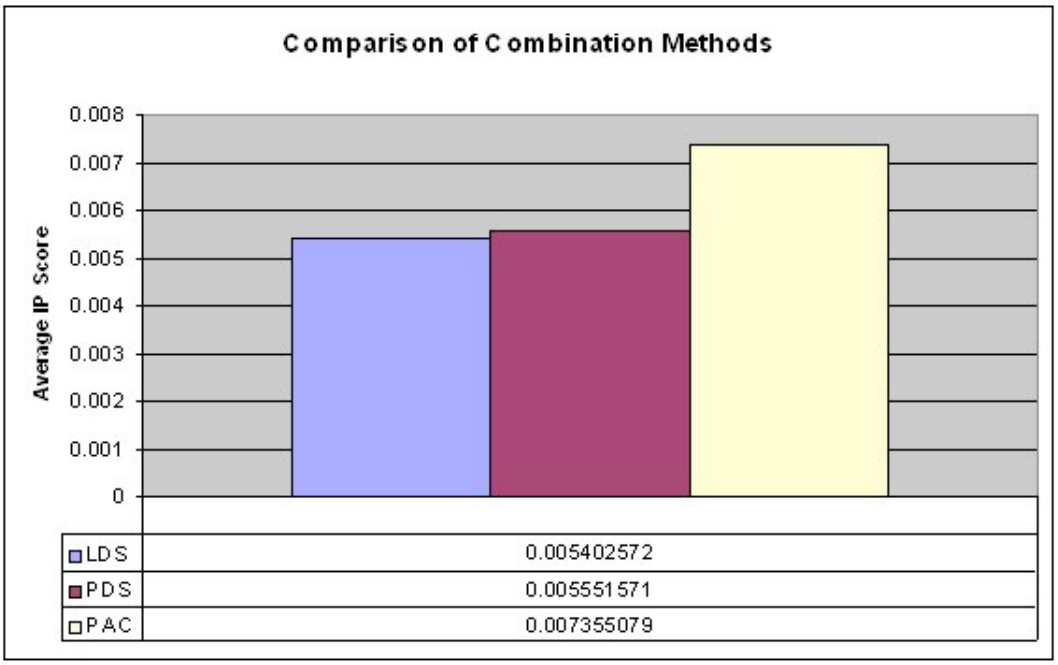


Figure 6.56: Comparison between the different combination methods.

browses from page to page, then at certain points becoming bored and jumping to an alternative page. In the typical calculation of PageRank this "random jumping" is done at random and all the documents in the collection have an equal chance of being selected. However the PageRank calculation does allow for this random jumping to be somewhat less random, and in fact we can change this so that rather than being random at all, the jumping is done to our own preferences. In this way a *personalised PageRank* may be calculated. Although rather than generating a personalised PageRank for any particular user's perference, we wish to personalise the PageRank calculation, so that is prefers to jump to higher quality documents (as estimated by our previous experiments).

For this we generate a *preference vector* which gives a preference score to each document in the collection, and as our previous experiments suggest that our SVM PAC methods produces the best results, we use this to generate our preferences. In order to do this we use the same PAC method that is tuned using the IP scores on the topics 750-800 we generate these preferences. Figure 6.57 shows the performance of this personalised PageRank, compared with the default PageRank calculation.

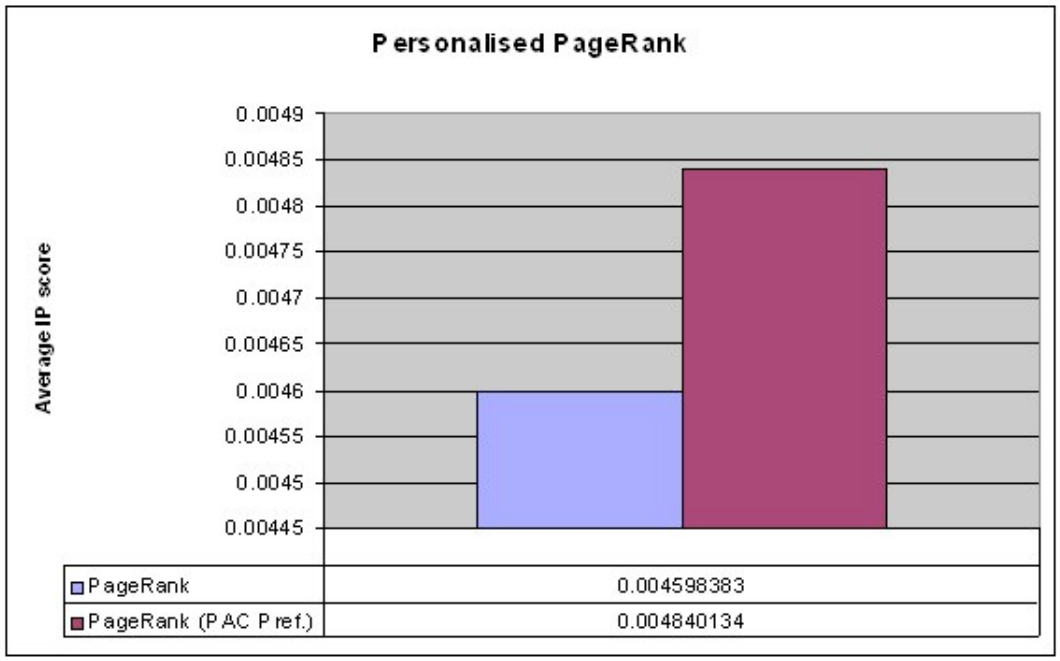


Figure 6.57: Personalised PageRank performance.

This shows that modifying the preferences of the "random surfer" so that it now prefers to jump to the documents ranked highly by the PAC measure, increases the performance of the default PageRank measure.

Also as defined within the calculation of PageRank is the *damping factor*, which assigns the probability that the random surfer will follow a link from the current page, this again allows us another means by which to modify the surfing behaviour of this random surfer. Our default calculation of PageRank, as well as our additional calculation with the *preference vector* both use a damping factor of 0.85 – i.e. there is a 85% probability of the surfer following a link on the current page. Next we decrease the value that we assign this damping factor (and so increase the probability that the surfer jumps to our preference vector). Figure 6.58 shows the performance of more personalised PageRanks with damping factors of 0.85, 0.25, 0.1 and 0.1. This shows that decreasing the damping factor (and so increasing the likelihood of jumping to our preference documents) increases the performance, however, this does not perform as highly as using the PAC measure on its own.

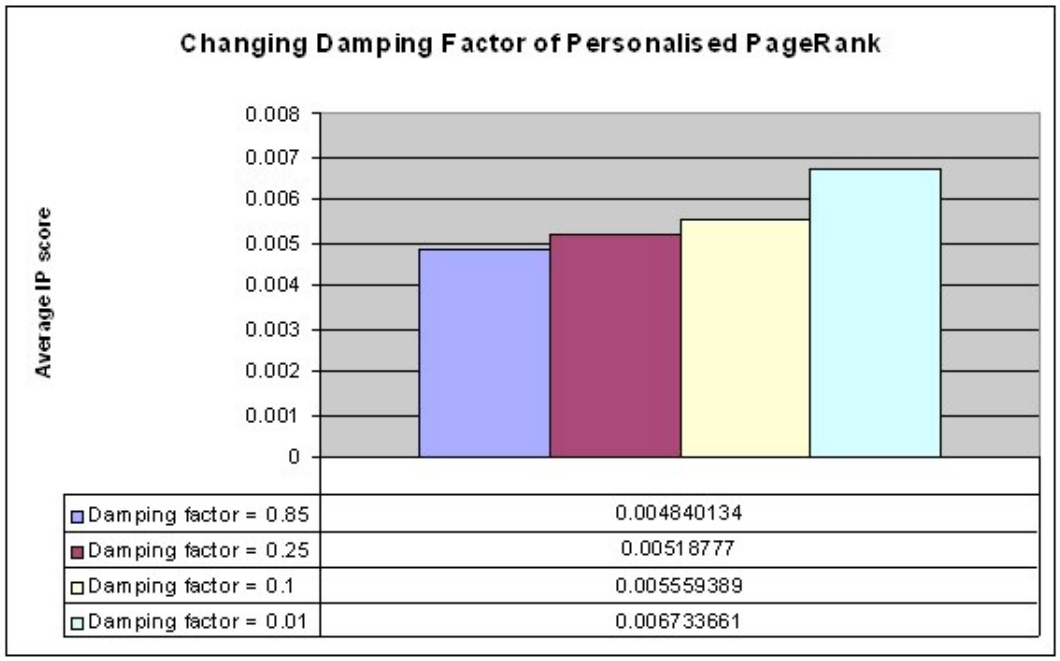


Figure 6.58: Changing damping factor of personalised PageRank.

6.7.5 Combining Static and Term-specific Scores

In our previous experiments we have concentrated on combining static query-independent measures together in order to produce a single more effective measure. Having done this we can still see that when comparing the best of our static measures (PAC, PDS, and LDS) with the best of our term-specific measure (BM25), as shown in Figure 6.59, the term-specific measure clearly outperforms the static measures. This is not very surprising, as we would expect that a term-specific measure would perform much better than a static measure alone, however, there does seem to be quite a large gap between the two types of scores (term-specific and static). In this section we will combine these two types of scores, in an attempt to provide a more effective overall measure.

