Towards Understanding Latent Semantic Indexing

Bin Cheng

Supervisor: Dr. Eleni Stroulia

Second Reader: Dr. Mario Nascimento

TABLE OF CONTENTS

TABLE OF FIGURES

Abstract

The increasing amount of information available has made information retrieval tools become more and more important. Traditionally, these tools retrieve information by literally matching terms in the documents with the terms in the query. Unfortunately, because of synonymy and polysemy, the retrieval results of lexical matching approaches are sometimes incomplete and inaccurate. Conceptual-indexing techniques such as Latent Semantic Indexing (LSI) have been used to overcome the problems of lexical matching. The LSI model uses a statistical technique, singular value decomposition (SVD), to reveal the "latent" semantic structure and eliminate much of the "noise" (variability of word choice). Therefore, LSI is able to deal with the problems caused by synonymy and polysemy. Experiments show that LSI outperforms lexically matching methods on some well-known test document collections.

In this essay, we develop a complete retrieval system based on the LSI model. The experimental results show that the system can retrieve documents effectively. We also use different parameters such as rank, similarity threshold and different term composition to test the retrieval system, so that we can choose an appropriate setting to get the best retrieval results. Furthermore, we apply different retrieval performance-enhancing techniques on the system. The experimental results demonstrate that relevance feedback and query expansion techniques yield significant improvement in the retrieval effectiveness of the system. We also exploit the folding-in method to append new documents and new index terms into the collection to save the time and effort required by frequent SVD recomputing.

1 Introduction

With the recent development of the Internet, databases and digital libraries, the amount of information available is increasing at an exponential rate. Therefore, information retrieval tools, which can search a large information collection and return only the relevant information according to the user's information request, have become more and more important. Traditionally, information is retrieved by literally matching terms in the documents with the terms in the query. However, the retrieval results of the literally matching retrieval approaches sometimes are inaccurate and incomplete for two main reasons. Firstly, the same concept can usually be expressed in many ways. The terms in the user's query may not appear in a relevant document, and thus, some relevant documents cannot be retrieved. Secondly, most words have more than one meaning; as a result, literally matching the terms in the user's query may return irrelevant documents.

Latent semantic indexing (LSI) [DDF+90] is a conceptual-indexing technique that tries to overcome the problems of lexical matching. In [DDF+90], the authors propose that some underlying or "latent" semantic structure according to the overall word usage pattern is partially obscured by the variability of word choice. The LSI model uses statistical techniques to reveal this latent semantic structure and eliminate much of the "noise" (variability of word choice). Therefore, LSI is able to deal with the problems caused by synonymy and polysemy. Experiments show that LSI outperforms lexically matching methods on some well-known test document collections [Dum91][Dum95a].

In this essay, we implement an LSI information-retrieval component in the context of the EduNuggets application document collection. The main functions of the LSI information retrieval system are the following:

- Automatically infers the terms of a documents' collection,

- Obtains documents from the database used to store them,
- Preprocesses the collection documents by using stems of the terms instead of original terms and removes the stop words,
- Builds the term-by-document matrix,
- Calculates the singular value decomposition (SVD) of the term-by-document matrix, and
- Retrieves documents based on the query.

To evaluate the overall process, we run these sets of experiments on three standard well-known test collections. The experimental results show that the system can retrieve documents effectively. In the experiments, we use different parameters such as rank, similarity threshold, and different term compositions to test the retrieval system, so that we can evaluate the impact of the of the various parameters on the quality of the results. Furthermore, we apply different retrieval performance enhancing techniques on the system and run experiments on different standard collections to evaluate those techniques. The experiment results demonstrate an obvious improvement by using the relevance feedback and query expansion techniques. We also exploit the folding-in method to append new documents and new index terms into the collection, in order to save the time and effort required by frequent SVD recomputing.

The rest of this essay is organized as follows. Chapter 2 is a review of basic concepts of information retrieval and vector-space models. Chapter 3 discusses Latent Semantic Indexing (LSI) and techniques that can be used to improve the performance of LSI system. Chapter 4 describes the implementation details of the LSI system. Chapter 5 presents the experimental results of the LSI system. Chapter 6 concludes the study.

2 Related works

This chapter provides background knowledge about information retrieval (IR). This knowledge is necessary to understand the LSI and IR techniques that can be used to improve the performance of IR systems.

2.1 Traditional Information Retrieval Systems

Traditionally, textual information is retrieved from a document database by literally matching terms in the documents with the terms appearing in a query. In some cases, the literally matching approaches can retrieve useful information efficiently from the database. However, users of an information system usually want to retrieve relevant documents according to a certain conceptual meaning of a document other than according to certain query terms; therefore, in many cases, the literally matching information retrieval system cannot satisfy the user's information request and becomes inaccurate and less useful.

In this section, we discuss the index terms that are frequently used in an IR system, the problems of traditional IR systems, and the techniques that can be used to improve the performance of IR systems.

2.1.1 Index terms

Traditional information retrieval systems usually rely on a set of index terms to index and retrieve documents. An "index term" is a word or a phrase that describes the semantics of a document. In written language, some words may have more meanings than others. Usually, nouns are more representative of the semantics of a document than other words. Particularly in some domain-specific document collections, the terms of that specific area can always interpret the concepts of the documents much better than other terms in the

document collection [BR99]. Therefore, it is usually considered worthwhile to preprocess the text of the documents in the collection to determine which terms should be used as index terms. However, it is always difficult to decide what index terms should be assigned to a given document. Doing so requires a great deal of effort by experts in the relevant area. Moreover, different experts may assign different index terms to the same document, and the same expert might assign different index terms to the same document at different times. Because of the huge expense of selecting index terms from text collections, many methods that can automatically derive index terms in an unsupervised manner from text collections are being used. For example, one simple but efficient strategy for choosing index terms is to use the most frequently used nouns as index terms. Experimentation has shown that automatically derived index terms are comparable with carefully crafted manual index terms. The most modern information retrieval systems use automatically derived words and phrases as index terms for documents. However, no indexing strategy is perfect, and an indexing strategy may thus fail to index the relevant documents under the query terms even though the documents are very relevant to those terms.