6.7.5.1 Combining at Index Creation Time

In Chapter 4 we described the process for combining a term-specific measure with a static measure. This is essentially the same as combining two static measure together, except that because the term-specific measure generates different scores

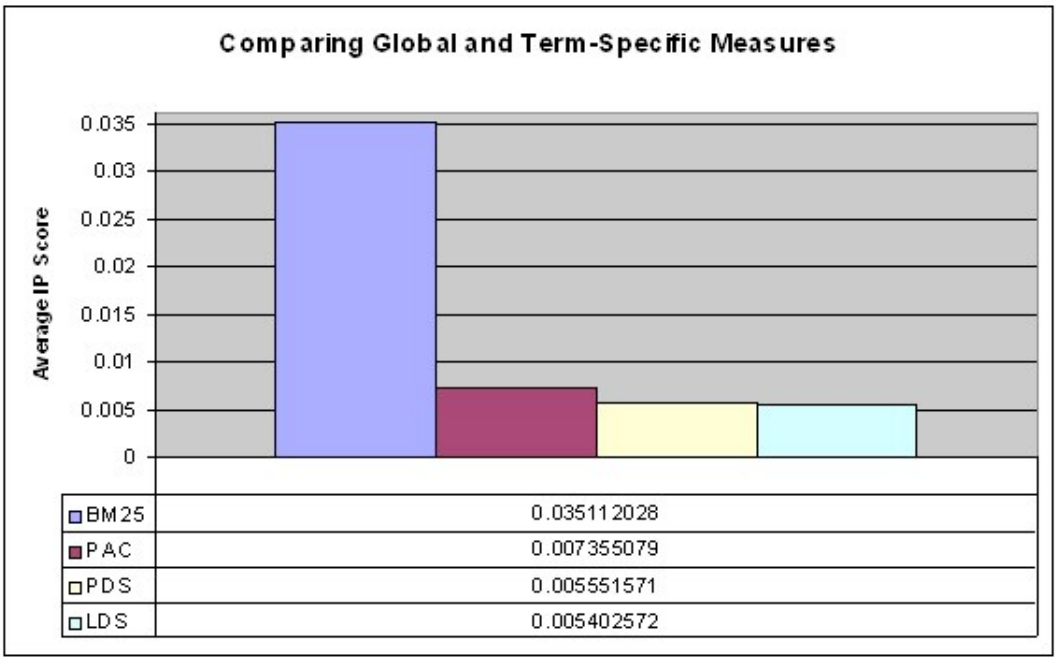


Figure 6.59: Comparing static and term-specific measures.

for documents based on the term being processed, the combination must also occur as each term is processed during the index creation process.

In order to combine these measures, like combining the static measure, we can choose from a number of combination methods such as combSUM and linear combination, however, due to the success of the Dempster-Shafer method in combining the static measures we choose to use only this method to combine the term-specfic and static measures. We do this not only because of its prior success, but also as a new inverted index must be created for each different combination that we investigate, and as this is a very time consuming process (as well as consuming a large amount of hard disk space) we feel we must only persist with the most likely approach to produce a superior performing combination.

We combined each of the static measures PAC, PDS and LDS with BM25 for each postings list in the inverted index using the Dempster-Shafer approach, where we varied the weights given to each measure from 0.1 to 0.9 (in increments of 0.1), in a similar way as we did in the pair-wise combination strategy (in section 6.7.1). This resulted in a the creation of 9 different indexes for each of the static measures

that we combined with BM25 (27 in all for the 3 measures we used). For each of these we calculated the average IP scores for the topics 801-850 and selected the highest performing index for each of the static measures that was being combined, these results are shown in Figure 6.60.

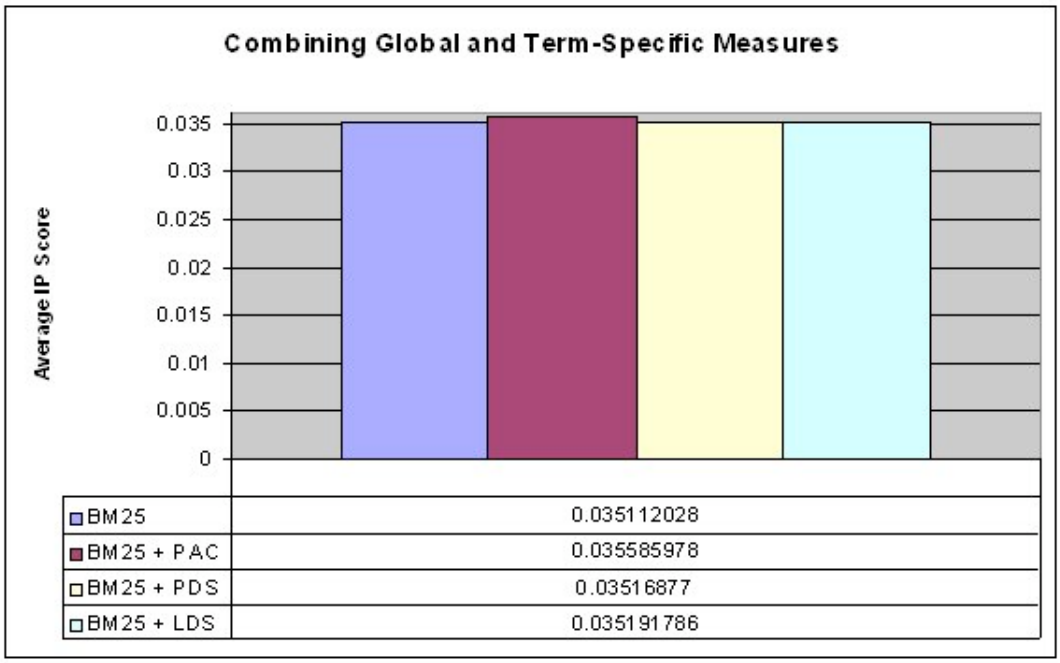


Figure 6.60: Combining static and term-specific measures.

Table 6.8 shows the optimal weights (between the static measures and BM25) that were used to produce the results in Figure 6.60. This shows that overall the PAC measure contributed the most to the combination, with a weight of 0.7, while BM25 only contributed 0.3, while the other measures contributed much less to their optimal combinations.

Measure 1	**Weight 1**	**Measure 2**	**Weight 2**
BM25	0.3	PAC	0.7
BM25	0.8	PDS	0.2
BM25	0.9	LDS	0.1

Table 6.8: Optimal weights used in the combinations

As shown in Figure 6.60, the combination of the static measures all gain an increase in performance over BM25 alone, however, only the combination of BM25

with PAC and PDS measures provide results that are statistically significant from that of BM25 (as shown in Table 6.9) using a paired t-test (with the BM25 and PAC combination being the only combination that is highly significantly different from BM25). This can probably be attributed to the amount of weight that was given to the PAC measure in the combination, when compared to the others (PDS and LDS), however, as more weight was given to these measure they degraded the performance of the index (in terms of average IP).

Measure	**Probability of not being significant**
PAC + BM25	0.0028
PDS + BM25	0.0243
LDS + BM25	0.1450

Table 6.9: Measuring statistical difference between combinations and BM25

Having managed to produce a method that can outperform the BM25 measure, using the average IP, next we now revert back to using more conventional measures such as MAP and P10 to see if the use of the IP measure that we adopted will also perform well with these other measures. Ultimately we wanted to produce a measure that would perform well in terms of measures such as MAP and P!0, having adopted the IP measure as a means to evaluate the effectiveness of the sorting within the postings lists in the inverted index.

6.7.5.2 Conventional IR Evaluation

For this we wish to evaluate our inverted index created using our optimal combination of BM25 with the PAC measure, and evaluate this using MAP and P10. This requires us to revert back to our previous approach of ranking the documents using BM25, while limiting the number of postings that are evaluated for each query-term, using the *maximum postings size cut-off* approach. For this experiment we will use the following cut-off sizes: 10,000, 50,000, 100,000, and 200,000, 300,000 and 400,000. As we discussed previously we feel that at the cut-off point of 200,000

(postings per query-term) is a reasonable cut-off point that eliminates a large number of postings from being processed, while generally allowing enough to obtain a satisfactory level of accuracy. Figure 6.61 shows that by selecting the 200,000 cut-off point the system processes less than 15% of all the postings for each of the query-terms. This will save our system from examining a large number of postings, which will lead to a faster query-time as well as reducing the necessary system resources, such as memory and CPU clock cycles. The goal is then to limit the trade-off between system accuracy and processing more postings that is made, as we process only a small number (e.g. 15%) of the overall postings for each query term.