2.1.2 Problems of Tradition Information Retrieval Methods

As we mentioned in Section 2.1.1, traditional information retrieval methods retrieve documents according to the specific query terms instead of the conceptual meaning of the query; therefore, the performance of traditional IR systems is not as good as the user's expectation for some relevant documents are missed, and some irrelevant ones are retrieved. In information retrieval, recall and precision are always used to measure the performance of information retrieval systems. "Recall" is the fraction of the relevant documents that are returned by the system. "Precision" is the fraction of relevant documents in the set of returned documents [BR99]. We use the recall-precision curve to

evaluate the performance of the system. This curve reflects precision at different levels of recall.

A basic problem of current information retrieval methods is that the users often do not use the same words as the index terms by which the documents they are interested in have been indexed. This issue involves two factors*:* synonymy and polysemy [DDF+90][BR99]. Therefore, users of the information retrieval system will make the same information request by using different terms. The prevalence of synonyms tends to decrease the recall performance of retrieval systems. "Polysemy" refers to the fact that most words have more than one distinct meaning. In different contexts, the same term may have different meanings. Therefore, a document containing or indexed by a term or terms appearing in the query is not necessarily a relevant document of that query. Polysemy is one reason for poor precision. [DDF+90]

2.1.3 Techniques used to improve the performance of IR Systems

2.1.3.1 Stemming

In information retrieval systems, terms are usually identified by their word stems. A "stem" is the part of a term that remains after removing its prefixes and suffixes from the term. A "stem" is used to represent a group words that have the same conceptual meaning, such as "compute", "computed", "computing", "computation" and "computer". Without the use of stemming, the information retrieval system will treat different variants of the same word as different words and may cause the problem of vocabulary mismatch. Using stems instead of the original words can partly solve this problem. As well, stemming can reduce the storage requirements because the number of unique index terms is considerably reduced.

The Porter stemmer is a well-known stemming algorithm because of its simplicity and efficiency. It uses a suffix list and a set of rules to remove the suffixes. For example, the rule s → ∅ is used to convert plural forms of a term into their singular forms by removing the last letter s. The algorithm has 5 phases. In the different phases, the complex suffixes are removed step by step. A detailed description can be found in [Porter80]. In [Porter80], the author reports that using stemmer can reduce the size of the vocabulary by about one-third. In our experiments with different parameters on different test beds, the use of Porter's stemmer always improves the performance of the retrieval system. Therefore, we use Porter stemmer in our retrieval system.

2.1.3.2 Query Expansion

To solve the problem of term mismatches between the user's query and document contents, many researchers have concentrated on query expansion. "Query expansion" is the automatic expansion of a query with lexically related terms, which are selected from a set of co-occurrence terms or from a thesaurus [PW91]. In information retrieval, the more additional query terms are used, the more documents are returned. Because the additional terms are closely related to the query terms, some documents that have not been retrieved during the original search are retrieved. Generally speaking, query expansion enhances the recall of an IR system [Miy90][Oga91], but the enhancement of recall can cause a decline in precision. The change of the both the recall and precision of a system when using query expansion depends on the selection of additional query terms.

One method of query expansion is to expand the query with co-occurrence terms. Statistical information of the document set is used to determine the relation of two terms. Generally, if two terms are correlated, they are likely to occur together in many documents in the document collection. If one term appears in a document, the chance of

the other term appearing in the same document is expected to be relatively high. On the other hand, if two words refer to different concepts, the occurrences of the terms should not be apparently correlated [BR99]. This idea is appealing, because the relations can be easily generated from the document collections, eliminating the large manual or computational overhead of constructing and maintaining the thesaurus. However, such methods have had little success in improving retrieval effectiveness. In [PW91], Peat and Willett present the limitations to the effectiveness of using co-occurrence terms to expand queries.

Using thesauri to expand the queries is another method used to solve the problem of vocabulary mismatch. A "thesaurus" consists of (1) a list of important terms in a domain of knowledge and (2) for each term in the list, a set of related terms. In the more complex thesauri, the synonyms and related terms are organized in categories and subcategories. The user can use this kind of thesaurus to broaden or narrow the query request. In [KKC94], the authors report that using a thesaurus appropriately to expand the queries can enhance the performance of the information retrieval system significantly. However, constructing a reliable thesaurus for a specific domain of knowledge is very expensive [BR99]. Attempts have been made to construct thesauri automatically, but these attempts have not been very successful [SP99].

In addition to automatic query expansion, some information retrieval systems also use semi-automatic query expansion. Unlike fully automated methods, the semi-automatic ones involve the user in the selection the additional query terms. That is, after the user inputs a query, a list of related or similar terms is returned. The user can choose additional query terms from the list to expand the query. The authors report that this method gave better experimental results than automatic query expansion [KKC94]. However, in this case, the user's interaction is required.

More recently, in [QF93], the authors present a method of expanding the query with terms related to the entire concept of the query, rather than with terms related to individual query terms. Their experiments show that this kind of query expansion improves the retrieval effectiveness of the system noticeably.

Experimental results show that these query expansion techniques make little difference in retrieval effectiveness if the original queries are relatively detailed descriptions of the information needed. “Less detailed queries can be significantly improved by appropriate expansion of hand-chosen concepts. In fact, inexperienced users always input very short queries. It is said that the average query submitted by users to World Wide Web search engines is only two words long” [CCW95]. Therefore, it is possible to improve the effectiveness of an information retrieval system considerably by using appropriate query expansion. As we mentioned before, query expansion is a recall-enhancing technique. It has the potential to improve an original query, but in some cases, it may degrade retrieval performance [Voorhees94].

2.1.3.3 User Relevance Feedback

“Relevance feedback” is an effective query reformulation technique. It can alleviate the problem of vocabulary mismatch and also add necessary additional query terms to the original query. Relevance feedback is especially useful for beginner users, because they do not always know how to effectively express what kind of information they need. In the process of relevance feedback, the user is given a list of the retrieved documents, and after checking those documents, the user marks the relevant ones. In practice, only the top 10 (or 20) ranked documents need to be examined [BR99]. The idea of relevance feedback is that, by adding new terms from the relevant documents and modifying the term weights based on the user’s judgment of relevance, the new query can be moved

towards the relevant documents and away from the irrelevant ones. Experiments on different systems have shown that using relevance feedback can improve the performance [Rocchio71][SB90].

The following are the main advantages of relevance feedback: First, the user need not know the detailed strategies of query reformulation. The user just has to check the documents in the list and to mark the relevant ones. Second, the user need not give an effective query at the beginning, but can gain the needed information step by step [BR99]. The drawback of relevance feedback is that the user has to be involved to check the documents and indicate which ones are relevant. This task is difficult and boring for casual users.

When user-relevance feedback is not available, "pseudo" relevance feedback can be used to enhance the queries. Unlike user-relevance feedback, "pseudo" relevance feedback does not require the user to make a relevance judgment, but simply assumes that all top N documents in the ranked list of the original query are relevant, and then uses these to enhance the original query, and finally returns to the user the retrieval results of the enhanced query. In [XC96], the author claims that using "pseudo" relevance feedback can increase performance over 23% on the TREC3 and TREC4 corpora. This method's main drawback is that a large number of the assumed relevant documents might not be relevant. If so, many unrelated terms are likely to be added into the new query, which will move away from relevant documents, and thus, the system will yield poor results.

2.1.3.4 Domain Knowledge

Domain knowledge can be used in information retrieval systems to improve the performance. "Domain knowledge" is represented as a collection of frames containing rules specifying recognition conditions for domain concepts and relationships among

them [Croft86]. The description of a domain concept consists of three parts: (1) the name of the concept, (2) information about how to recognize the concept in the text, and (3) the relationships between the concept and other concepts [Croft86].

A thesaurus is a kind of domain knowledge. It is applied in information retrieval system for indexing the documents and expanding the user's query to solve the problem of term mismatch. In Section 2.1.3.2, we have discussed the use of thesauri to expand queries.

In information retrieval systems, the relationships between the concept and other concepts can also be used to reformulate the user's query. As we have already mentioned, the beginner users do not always know how to express what kind of information they really need. Therefore, domain knowledge may be used interactively to help the user to identify the actual concept of his/her query. For instance, if the user inputs "automobile" as a query term, there will be too many documents about “automobile" in the text collection. The system may use domain knowledge to ask the user whether he/she want to use a more specific term, like "car" "mini-van" or "truck", to narrow down the search term. Sometimes the system may also suggest that the user should broaden the query terms or reformulate the query with relevant or similar terms. In this process, the system may interact with the user recursively until the user gets satisfactory retrieval results. The major drawback of using domain knowledge to improve the retrieval performance is that building a practical domain knowledge base for a specific document collection is always expensive.

2.2 Vector Based Information Retrieval Methods

Vector space models are one classic subclass of information retrieval techniques. In a vector space model, a document is represented as a vector of terms. Each term in the vector has its own weight to reflect how important it is in describing the concept of the

document. Typically, the weight is decided according to the frequency of the term in that document and in the whole text collection. In Section 2.2.4, we discuss how to assign a weight to a term in a document vector.

2.2.1 Constructing a term-by-document matrix

A term-by-document matrix is used to represent a text collection. To represent a document collection with n documents and m index (or unique) terms, a $m \times n$ matrix will be used. In the matrix, each row represents one term. The columns of the matrix are the n vectors that represent the n documents with m-dimension in the space. The element in the matrix $a_{i,j}$ is the weight, which is always a function of the frequency at which term i appears in document j. Thus, all the semantic content of the text collection is contained in the term-by-document matrix, so that every document has its own position in the term-document space representing the content of the document.

2.2.2 Geometric relationships between document vectors

In information retrieval systems, the geometric relationships between document vectors are used to calculate the similarities of document vectors in content. Several geometrical measures can be used to calculate the similarity between two vectors. The most commonly used measure of similarity is the cosine of the angle between the two document vectors.

The formula of cosine similarity is

$$sim(d_j, d_k) \equiv \frac{\vec{d}_j \bullet \vec{d}_k}{|\vec{d}_j| \times |\vec{d}_k|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,k}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,k}^2}}$$

[BR99].

The higher the cosine of the angle between two vectors, the more similar the two vectors are, regardless of the Euclidean distance between them.