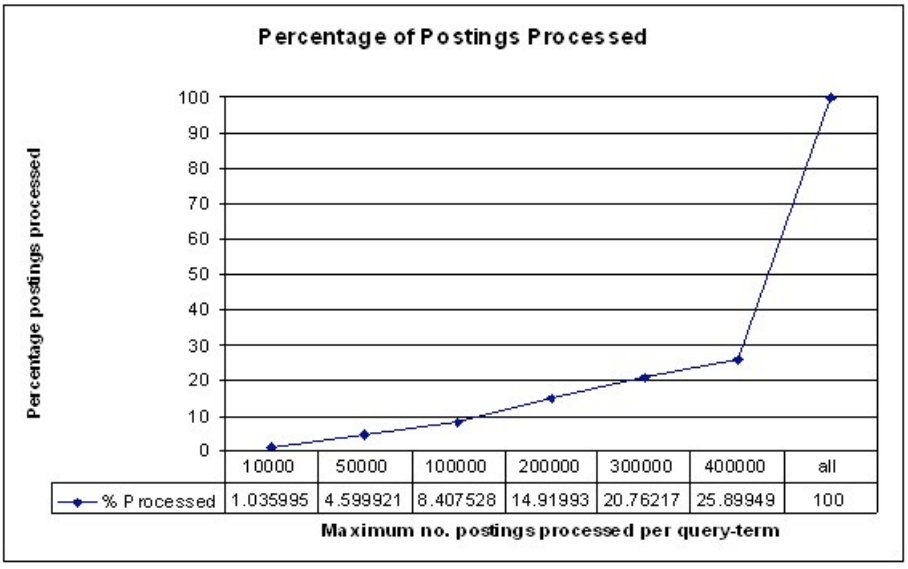


Figure 6.61: Percentage of postings processed at each cut-off point on topics 801-850

Figures 6.62 and 6.63 show the performance of the index created by combining BM25 with the PAC measure, when compared with the standalone BM25 measure. Here we can see that although the combined index does prove to be more effective, both for MAP and P10 at the 200,000 cut-off point, overall there is little to separate between the two indexes (particularly at the earlier cut-off points). For MAP the two indexes perform quite similarly, while for P10 the BM25 index performs above the combined index at certain earlier cut-off points.

However we notice that there may be an unfair bias towards BM25 with this

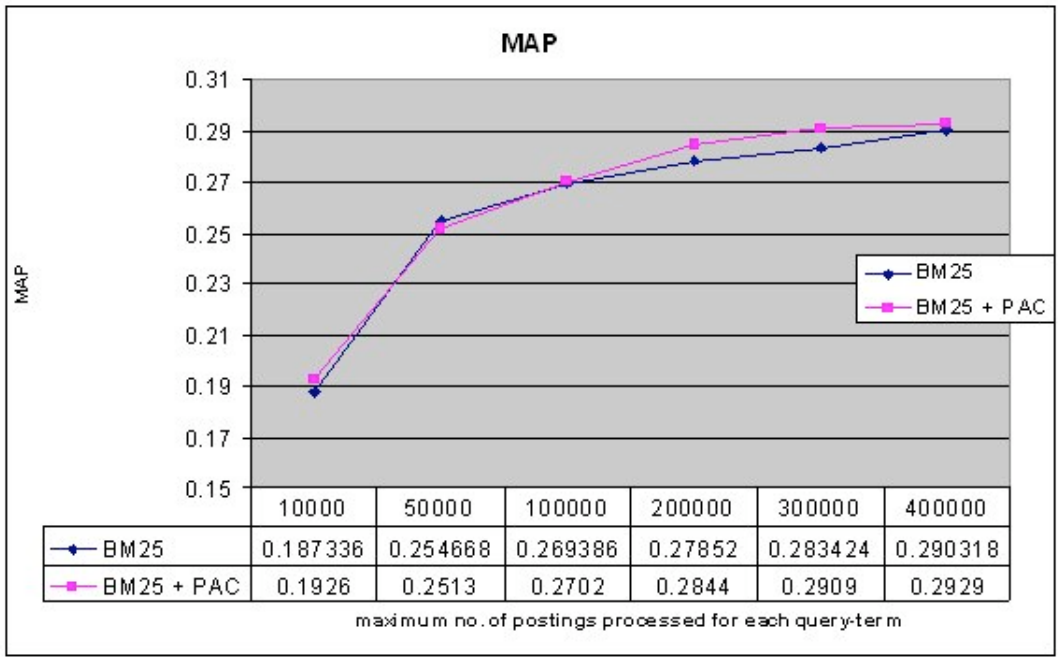


Figure 6.62: Evaluating BM25 + PAC index using MAP.

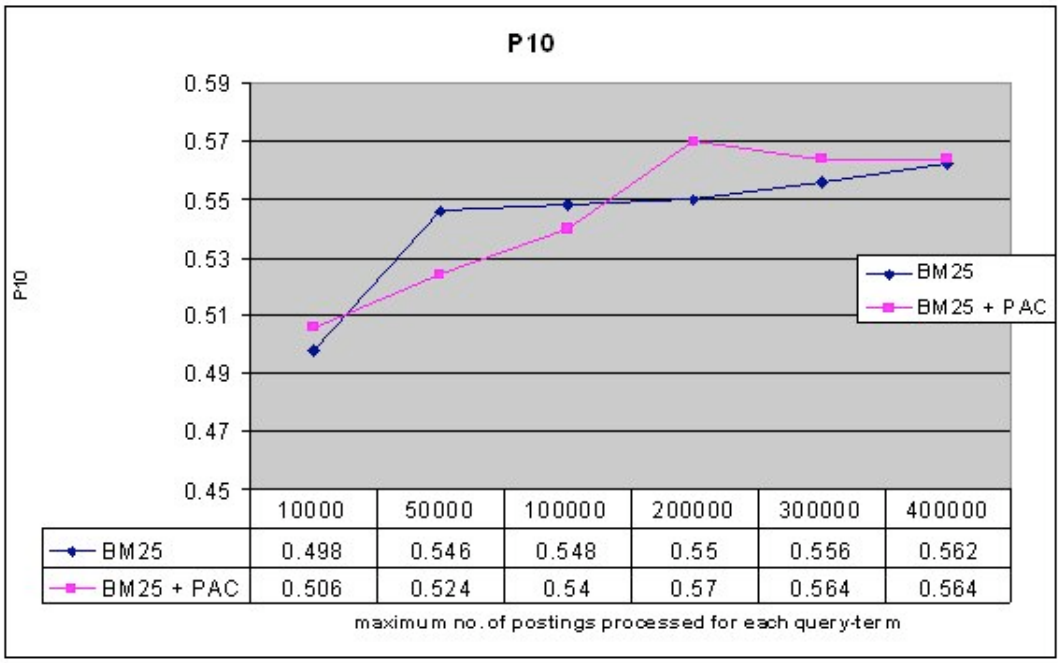


Figure 6.63: Evaluating BM25 + PAC index using P10.

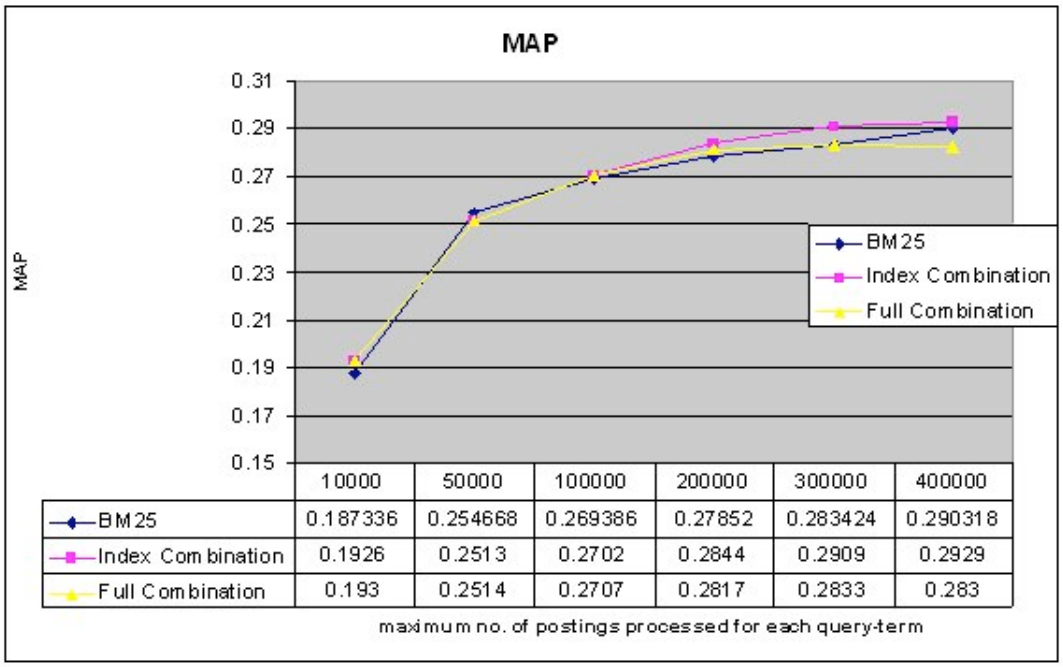
evaluation: in a similar way to the way in which we believed that the standalone static measures were perhaps being unfairly evaluated at retrieval time, by the use of the BM25 measure in retrieval. This is because in the creation of the index we

have re-ranked the postings using a combination of the BM25 and PAC measures, now at retrieval we ranking documents only BM25 and so we are diluting the effect of the PAC measure, as we increase the size of the cut-off. In order to fully evaluate the contribution of the both measures at retrieval time we believe that the measures must also be combined at retrieval time.

6.7.5.3 Combining at Retrieval Time

In order to combine the same static measure (PAC) with the BM25 scores at retrieval, this done is the a very similar way as to the way it was done at the index creation stage. Although at the index creation stage the BM25 scores were calculated in isolation for each term, at retrieval time BM25 combines the scores for each of the query terms. So rather than combining after each term's scores are calculated, the combination is carried out after all the BM25 calculations have been made. Here (again) we use the Dempster-Shafer method for combination, and again we use a number of weights for each of the elements being combined: choosing two weights so that they sum to 1, starting at 0 and incrementing by 0.05 (giving 20 combinations in all). In choosing an optimal combination from these we can choose to make a trade-off between the performance of MAP and P!0, in this case we choose in favour of the P10 performance: we do this because of the nature of our type of retrieval, we are trying to speed up the query-time while presenting the best top 10 results that we can, as for a Web search scenario, most users do not look at the results past this point. Figures 6.64 and 6.65 present the MAP and P10 performance figures for the previous approach, where the combination occurred only in the index (*index combination*) and now where the combination is done at both the index and retrieval stages (*full combination*).

Here we can see that although there is relatively little change between the scores for MAP, for P10 there is a clear improvement, and yet another gain is made over the standalone BM25 index: for our 200,000 cut-off point there is a 8% increase in P10 (the difference between the P10 values is also statistically significant with a



	10000	50000	100000	200000	300000	400000
BM25	0.187336	0.254668	0.269386	0.27852	0.283424	0.290318
Index Combination	0.1926	0.2513	0.2702	0.2844	0.2909	0.2929
Full Combination	0.193	0.2514	0.2707	0.2817	0.2833	0.283

Figure 6.64: Evaluating BM25 + PAC at both the index and retrieval stages.

value of 0.014). We feel that this shows a truer reflection of the benefit that can be obtained from combining our static measure with BM25 .

6.8 Examining Performance Trade-offs

We now consider the performance of the conventional IR system, where all postings are examined and then ranked using BM25, compared with a sorted index created using our discussed approach, which combines the SVM PAC measure with BM25 at both index creation and retrieval time. Tables 6.10 and 6.11 present the performance figures for MAP and P10 respectively for the convention index (ranking using BM25) and our sorted index using the 200,000 posting cut-off point (which amounts to 15% of the total postings for all query-terms).

This equates to a 6.9% loss in MAP, however, accomplishes an increase of 6.1% in the P10 performance – even though the system is processing much less postings. Again we feel that the trade-off in MAP is acceptable for a search system whose users are primarily concerned with the accuracy of top 10 results, as well as the

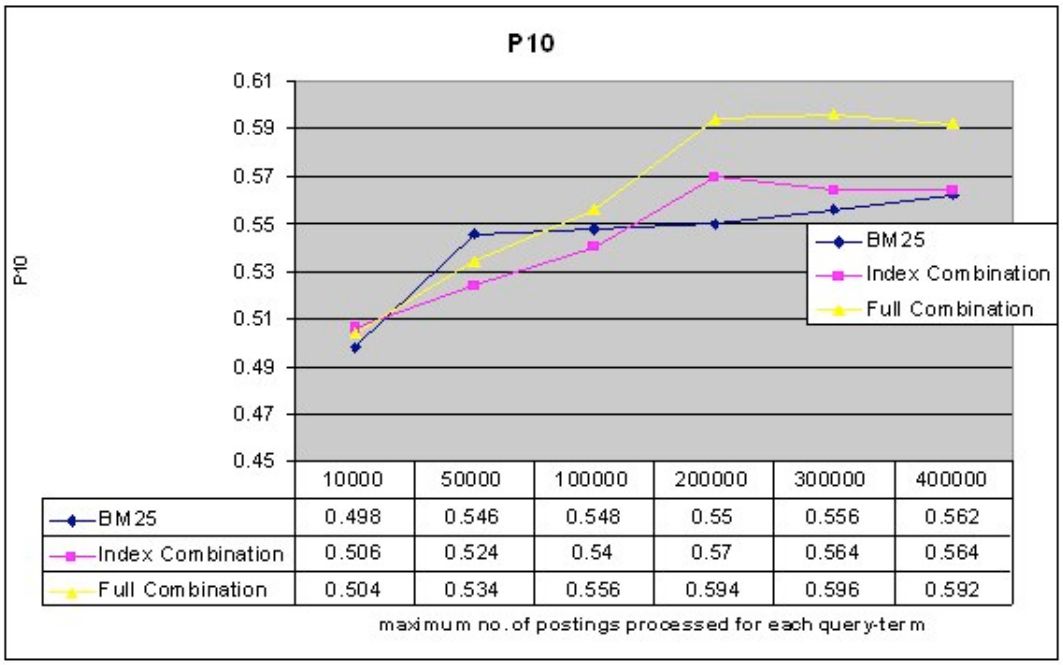


Figure 6.65: Evaluating BM25 + PAC at both the index and retrieval stages.

Index Type	**MAP Score**
Conventional Index	0.3044
Sorted Index (only processing 15%)	0.2817

Table 6.10: Examining the trade-off in MAP between a conventional index and a ("pruned") sorted index.

responsiveness of the search system, and we believe that our approach provides an ideal candidate for this. Also a certain drop in MAP performance is to be expected as we are processing much less documents and inevitably a small number of relevant documents will be missed, and so this drop in MAP is most likely due to a drop-off in recall. For a Web search scenario, where the typical user does not look past the top ten results this drop in recall will not be noticed.

As we are primarily interesting in judging the effectiveness of our P10 performance for our fully combined sorted index, we look to compare this with the results of another search system. For this we look to the results produced from the publicly available *Zettair* search engine which was used by the RMIT group in the TREC terabyte conference Garcia et al. (2006). Their *zetabm* run from their TREC 2006

Index Type	P10 Score
Conventional Index	0.56
Sorted Index (only processing 15%)	0.594

Table 6.11: Examining the trade-off in P10 between a conventional index and a ("pruned") sorted index.

submission (details in Garcia et al. (2006)), uses the Okapi BM25 ranking and so we assume that this should be comparable with our own baseline BM25 performance. Figure 6.66 compares our own baseline (using BM25 ranking, as shown in Table 6.11), as well as our sorted index at the 200,000 cut-off point, with the Zettair (*zetabm*) run – as we can see from this our baseline run achieves an improvement of 12.45% over the *zetabm* score, while our sorted index (at 200,000 cut-off) gains an increase of 19.28%.

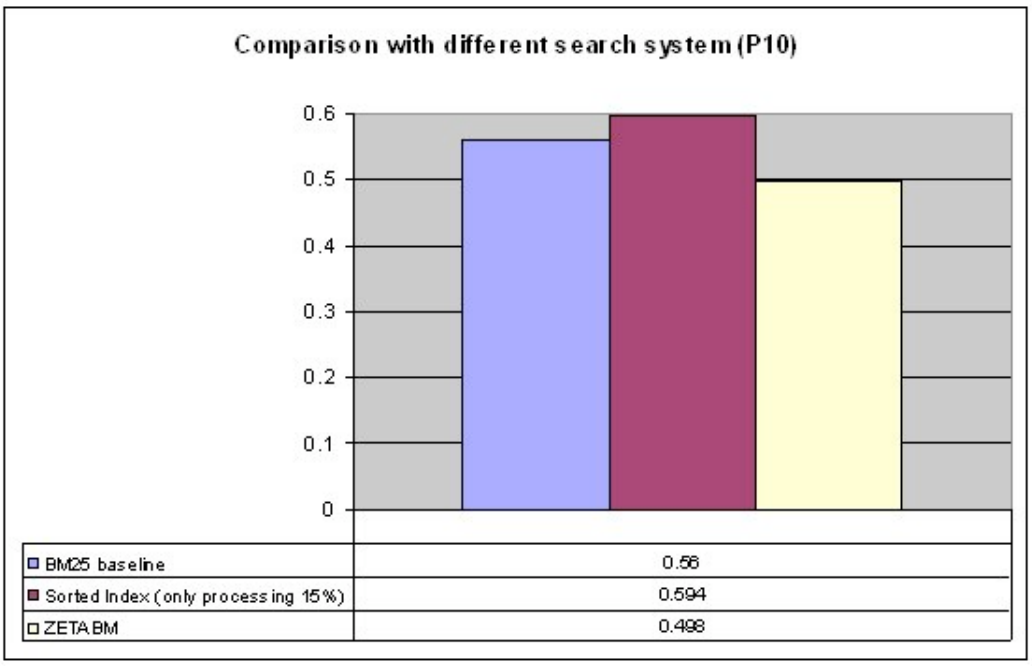


Figure 6.66: Comparing performance with a different search system (P10).

This gives a more objective view of the performance of the system, and shows that our system's BM25 ranking provides a relatively strong baseline. Overall this shows us that our sorted index approach allows us to effectively eliminating large portions of the inverted index without degrading the system accuracy and in fact

improve upon a standard Okapi BM25 ranking which evaluates all postings.

6.9 Impact of the Sorted Index

In order to investigate the impact that the sorted index has on improving the systems performance, we decided to combine our best static measure (PAC) with our baseline BM25 search (which processes all the postings) at retrieval, so that we could see if that the system can achieve the same effectiveness (in terms of P10), even though it will be much less efficient (as it will process all the postings for each query-term). This will show us if the use of a sorted index contributes to the effectiveness of the system as well as to its efficiency.