In the formula, $|\vec{d}_j|$ and $|\vec{d}_k|$ are the norms of the two document vectors. Because the weight of the element is always no less than 0, the range of $sim(d_j, d_k)$ is from 0 to 1. If two document vectors share no common index terms, the $sim(d_j, d_k)$ is 0; on the other hand, if two document vectors are exactly the same, the $sim(d_j, d_k)$ is 1.

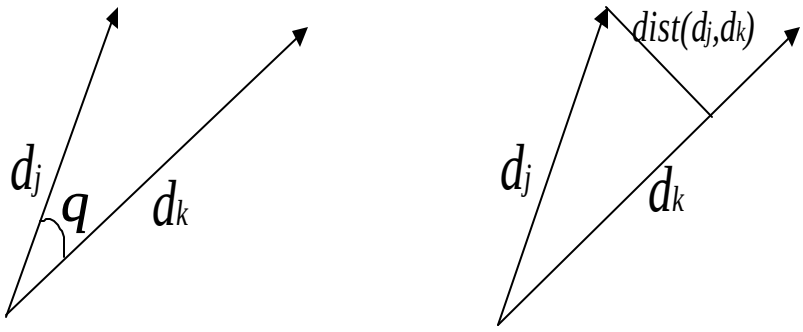


Figure 1 Two similarity methods that can be used to calculate the similarity between two document vectors d_j and d_k in the vector space.

Another similarity method that can be used is to measure the Euclidean distance between the positions of two document vectors in the space. Figure 1 shows the difference between the two similarity methods. In the left one, the cosine of θ is used to indicate the similarity between d_j and d_k. In the right one, the Euclidean distance between vectors is used to calculate the similarity between d_j and d_k. However, in some cases, the direction between vectors is more reliable for determining the semantic relevance of two documents than distance between two document vectors in the term-by-document space [FBY92].

2.2.3 Query Matching

When a user inputs a query, it is treated just like another document and is also represented as a vector. In the retrieval process, the system compares the position of the query to the location of each document in the vector space and ranks the documents according to their degree of similarity to the query. Then the system returns the top N most similar documents to the query or returns all the documents with a degree of similarity above the pre-defined threshold. One main advantage of vector space models is that they can precisely and effectively rank the relevant documents according to their similarity to the query, whereas the lexical matching methods often provide no ranking scheme or very rough ranking schemes [BR99], which may put one document above another one only because the query terms appear more often in the first one. In the retrieved document list, the users always read the documents with the greatest similarity first because they are supposed to be more semantically relevant to the query. However, to calculate the similarity between vectors, we need first to determine a scheme for how to assign a weight to each index term in the document vectors. We discuss this topic in the next section.

2.2.4 Index-terms Weighting Schemes

Many different schemes are used to calculate the weight of the index terms. Generally, a non-zero element $a_{i,j}$ of the term-by-document matrix is assigned a product of global and local weighting. Thus $a_{i,j}$ is defined by $a_{i,j} = lw(i,j) \times gw(i)$, where $lw(i,j)$ is the local weighting for term i in document j, and $gw(i)$ is the global weighting of term i in the text collection. The global weighting reflects the overall importance of the index term in the whole text collection, while the local weighting reflects the importance of the term in a particular document. Both global and local weights are used to increase or decrease the

relative importance of the index terms. For example, consider that in a document collection about computer science, the term *computer* may appear in almost every document in the collection, while an infrequent term, say *CD-ROM*, appears in only two documents. Thus, the term *computer* is not very important for describing the content of a document, and so a small global weighting should be assigned to the term *computer*, while a relatively larger global weighting should be assigned to the term *CD-ROM*. Next, consider the two documents that the term *CD-ROM* appears in: in one document, the term *CD-ROM* occurs 20 times, while in the other document, the term *CD-ROM* only appears once. It is reasonable to assign a much bigger local weighting to the term *CD-ROM* in the first document than to the term *CD-ROM* in the second document because the first document is very likely to be about the *CD-ROM*, while in the second document, the term *CD-ROM* may not be very important.

The following are the most well-known local weighting functions in information retrieval systems [Dum91]:

- Binary: $lw(i, j) = \begin{cases} 0 & if \quad tf_{ij} = 0 \\ 1 & if \quad tf_{ij} > 0 \end{cases}$
- Term-Frequency: $lw(i, j) = tf_{ij}$
- Log(Term-Frequency+1): $lw = \log_2(tf_{ij} + 1)$

In these functions, $lw(i, j)$ is the local weighting for term i in document j, and tf_{ij} is the frequency of term i in document j. Binary weighting is seldom used in vector space models. Term-Frequency is the most frequently used local weighting function. The Log(Term-Frequency+1) weighting function is used to dampen the effect of the Term-Frequency function when a large difference exists in term frequencies [Dum91].

Some popular global weighting functions include [Dum91]

- Normal: $gw(i) = \sqrt{\frac{1}{\sum_j tf_{ij}^2}}$
- GfIdf: $gw(i) = \frac{gf_i}{df_i}$
- Idf: $gw(i) = \log_2(\frac{ndocs}{df_i}) + 1$
- 1-Entropy $gw(i) = 1 - \sum_j \frac{p_{ij} \log_2(p_{ij})}{\log_2(ndocs)}, \qquad where\ p_{ij} = \frac{tf_{ij}}{gf_i}$

In these functions, $gw(i)$ is the global weighting of term i in the text collection. tf_{ij} is the frequency of term i appearing in document j. df_i is document frequency, which is the number of documents in the text collection that the term i appears in. gf_i is the global frequency at which the term i appears in the entire document collection. *ndocs* is the number of documents in the whole text collection.