For this we combined the BM25 scores with our PAC measure using the Dempster-Shafer approach (in the same way as we combined our sorted index's results with PAC previously). Again we selected the top performing combination, using a number of weights for each combination source and Figure 6.67 shows the P10 performance for the baseline BM25 ranking (using all postings), as well this same baseline combined with our PAC measure at retrieval time, and this is compared with our fully combined sorted index (using PAC and BM25).

This shows that even though the incorporation of the static measure improves the results over using BM25 alone, it is still not able to achieve the same level of performance as the sorted index which uses the same measure. This highlights the usefulness of the sorted index, as a conventional index is unable to achieve the same gains by incorporating the same static measure at retrieval time.

We believe that this is because the sorted index is able to filter out the lower quality documents that a term-weighting ranking function such as BM25 does not account for and so by choosing to process less postings (using the sorted index) the postings that are not selected are less likely to be relevant and so the retrieval accuracy is improved along with the efficiency of the search.

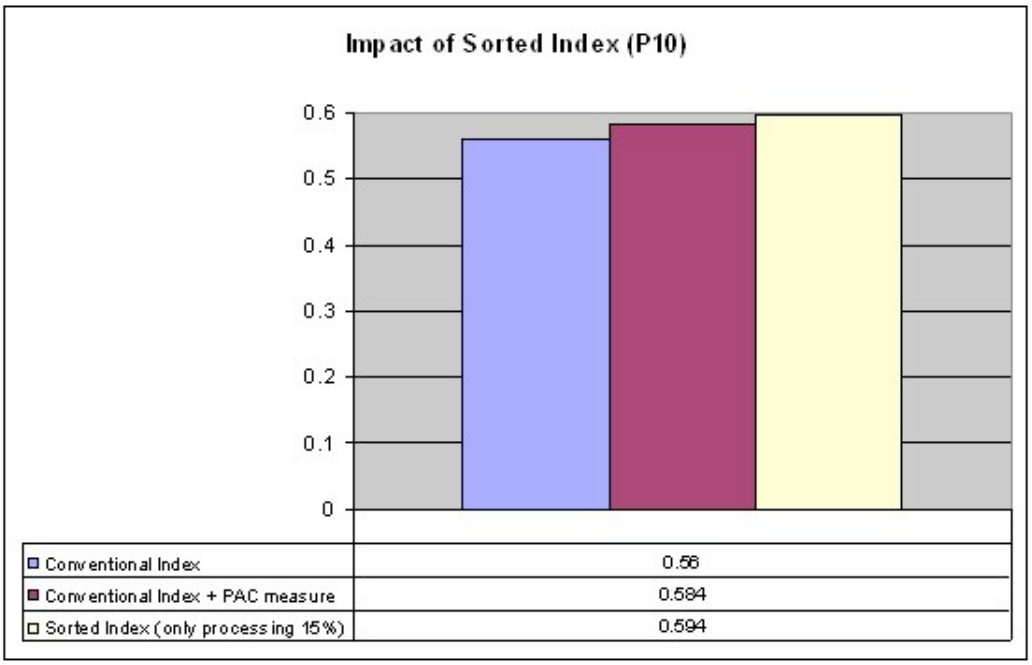


Figure 6.67: Impact of using a sorted index (P10).

6.10 Summary

Firstly in this chapter we evaluated a number of early-termination schemes that allow for a number of postings to be eliminated from each of the postings list. From this we found that the *maximum postings size* approach allowed to us the best trade-off between the number of postings processed and retrieval accuracy.

We then experimented with many different query-independent measures for the task of effectively sorting an inverted index. First of all we evaluated these in a standalone fashion, as if being used in a search system, and using a BM25 ranking of the documents and evaluated the performance using MAP and P10. From these experiments we found that the term-specific measures, such as BM25 and with document term frequency (TF) were the highest performing

We later re-assessed the way in which we evaluated the performance of these measures and eventually introduced the IP measure, which we felt was a more direct measurement of the sorting which the postings lists. Again using this approach we found that the term-specific measure were the highest performing (with a greater

increase over the static measures than before). We also found that using this IP measure gave us significant savings in a number of areas, and made it feasible for us to evaluate a large number of combinations.

In section 6.7 we experimented with a number of different combination strategies: leave-one-out, pair-wise, as well as combination using support vector machines (SVMs). Firstly we concentrated on combining the static measures together to find a number of best performing static measures. From these experiments we found that the SVMs provided the most effective combination results. Next we experimented with combining the best performing (combined) static measures with the best term-specific measure (BM25) in order to improve upon the term-specific scores. We found that we were also able to increase the term-specific scores by combining these with our combined static measures. Also in section 6.7 then sorted an inverted index using our best performing measures (PAC – as indicated by the IP scores) and then evaluated this using MAP and P10. We found that this could give us a slight increase in MAP as well as P10, but when we combined our PAC measure with BM25 at retrieval-time as well, this gave a further increase to performance to P10, while decreasing MAP slightly.

Having produced what we believed to be an effective sorted index we then took a look at the trade-offs that were being made by using this sorted index and eliminating a large number of postings, compared against our own baseline system (where no postings are eliminated from the query-terms), as well as comparing against a different search system. Overall we found that by using our sorted index approach and eliminating postings we could actually gain an increase in P10 over our own search engine's baseline BM25 search (which processed all postings), as well as that of the publically available Zettair search engine (using a similar BM25 ranking on all postings).

Finally we tried to isolate the impact that the use of the sorted index has on the effectiveness of system. For this we combined the results of our baseline search (which processed all postings for each query-term) with our best static measure

(PAC) at retrieval. This was to see if the static measure was used with a conventional index and just combined at retrieval time would be as effective as combining it with a sorted index (even though it would process 85% more postings and would therefore be much less efficient). Here we found that the the conventional index could not gain the same level of effectiveness as the sorted index, again highlighting the usefulness of the sorted index at improving both the system's effectiveness and efficiency.

Chapter 7

Conclusions

In this chapter we draw some conclusions from our work, and we then suggest some extensions and future directions.

7.1 Conclusions

Here we give our conclusions from our main topics of investigation, and then draw some overall conclusions from our work.

Early Terminiation

Firstly we found that sorting an inverted index by an effective measure (such as BM25) and choosing to eliminate a large number of postings using various strategies, and in particular the *maximum postings size* approach, allowed only a small drop-off query effectiveness as measured by MAP and P10. This provided us with a means by which our system could process less than 15% of the total number of postings that were available of all the query-terms and still provide reasonable results to the end user – therefore increasing the *efficiency*, with a minimal decrease in *effectiveness*.

From our initial experiments we were quite impressed with this level of performance and considered this a relatively strong performing baseline. Although our main goal was then to generate an effective static measure and then determine if

this could be incorporated with our baseline sorting to further improve upon these initial results.

Term-Specific Versus Static Measures

Our next area to investigate was the effectiveness of using different query-independent measures to sort an inverted index. We found that the term-specific measures such as BM25 far outperformed the static measures. This was not particularly surprising to us, as these term-specific measures provide a specific sorting for each postings list which is more accurate than sorting all the postings by the same static sorting. Our original intention was to combine these static measure together in such a way, so that they provided a richer representation of the documents and then combine these with the term-specific measure to improve upon their performance also.

Evaluation

Having re-assessed the way in which we were originally evaluating the effectiveness of the each sorted index we then proposed the use of an *Index Precision* (IP) approach, in order to calculate the effectiveness of using different query-independent measures to sort an inverted index. Not only did this save us a large amount of computing time, and hard-disk space, but this also made it feasible to evaluate the large number of variants of different measures (that would be generated as a result of our combination experiments).

We also believe that this is a more accurate way of accessing the performance of a sorted index, compared with the previous approach of analysing the performance of a BM25 ranking at various cut-off points. This approach also eliminates the diluting effect that can be caused by assessing the ranking after applying another query-time ranking such as BM25. This method assesses the position of the documents with the postings lists themselves and so should produce a more accurate assessment of the sorting performance, than trying to assess this after an additional ranking has been applied.

Combination

We next found that of the combination methods that we experimented with, our SVM approach produced the best results. Although we found that the way in which the positive and negative example documents can have a significant effect on the performance on the classification, because of this we believe that these results could be further improved with further experimentation into the way representative sets of *positive* and *negative* documents are selected from the collection. Perhaps a manual intervention in selecting high and low quality documents from a collection (while time-consuming) would allow an SVM to more accurately classify documents.

In general we found that the combination of different measures, with traditional combination methods such as linear combination (which required weights) was difficult and time consuming, while combining based on the documents' rankings proved to be quite ineffective for our specific task. Even when combining less than ten sources together it can be time-consuming to try all possible combinations (for a weighted combination), while also varying the weights given to each combination. This would also grow if we were to add additional sources into this combination. Rather than trying all possible combinations we experimented with two different combination strategies, which we felt could gain the most from the sources that we were combining (while remaining computationally feasible within a reasonable timeframe). Using these combination methods we found that we could only achieve minor improvements over the highest performing single static measure and overall these lagged behind the more effective combination generated by the SVM combination approach.

Our experiments also found our best combined static measures could also be combined with our best term-based measure (BM25) to provide an increase in effectiveness in terms of IP. We then wanted to re-evaluate the effectiveness of the sorted index using the conventional MAP and P10 evaluations, and here we found that to gain maximum performance from the sorted index our proposed static mea-

sure should be combined at both indexing-time as well as retrieval-time with our term-based measure. Using this approach the system was able to gain a significant increase in P10 performance, even though it was only processing 15% of all available postings for the terms in the queries (compared with the system when it processed all postings for each query-term).

Personalising PageRank

We experimented again with the use of PageRank as a means of sorting the inverted index, this time by specifying the browsing preferences of the *random surfer*, as well as modifying the likelihood that this random surfer is to jump to a different page (by modifying the *damping factor*). Here we found that by changing the preference of the random surfer to the same preferences of our best performing static query-independent measure, that this increased the performance. Also by increasing the liklihood that the surfer would jump to these "preferred" pages, a further increase in performance could be observed, however, this performance was still below that of using the static measure on its own to sort the index.

Hypothesis Re-visited

Our original hypothesis was that if we could provide an effective static measure that could promote the higher quality documents within the collection, while demoting the documents of lower quality, then we could use this measure to successfully eliminate a number of documents from consideration at retrieval time, which would allow for more efficient retrieval, but also for a more effective search – as the documents that would be eliminated should be unlikely to be relevant.

The way in which we eliminated these documents at retrieval time was with the use of a sorted index, combined with the use of early termination applied uniformly to each of the postings lists of the query terms. We found that sorting the postings lists by an effective term-specific measure such as BM25 allowed only a small drop-off in effectiveness while only processing a small percentage of the overall postings.

We then investigated the use of static measures, in order to promote higher quality documents from the entire collection. We found that by combining these static measures together and then combining these with BM25 our system could achieve gains in both effectiveness (in terms of P10 performance), as well as efficiency (in terms of number of postings processed), over both our own search engine's BM25 search, as well as that of the Zettair search engine using a similar BM25 ranking (even though these were processing all available postings for each query-term).

We believe that these results validate our original hypothesis as the use of a term-weighting approach such as BM25 can only limit the drop-off in system effectiveness, while with the integration of an successful static measure the system is able to process less documents, while at the same time improve the accuracy of the results.

The fact that the system can increase either the effectiveness or the efficiency is not a particularly impressive feat, however, to increase both is much more difficult to achieve. Our findings from these experiments illustrate to us that it is possible to gain an increase in both the effectiveness and the efficiency of a search system, with the incorporation of a sorted index, which is based on the use of an effective static measure for identifying high quality documents in a query-independent manner. Although we have proved that our current approach provides effective results, we believe that there is scope for the further refinement of this work, as well as extensions to this field of research.

7.2 Extensions and Future Work

We believe the research in this thesis allows potential for future work, which we describe in this section.

Index Pruning

When using a pre-defined cut-off point with a sorted index (with the *maximum*

postings size approach for example), this is in effect ignoring the postings that lie past the specified cut-off point, and therefore the postings past this point may be pruned, while still retaining the same performance as the sorted index using the same cut-off point. The essential difference with this approach (compared with the previously discussed sorted index) is that rather than storing all the postings and then at query-time choosing a specified cut-off point, the cut-off point is now chosen at indexing time. The benefit of this approach is that a substantial amount of disk-space can be saved as a large number of postings are not saved to disk. However, this does have the disadvantage that the cut-off point is now pre-defined.

A further extension that is allowed by using this pruned index is that the index can then be re-sorted by the document identifiers so that conventional compression techniques (discussed in Chapter 2) may be used on the index. This access-pruned approach was proposed for an access count ordered index by Garcia et al. (2004), and was shown to have the dual benefits of decreasing storage space, as well as increasing query throughput.

Dynamic Cut-off Selection

One of the advantages of using a sorted index (without pruning), is that it allows the cut-off point for termination to be changed. This may be of benefit when answering potentially difficult queries, where there are few relevant documents, and so using a larger (or no) cut-off point may be of benefit. This would require us to identify the potential difficulty of a query and then dynamically choose the number of postings to process for each query term. One possible way of doing this would be to assume that the length of the query indicates how *vague* or *specific* is – with a query that is more specific perhaps requiring more postings to be processed in order ensure that sufficient accuracy is achieved with the returned results. Conversely for a vague query that may have a large number of relevant documents within the collection, it may be possible to process less postings for each query-term, and still retain high accuracy with the top 10 documents.

Also the facility for the user to specify the amount of information that is to be processed in response to their query may be useful in certain circumstances. For example, for certain searches it may be of vital importance that all available documents are returned (i.e. high recall) and as we have previously discussed the use of a sorted index with a relatively small cut-off point would not be ideal for this case and so the user may be willing to wait longer for a search to complete so that higher recall is achieved.

Different Features

Although we clearly found benefit with the incorporation of our chosen query-independent measures, we do not claim that this was an exhaustive list of the best measures that are available. Other query-independent measures, such as click-through data (which we described in Chapter 4 but did not include in our experiments due to the difficulty in acquiring such data), may be another good candidate for further investigation. There may be many other indicators of document quality that we may not have even considered, and the inclusion of many other such measures may bring about a further increase in performance.

Positive and Negative Examples

In order to combine measures using the SVM approach we selected both positive and negative example documents (i.e. high and low quality examples) to train and test the SVM model. We believe that with greater experimentation with the way in which we select these positive and negative examples we could improve the performance of this SVM approach.

Perhaps rather than using binary relevance judgments (as currently used by TREC), where documents are judged as *relevant* or *not*, a scaled ranking of relevance might give more discrimination between positive examples. Also manual intervention might be useful in order to provide more accurate positive and negative examples.

Apply on Larger Collections

Although we carried out our experiments on quite a large collection of documents, we believe that even greater gains could be made on larger collections, such as the Web, which we believe contains larger amounts of low-quality documents. This may also gain a benefit from the inclusion of a spam detection measure, which would demote documents that are likely to contain spam.

Updating the Index

One of the major drawbacks that we see with the use of a sorted index (as currently implemented) is its ability to add new documents to the index, without the need to re-sort the index. For the conventional index, which is sorted in order of the document identifiers, a new document can be added to the end of the relevant postings lists, however, when using a sorted index for each of the terms in the new document, these terms' postings lists will have to be re-sorted. Perhaps further research into this area could provide a more effective way of allowing updates to a sorted index.

Experiment with Different Types of Data

Although we applied our specific approach to the domain of text retrieval, we believe that a similar approach may be applied to different types of data collections.

For instance, in a video retrieval system which deals with the retrieval of large amounts of video that is continuously captured from different sources, there would inevitably be large amounts of data in the system that may not be of much relevance to an end user. Perhaps a system that can identify events that are likely to be of interest, for example in CCTV footage this may be the appearance of a person, which would be given a higher weighting than several minutes (or hours) of video that contain no significant activity and so would be unlikely to be of interest to a user. In this system the video segments could be stored in such a way that these more significant events are stored first and potentially allow for more efficient retrieval on

this type of data, with more effective results.

Due to the development of ubiquitous, wearable, multimedia devices, researchers have started work on devices that passively capture data, so that people can automatically capture and record their daily lives (Lamming and Flynn, 1994; Gemmell et al., 2002). This data, which is also known as *life logs*, can grow extremely quickly and this causes new problems to the way in which data should be stored and retrieved. Some of the research work in this area is concerned with the extraction of *landmark* events from this data so that significant events can be found (Blighe et al., 2006). Perhaps the use of these landmark events could be used to order the way in which this data is stored so that more significant events are store first (and searched first) – in a similar way to the way in which we utilise a sorted index for text retrieval.

Similarly, if information is being continually captured from a wireless sensor network, much of this data may not be significantly different from a lot of data that has already been captured and so a method for identifying potentially interesting events may also be useful in this case so that the data may be filtered, in order to provide more efficient and effective retrieval on this data, for certain retrieval tasks.

Appendices

Appendix A

TREC Terabyte Ad-hoc Topics

Table A.1: TREC terabyte topics (701-801).

Topic Number	Topic Title
701	U.S. oil industry history
702	Pearl farming
703	U.S. against International Criminal Court
704	Green party political views
705	Iraq foreign debt reduction
706	Controlling type II diabetes
707	Aspirin cancer prevention
708	Decorative slate sources
709	Horse racing jockey weight
710	Prostate cancer treatments
711	Train station security measures
712	Pyramid scheme
713	Chesapeake Bay Maryland clean

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
714	License restrictions older drivers
715	Schizophrenia drugs
716	Spammer arrest sue
717	Gifted talented student programs
718	Controlling acid rain
719	Cruise ship damage sea life
720	Federal welfare reform
721	Census data applications
722	Iran terrorism
723	Executive privilege
724	Iran Contra
725	Low white blood cell count
726	Hubble telescope repairs
727	Church arson
728	whales save endangered
729	Whistle blower department of defense
730	Gastric bypass complications
731	Kurds history
732	U.S. cheese production
733	Airline overbooking
734	Recycling successes
735	Afghan women condition
736	location BSE infections
737	Enron California energy crisis

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
738	Anthrax hoaxes
739	Habitat for Humanity
740	regulate assisted living Maryland
741	Artificial Intelligence
742	hedge funds fraud protection
743	Freighter ship registration
744	Counterfeit ID punishments
745	Doomsday cults
746	Outsource job India
747	Library computer oversight
748	Nuclear reactor types
749	Puerto Rico state
750	John Edwards womens issues
751	Scrabble Players
752	Dam removal
753	bullying prevention programs
754	domestic adoption laws
755	Scottish Highland Games
756	Volcanic Activity
757	Murals
758	Embryonic stem cells
759	civil war battle reenactments
760	american muslim mosques schools
761	Problems of Hmong Immigrants