In all these global weighting functions, the more frequently the term occurs in the collection, the smaller is the global weighting assigned to the term. Because the frequently appearing terms always have limited capability to describe the content of the documents, smaller global weightings should be assigned to them to de-emphasize their importance.

Depending on text collection, the results of different weighting schemes are not quite consistent. In [Dum91], the author believes that Normal and GfIdf can decrease the performance compared with the raw term frequency scheme, and that Idf, Entropy and LogEntropy can improve the performance. The author reports that in several standard text collections, using LogEntropy which is a combination of a local log weight and a global

Entropy weight, can enhance the performance up to 40% over the raw term frequency scheme.

2.2.5 Problems of Vector Space Models

The main problem of vector space models is that they still cannot solve the problem of vocabulary mismatch. The vector space models treat the terms as unrelated objects in the semantic space. For example, "*computer*" and "*laptop*" are relevant terms, but the vector space models cannot detect this kind of relevance. If "*computer*" is in the user's query, for the vector space models, "*laptop*" is one totally irrelevant term, just like the term "*river*". Thus, if the query and documents in the text collection share no common words, zero similarity values will be the result and no document will be retrieved, even if some documents may be actually relevant to the query. Another drawback of vector space models is that in large text collections, the term-by-document matrix is always a huge and sparse one, needing a lot of disk storage space. To adress these two problems, latent semantic indexing (LSI), which is a variant of the vector space model, has been studied in recent years. LSI will be discussed in Chapter 3.

3 Latent Semantic Indexing

3.1 Introduction to Latent Semantic Indexing

The Latent Semantic Indexing (LSI) [DDF+90][Dum91] information retrieval model is a variant of the Vector Space model. It attempts to overcome the problem of vocabulary mismatch that plague automatic information systems. In [DDF+90], the authors assume the existence of some underlying or "latent" semantic structure according to the overall word usage pattern that is partially obscured by the variability of word choice. The LSI model uses statistical techniques to reveal this latent semantic structure and eliminate much of the "noise" (variability of word choice). One main advantage of LSI is that it uses semantic concepts instead of individual words to index and retrieve the documents, thus allowing a relevant document to have a positive similarity to the query even when they share no common words. Experiments have shown that it substantially outperforms the Vector Space models, implying that to indicate the meaning of a document, its latent semantic concept is more reliable than its individual words [Dum91][Dum95a].

Latent Semantic Indexing is an extension of the Vector Space Model, which we discussed in Section 2.2. LSI starts with the term-by-document matrix. We discussed the term-by-document matrix and weighting scheme in Section 2.2.1 and Section 2.2.4. LSI uses a truncated singular value decomposition (SVD)[GL89] of term-by-document matrix A to approximate the original m-dimensional vector space in a reduced space of k orthogonal dimensions, where k is substantially less than m and n. After the SVD, an indexing semantic structure is constructed; that is, terms and documents are placed into a semantic space.

3.2 Singular Value Decomposition (SVD)

The SVD of the matrix A is defined as the product of three matrices,

$$A = U\Sigma V^T,$$

where the columns of the U and V have orthonormal columns and Σ is diagonal. U and V are the matrices of the left and right singular vectors, respectively, and Σ are elements in decreasing order, which are called the "singular values" of the matrix [GL89].

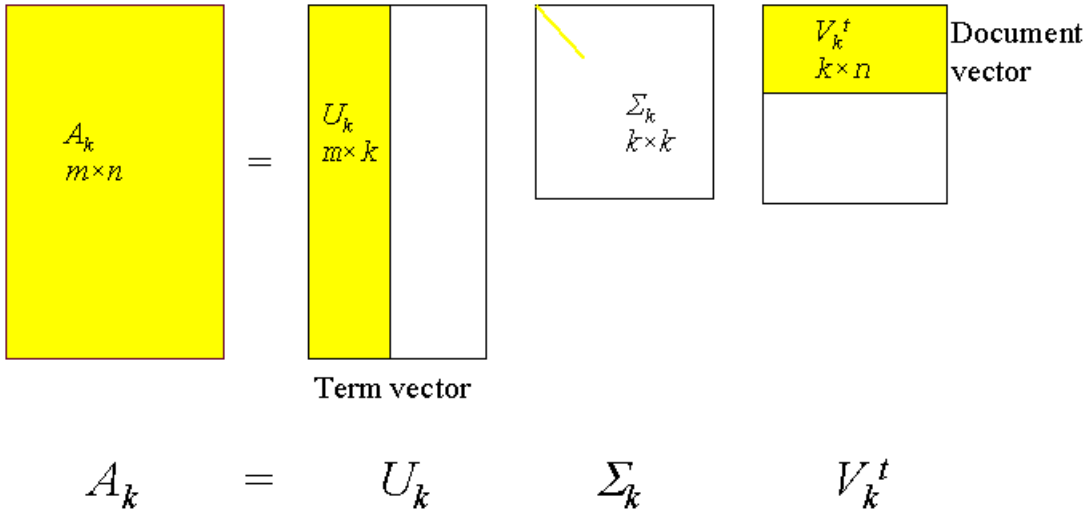


Figure 2 A pictorial representation of the SVD [BDL95].