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
762	History of Physicians in America
763	Hunting deaths
764	Increase mass transit use
765	ephedra ma huang deaths
766	diamond smuggling
767	Pharmacist License requirements
768	Women in state legislatures
769	Kroll Associates Employees
770	Kyrgyzstan-United States relations
771	deformed leopard frogs
772	flag display rules
773	Pennsylvania slot machine gambling
774	Causes of Homelessness
775	Commercial candy makers
776	Magnet schools success
777	hybrid alternative fuel cars
778	golden ratio
779	Javelinas range and description
780	Arable land
781	Squirrel control and protections
782	Orange varieties seasons
783	school mercury poisoning
784	mersenne primes
785	Ivory-billed woodpecker

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
786	Yew trees
787	Sunflower Cultivation
788	Reverse mortgages
789	abandoned mine reclamation
790	women's rights in Saudi Arabia
791	Gullah geechee language culture
792	Social Security means test
793	Bagpipe Bands
794	pet therapy
795	notable cocker spaniels
796	Blue Grass Music Festival history
797	reintroduction of gray wolves
798	Massachusetts textile mills
799	Animals in Alzheimer's research
800	Ovarian Cancer Treatment
801	Kudzu Pueraria lobata
802	Volcano eruptions global temperature
803	May Day
804	ban on human cloning
805	Identity Theft Passport
806	Doctors Without Borders
807	Sugar tariff-rate quotas
808	North Korean Counterfeiting
809	wetlands wastewater treatment

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
810	timeshare resales
811	handwriting recognition
812	total knee replacement surgery
813	Atlantic Intracoastal Waterway
814	Johnstown flood
815	Coast Guard rescues
816	USAID assistance to Galapagos
817	sports stadium naming rights
818	Chaco Culture National Park
819	1890 Census
820	imported fire ants
821	Internet work-at-home scams
822	Custer's Last Stand
823	Continuing care retirement communities
824	Civil Air Patrol
825	National Guard Involvement in Iraq
826	Florida Seminole Indians
827	Hidden Markov Modeling HMM
828	secret shoppers
829	Spanish Civil War support
830	model railroads
831	Dulles Airport security
832	labor union activity
833	Iceland government

Continued on Next Page...

Table A.1 – Continued

Topic Number	Topic Title
834	Global positioning system earthquakes
835	Big Dig pork
836	illegal immigrant wages
837	Eskimo History
838	urban suburban coyotes
839	textile dyeing techniques
840	Geysers
841	camel North America
842	David McCullough
843	Pol Pot
844	segmental duplications
845	New Jersey tomato
846	heredity and obesity
847	Portugal World War II
848	radio station call letters
849	Scalable Vector Graphics
850	Mississippi River flood

Bibliography

Amento, B., Terveen, L., and Hill, W. (2000). Does authority mean quality? Predicting expert quality ratings of web documents. In *Proceedings of the Twenty-Third Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.

Anh, V. N., de Kretser, O., and Moffat, A. (2001). Vector-space ranking with effective early termination. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR.

Anh, V. N. and Moffat, A. (2002a). Impact transformation: effective and efficient web retrieval. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10.

Anh, V. N. and Moffat, A. (2002b). Improved retrieval effectiveness through impact transformation. In *CRPITS '02: Proceedings of the thirteenth Australasian conference on Database technologies*, pages 41–47, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Anh, V. N. and Moffat, A. (2004). Melbourne university 2004: Terabyte and web tracks. In *The Thirteenth Text Retrieval Conference TREC2004*.

Aslam, J. A. and Yilmaz, E. (2006). Inferring document relevance via average

precision. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 601–602.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley.

Bartell, B. T., Cottrell, G. W., and Belew, R. K. (1994). Automatic combination of multiple ranked retrieval systems. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 173–181.

Bharat, K. and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111.

Blighe, M., le Borgne, H., O'Connor, N., Smeaton, A. F., and Jones, G. (2006). Exploiting context information to aid landmark detection in sensecam images. In *ECHISE 2006 - 2nd International Workshop on Exploiting Context Histories in Smart Environments - Infrastructures and Design, 8th International Conference of Ubiquitous Computing (Ubicomp 2006)*.

Blott, S., Boydell, O., Camous, F., Ferguson, P., Gaughan, G., Gurrin, C., Murphy, N., Connor, N. O., Smeaton, A. F., Smyth, B., and Wilkins, P. (2004). Experiments in Terabyte Searching, Genomic Retrieval and Novelty Detectionn for TREC-2004. In *Proc. Thirteenth Text Retrieval Conference (TREC-13)*.

Bookstein, A. and Swanson, D. (1974). Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 25:312–318.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117.

Broder, A. (2002). A taxonomy of web search. *ACM SIGIR Forum*, 36(2):3 – 10.

Buckley, C. and Lewit, A. F. (1985). Optimization of inverted vector searches. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 97 – 110.

Carmel, D., Amitay, E., Hersovici, M., and Maarek, Y. (2001a). Juru at trec 10 - experimnets with index pruning. In *Proceedings of Text Retrieval Conference*.

Carmel, D., Cohen, D., Fagin, R., Farchi, E., Herscovici, M., Maarek, Y., and Soffer, A. (2001b). Static index pruning for information retrieval systems.

Chakrabarti, S., van den Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31:1623–1640.

Chakrabati, S., Dom, B., Gibson, D., Kleinberg, J., Kumar, S., Raghavan, P., Rajagopalan, S., and Tomkins, A. (1999). Mining the link structure of the world wide web. *IEEE Computer*, 32(8):60–67.

Clarke, C. (2006). http://plg.uwaterloo.ca/ claclark/TB06.html.

Clarke, C., Craswell, N., and Soboroff, I. (2004). Overview of the trec 2004 terabyte track. In *In Proceeding of Text Retrieval Conference (TREC)*, pages 80–88.

Cleverdon, C. (1984). Optimizing convenient online access to bibliographic databases. *Information Services and Use*, 4:3747.

Cooke, A. (2001). *A Guide to Finding Quality Information on the Internet*. Library Association Publishing, 2 edition.

Cool, C., Belkin, N., Frieder, O., and Kantor, P. (1993). Characteristics of text affecting relevance judgments. In *Proceedings of the 14th National Online Meeting*, pages 77–84.

Cortes, C. and Vapnik, V. N. (1995). Support vector networks. *Machine Learning*, 20:273–297.

Croft, W. and Harper, D. (1979a). Using probabilistic models of document retrieval without relevence information. *Journal of Documentation*, 35(4):282–295.

Croft, W. B. and Harper, D. J. (1979b). Using probablisitic model of document retrieval without relevance information. *Journal of Documentation,*, 35:285–295.

Das-Gupta, P. and Katzer, J. (1983). A study of the overlap among document representations. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 106 – 114.

Dempster, A. P. (1968). A generalization of the Bayesian inference. *Journal of Royal Statistical Society*, 30:205–447.

Eiron, N. and McCurley, K. S. (2004). Link structure of hierarchical information networks. In *Workshop on Algorithms and Models for the Web-Graph*.

Elias, P. (1975). Universal codeword sets and representations of the integers. In *IEEE Transactions on Information Theory*, volume IT-21(2), pages 194–203.

Ferguson, P., Gurrin, C., Smeaton, A. F., and Wilkins, P. (2005a). Dublin City University at the TREC 2005 Terabyte Track. In *Proc. Third Text REtrieval Conference*. Proc. Fourteenth Text Retrieval Conference (TREC-14).

Ferguson, P., Gurrin, C., Wilkins, P., and Smeaton, A. F. (2005b). Físréal: A Low Cost Terabyte Search Engine. In *Proceedings of European Conference in IR*.

Ferguson, P., Smeaton, A. F., Gurrin, C., and Wilkins, P. (2005c). Top Subset Retrieval on Large Collections using Sorted Indices. In *SIGIR '05: Proceedings of the*

28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 599–600.

Fox, E. A. and Shaw, J. A. (1995). Combination of multiple searches. In Harman, D., editor, *Proceedings of TREC-3*, pages 500–226.

Garcia, S., Lester, N., Scholer, F., and Shokouhi, M. (2006). Rmit university at trec 2006: Terabyte track. In *The fifteenth Text REtrieval Conference (TREC 2006)*.

Garcia, S., Williams, H. E., and Cannane, A. (2004). Access-ordered indexes. In *CRPIT '04: Proceedings of the 27th conference on Australasian computer science*, pages 7–14, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Garfield, E. (1955). Citation indexes for science: a new dimension in documentation through association of ideas. *Science*, 122:108–111.

Garfield, E. (1972). Citation analysis as a tool in journal evaluation. *Science*, 178:471–479.

Gemmell, J., Bell, G., Lueder, R., Drucker, S., and Wong, C. (2002). Mylifebits: fulfilling the memex vision. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 235–238.

Golomb, S. W. (1966). Run-length encoding. In *IEEE Transactions on Information Theory*, volume IT-12(3).

Google Inc. (2006). Google search engine http://www.google.com.

Gulli, A. and Signorini, A. (2005). The indexable web is more than 11.5 billion pages. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 902–903.

Gurrin, C. and Smeaton, A. F. (2004). Replicating web structure in small-scale test collections. *Journal of Information Retrieval, Special Issue on ECIR*, 7(3-4):239–263.

Harman, D. (1993). Overview of the first text retrieval conference (trec). In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 36–48.

Harter, S. P. (1975). A probabilistic approach to automatic keyword indexing (part 1). *Journal of the American Society for Information Science*, 26(4):197–206.

Haveliwala, T. (2002). Topic-sensitive pagerank. In *In Proceedings of WWW2002*, page 517526.

Haveliwala, T. (2003). Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *In IEEE Transactions on Knowledge and Data Engineering.*

Hawking, D. (2001). Overview of the trec-9 web track. In *The Ninth Text Retrieval Conference (TREC-9)*, pages 87–103.

Hawking, D. and Craswell, N. (2005). Very large scale retrieval and web search. In Voorhees, E. and Harman, D., editors, *TREC: Experiment and Evaluation in Information Retrieval.* MIT Press.

Hill, A. and Scharff, L. (1997). Readability of web sites with various foreground/background color combinations, font types, and word styles. In *Proceedings of the 11th National Conference of Undergraduate Research.*

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 13(2).

Jansen, B. J. and Spink, A. (2003). An analysis of web documents retrieved and viewed. In *The 4th International Conference on Internet Computing*, pages 65–69.

Joachims, T. (1998). Text categorization with suport vector machines: Learning with many relevant features. In *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, pages 137–142.

Joachims, T. (1999). Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA. ACM Press.

Jones, C. B., Purves, R., Ruas, A., Sanderson, M., Sester, M., van Kreveld, M., and Weibel, R. (2002). Spatial information retrieval and geographical ontologies an overview of the spirit project. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 387–388.

Jose, J. (1998). *An Integrated Approach for Multimedia Information Retrieval*. PhD thesis, The Robert Gordon University, Aberdeen, Scotland.

Jose, J. M., Furner, J., and Harper, D. J. (1998). Spatial querying for image retrieval: A user oriented evaluation. In *ACM SIGIR*, pages 232 – 240.

Keerthi, S. S. and Lin, C.-J. (2003). Asymptotic behaviors of support vector machines with gaussian kernel. *Neural Computing*, 15(7):1667–1689.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

L. Valet, Gilles Mauris, P. B. (2000). A statistical overview of recent literature in information fusion. In *Fusion Conference*.

Lamming, M. and Flynn, M. (1994). Forget-me-not: intimate computing in support of human memory. In *Proceedings FRIEND21 Symposium on Next Generation Human Interfaces*.

Lee, J. H. (1995). Combining multiple evidence from different properties of weighting schemes. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 180–188.

Lee, J. H. (1997). Analysis of multiple evidence combination. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 267–276.

Lempel, R. and Moran, S. (2001). Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19(2):131–160.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of Research and Deveopment*, pages 159–165.

Marchiori, M. (1997). The quest for correct information on the Web: Hyper search engines. *Computer Networks and ISDN Systems*, 29(8–13):1225–1236.

Maron, M. and Kuhn, J. (1960). On relevance, probabilistic indexing and information retrieval. *Journal of the ACM*, 7(3):216244.

Moffat, A. and Zobel, J. (1994). Self-indexing inverted files for fast text retrieval. In *ACM Transactions on Information Systems*.

Moffat, A., Zobel, J., and Sacks-Davis, R. (1994). Memory efficient ranking. *Information Processing and Management*, 30(6):733–744.

Montague, M. and Aslam, J. A. (2001). Relevance score normalization for metasearch. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 427–433.

Ng, A., Zheng, A., and Jordan, M. (2001). Stable algorithms for linka analysis. In *In Proceedings of 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 258–266, New York. ACM Press.

Ntoulas, A., Najork, M., Manasse, M., and Fetterly, D. (2006). Detecting spam web pages through content analysis. In *In Proceedings of the 15th International Conference on World Wide Web*, pages 83–92.

Ogilvie, P. and Callan, J. (2003). Combining document representations for known-item search. In 143-150, editor, *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*.

Osuna, E., Freund, R., and Girosi, F. (1997). Training support vector machines: an application to face detection. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition*.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project.

Park, T. (1993). The nature of relevance in information retrieval: An empirical study. *Library Quarterly*, 63(3):318–351.

Persin, M. (1994). Document filtering for fast ranking. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 339 – 348.

Persin, M., Zobel, J., and Ron-Sacks-Davis (1996). Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society of Information Science*, 47.

Plachouras, V. and Ounis, I. (2002). Query-biased combination of evidence on the web. In *Workshop on Mathematical/Formal Methods in Information Retrieval*.

Probost, F. (2000). Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets*.

Richardson, M. and Domingos, P. (2002). he intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Advances in Neural Information Processing Systems 14*.

Richardson, M., Prakash, A., and Brill, E. (2006). Beyond pagerank: machine learning for static ranking. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 707–715.

Robertson, S. E., Maron, M. E., and Cooper, W. S. (1982). Probability of relevance: a unification of two competing models for document retrieval. *Probability of relevance: a unification of two competing models for document retrieval*, 1:1–21.

Robertson, S. E. and Sparck Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society of Information Science*, 27:129–146.

Robertson, S. E., Walker, S., Sparck Jones, K., Hancock-Beaulieu, M. M., and Gatford, M. (1994). Okapi at TREC-3. In *Proc. Third Text REtrieval Conference (TREC-3)*, page 109126.

Rose, D. E. and Levinson, D. (2004). Understanding user goals in web search. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 13–19.

Salton, G. (1968). *Automatic Information Organization*. McGraw-Hill.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620.

Schamber, L., Eisenberg, M. B., and Nilan, M. S. (1990). A re-examination of relevance: Towards a dynamic, situational defination. *Information Processing and Management*, 26:755–776.

Scholkopf, B., Burges, C., and Vapnik, V. N. (1996). Incorporating invariances in support vector learning machines. In *ICANN 96: Proceedings of the 1996 International Conference on Artificial Neural Networks*, pages 47–52.

Shafer, G. (1976). A Mathematical Theory of Evidence. *Princeton University Press*.

Silverstein, C., Marais, H., Henzinger, M., and Moricz, M. (1999). Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12.

Singhal, A., Buckley, C., and Mitra, M. (1996). Pivoted document length normalization. In *19th ACM Conference on Research and Development (SIGIR)*.

Smeaton, A. F. and van Rijsbergen, C. J. (1981). The nearest neighbour problem in information retrieval: an algorithm using upperbounds. In *Proc. of ACM SIGIR conference on Information storage and retrieval*.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):1120.

Sparck Jones, K. and van Rijsbergen, C. J. (1976). Information retrieval test collections. *Journal of Documentation*, 31(1):59–75.

Spink, A. and Jansen, B. J. (2004). A study of web search trends. *Webology*, 1(2).

Strong, D., Lee, Y., and Wang, R. (1997). Data quality in context. *Communications of the ACM*, 40(5):103–110.

The World Wide Web Consortium (2007). The World Wide Web Consortium (W3C) HTML Specifications - http://www.w3.org/MarkUp/.

TREC (2000). Trec overview http://trec.nist.gov/overview.html.

Upstill, T., Craswell, N., and Hawking, D. (2003). Query-independent evidence in home page finding. *ACM Trans. Inf. Syst.*, 21(3):286–313.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc.

Vapnik, V. N. (1998). *Statistical Learning Theory*. John-Wiley and Sons Inc.

Vogt, C. C. and Cottrell, G. W. (1999). Fusion via a linear combination of scores. *Information Retrieval*, 1(3):151–173.

Voorhees, E. (1998). Variations in relevance judgments and the measurement of retrieval effectiveness. In *The 21st Annual International ACM SIGIR Conference (SIGIR)*, pages 315–323.

Voorhees, E. and Harman, D. (2000). mmnormalised.txt. In *The Eighth Text Rerieval Conference (TREC-8)*.

Wang, P. and White, M. D. (1999). A cognitive model of document use during a research pro ject. study ii. decisions at the reading and citing stages. *Journal of the American Society for Information Science*, 50(2):98–114.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis*. Cambridge University Press.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann.

Xue, G.-R., Yang, Q., Zeng, H.-J., Yu, Y., and Chen, Z. (2005). Exploiting the hierarchical structure for link analysis. In *The 28th Annual International ACM SIGIR Conference*. Association for Computing Machinery, Inc.

Y. Alp Aslandogan, C. T. Y. (2000). Diogenes: A web search agent for content based indexing of personal images. In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, pages 481–482.

Zhu, X. and Gauch, S. (2000a). Incorporating quality metrics in centralized/distributed information retrieval on the world wide web. In *Research and Development in Information Retrieval*, pages 288–295.

Zhu, X. and Gauch, S. (2000b). Incorporating quality metrics in centralized/distributed information retrieval on the world wide web. In *ACM SIGIR*

Conference on Research and Development in Information Retrieval, pages 288–295.

Zipf, G. K. (1932). *Selective Studies and the Principle of Relative Frequency in Language*. Harvard University Press.

Zobel, J. (1998). How reliable are the results of large-scale retreival experiments? In *Proceeding of the 21st Annual International ACM SIGIR Conference on Research Development in IR (SIGIR)*, pages 307–314.