Figure 2 shows the construction of the approximation matrix A_k. The first k columns of the U and V matrices and the first (largest) k singular values of A can be used to construct a rank-k approximation to A, which is $A_k = U_k \Sigma_k V_k^T$. U_k, V_k and Σ_k are truncated matrices of U, V and Σ. By reducing the dimensionality of A, the complexity of the original vector space is reduced, thus decreasing the disk space usage and query time needed to compare two vectors. Moreover, after the SVD, the latent semantic structure of the vector space is kept, while much of the "noise" is eliminated. In LSI systems, the choice of the rank *k* influences the performance of the system substantially [DDF+90]. If the rank *k* is too small, much pertinent data will be lost; on the other hand,

if the rank is too large (near *n* or *m*), the approximation matrix A_k is almost the same as the original term-by-document matrix *A*; therefore, the "noise" can not be eliminated, resulting in poor retrieval performance. We explain how to choose an appropriate rank in Chapter 3.4.1.

3.3 Query Projection and Matching

In the LSI model, a user's query, which is originally a *m*-dimensional vector of terms, has to be projected into a *k*-dimensional vector and then compared to every document in the text collection. The user's query can be represented by $\vec{q} = q^T U \sum^{-1}$ [DDF+90] [Dum91].

The geometric relationships between the query vector and document vectors are used to calculate the similarities in the contents of the query vector and document vectors. Several geometric measures can be used to calculate the similarity between two vectors. The most commonly used is the cosine of the angle between the query vector and document vectors. After the similarities between the query and all the documents have been computed, all the documents are ranked according to their similarity to the queries. Then the system will return the top N most similar documents to the query or return all the documents with a degree of similarity above the pre-defined threshold [BDO95] [Dum91].

3.4 Improving the performance of LSI

3.4.1 Choosing the rank

The retrieval quality of the LSI system heavily depends on its number of dimensions. As we discussed in Section 2.3.2, using an appropriately chosen dimension number (rank) can remove much of the noise and keep enough useful information to capture the latent structure in the term-document matrix. If too few dimensions are used, too much

statistical information will be lost, and the latent structure cannot be reflected sufficiently. However, if too many dimensions are used, the noise or irrelevant detail cannot be removed. Experiments show that LSI works well when using a relatively small rank compared with the number of documents and the number of terms [Dum91][DDF+90].

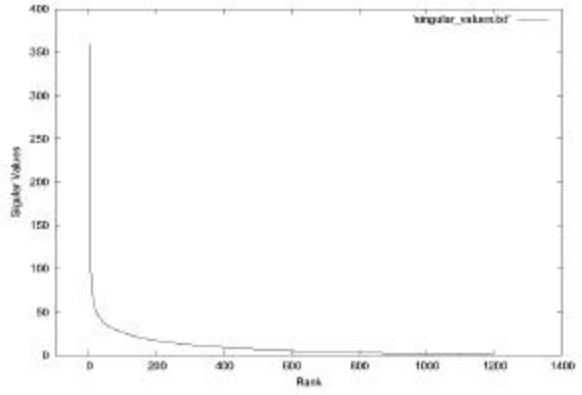


Figure 3 The distribution of singular values of the TIMES text collection.

Figure 3 shows the singular values distribution of the TIMES text collection. They decrease significantly at the beginning and become quite small after that and then decrease slowly, so most of the useful information can be kept when using a relative small number of dimensions. The small singular values are considered to be noise or non-relevant details. In [DDF+90], the authors report that for the MED text collection, which has 1033 documents and 5823 unique terms, the best retrieval results were obtained when the rank was 100, which is relatively small compared with the original dimensions of the text collection.

3.4.2 Relevance feedback

Relevance feedback (See Section 2.1.3.3) can be used to improve the performance of an information retrieval system [SB90]. Relevance feedback uses the terms appearing in the relevant documents to reformulate the original query and makes the query move to the

relevant documents in the semantic space. Because LSI represents terms, documents, and the user's queries in the same space, query feedback can be done effectively. Generally, a reformulated query vector is a combination of the original query vector and relevant document vectors. In [Dum91], the author uses the first relevant document vector or an average of the first three relevant document vectors instead of a combination of query and relevant documents as the queries, and gets much better retrieval results. Relevance feedback can be used iteratively until satisfactory retrieval results are obtained.

3.4.3 Query expansion

Query expansion (See Section 2.1.3.2) can be used to deal with the synonymy problem, and therefore to enhance the performance of LSI system. Terms occurring in a similar context of documents are near each other in LSI space and are always similar or closely related. Therefore, we can use LSI as an online thesaurus to expand the original query automatically by adding the terms that are similar enough to the query terms, instead of using a manually constructed thesaurus, which is very expensive to construct and maintain.

The following is an example of providing semantically related terms to the query term in the TIMES text collection.

Original query term: **buddhist**

buddhist	1.0
priest	0.8979113
catholic	0.82500756
monk	0.8062161
religion	0.6171877
nun	0.60505223
xa	0.5798228
pagoda	0.5521758

The figures after the terms are the similarities between the returned terms with the query term input by the user. Top 6 nearby terms that were returned are closely related to the query term. We can set an appropriate similarity threshold, for example, 0.6, and make the system automatically add all the terms that are similar enough to the query. If an appropriate threshold is set, the automatic query expansion has great potential to describe the user's information request more effectively and completely, and therefore to improve the performance of the LSI system.

3.3.4 SVD-Updating

New terms and documents are commonly added into an information retrieval system, making the current indexes obsolete or incomplete. For LSI, the most straightforward method for adapting to the addition of new terms or new documents is to recompute the new SVD for the modified term-by-document matrix; however, for a large text collection, the computing of SVD is very expensive in terms of space and time. Therefore, folding-in has been studied [DUM91]. Folding-in is very inexpensive in time and space, but the new vectors do not affect the existing vectors. Therefore, after the folding-in, the updated SVD is not the exact representation of the updated text collection, so that performance might decline. However, experimental results show that retrieval results of folding-in are almost as good as the results obtained from the SVD recomputing. In [Dum95b], the author reports that the experimental results of using LSI to index 50% of the documents and folding-in the remaining documents are indistinguishable from the results of using LSI to index all documents.

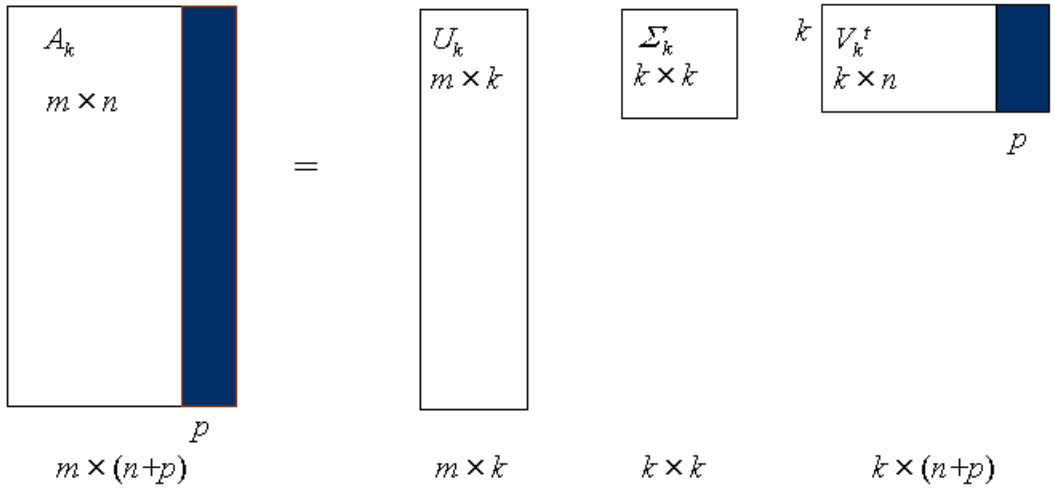


Figure 4 Mathematical representation of folding-in p documents [BDL95][O'Brien94].

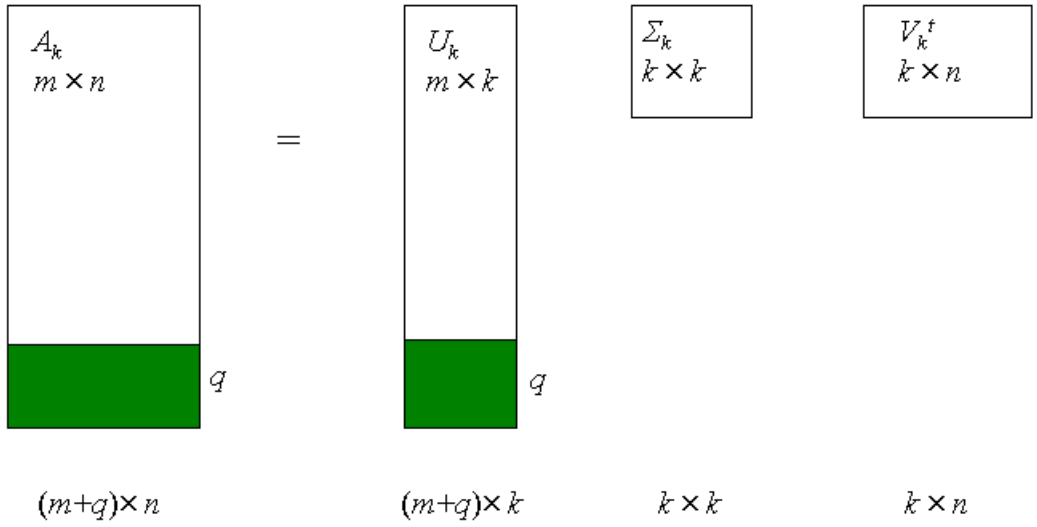


Figure 5 Mathematical representation of folding-in q terms [BDL95][O'Brien94].

The process of folding-in is the same as the process of generating a pseudo-document from queries (See Section 3.3). Each new document is projected into a *k*-dimensional vector. *k* is the rank used in the LSI retrieval system. In LSI space, the new document is

represented by $d_p = d_t V_k \sum_k^{-1}$ and then a new vector is appended to the set of existing document vectors, that is, the columns of V_k in Figure 4 [BDL95][O'Brien94].

Similarly, new terms are projected into the k dimensional vectors represented by $t_q = t_d U_k^T \sum_k^{-1}$ and then appended to the set of existing term vectors, that is, the column of U_k in Figure 5 [BDL95][O'Brien94].

4 Design and Implementation of the EduNuggets LSI Component

We have described the LSI model in Chapter 3. In this chapter, we discuss the design and implementation of the LSI system.

The LSI information retrieval system supports two functionalities

1) Preprocessing, and
2) Run-time querying.

In the preprocessing phase, the term-by-document matrix is built and is decomposed into three matrixes by using singular value decomposition (SVD). That is, LSI space is built in this phase. In the querying phase, queries are projected into the LSI space and documents that are near the query are returned according to a certain similarity measure. The preprocessing phase results in a one-time cost for a stable document collection. It is independent of the number of the queries executed in the second phase and will not affect the execution efficiency of the second phase.

In the following section, we describe the implementation of these two phases.

4.1 Implementation of preprocessing phase:

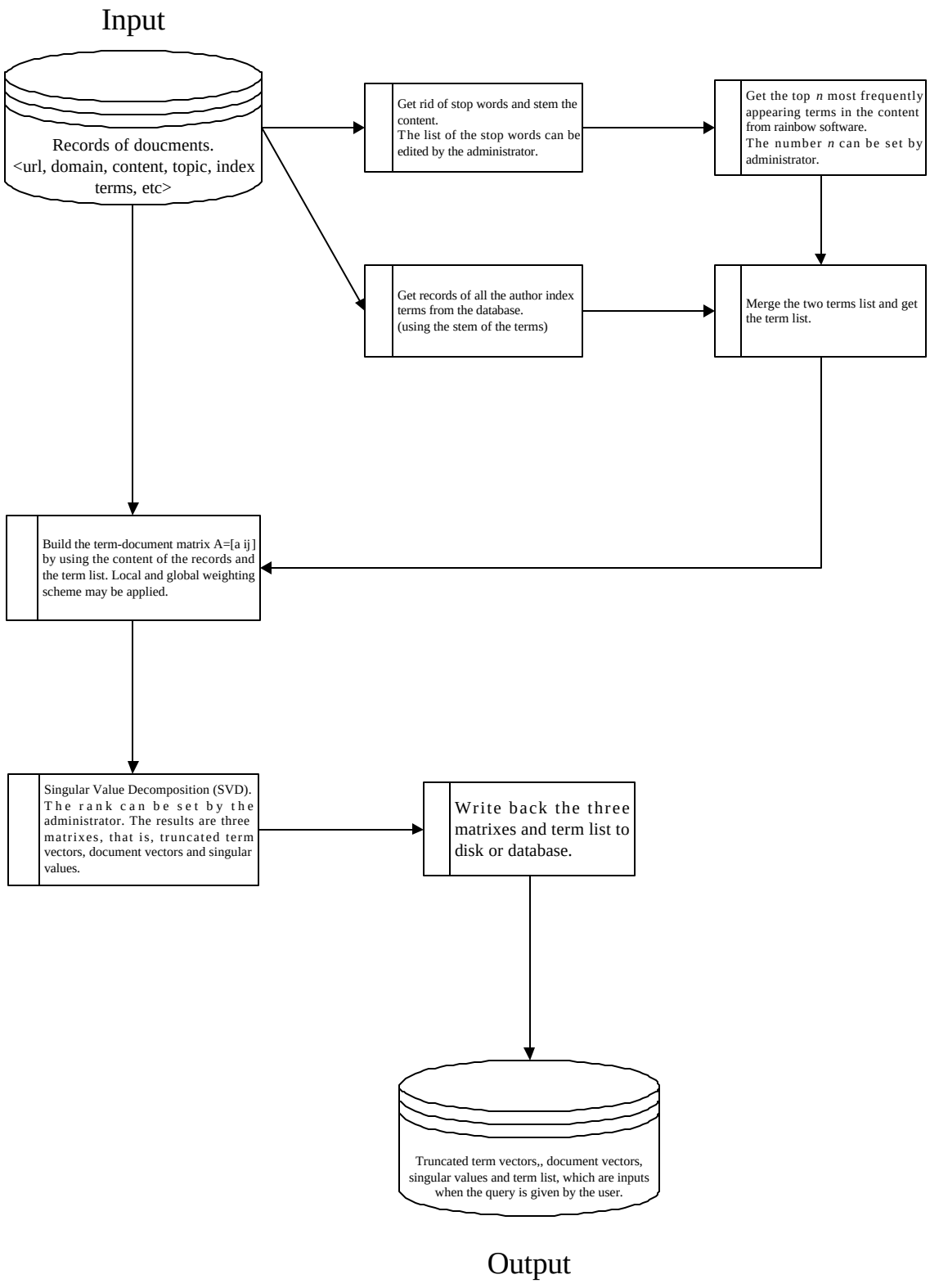


Figure 6 The preprocessing phases of LSI.

Figure 6 shows the process of the preprocessing phase, in which the LSI performs the following major tasks.

4.1.1 Connect with the document database and obtain the nuggets from the database.

All the documents are stored in the database as the records. Each document includes its content and the URL (or filename), domain, index terms attached to the nugget. At the beginning of preprocessing phase, all these records of nuggets are obtained from the database.

4.1.2 Eliminate the stop words from the contents and use the stems instead of the original words.

With the LSI system, in order to save space and to speed up the preprocessing phrase and the searches, we do not index the extremely common words, known as "stop words." They have no power to distinguish documents from one another. In our LSI system, we use the Brown Corpus Most Frequent word list produced by [Brown_Corpus]. The stopword list can be edited by the administrator of the system to include the extremely frequently appearing words in a specific document collection. When the user inputs a query, the terms that appear on the stopword list are removed from the query because they are not indexed in the document collection.

In our LSI retrieval system, we use the Porter stemmer [Porter80]; that is, in the system, documents and queries are all truncated into their stems according to Porter's suffix-stripping algorithm. See Section 2.1.3.1 for more details about the Porter stemmer.

4.1.3 Prepare the index term list

The index terms that are used to build term-by-document matrix come from two term lists. One is composed of the index terms of the nugget collection, which are assigned to the nuggets by the instructor. The other term list is composed of the frequently appearing terms of the nugget collection. For the index term list, we need only to connect to the database and get the record of the index term list from the database. We use the stems of

the index terms instead of the original terms. For the frequently appearing terms, we use Rainbow software [Rainbow] to get the top *n* most frequently appearing terms from the contents. The number *n* can be set by administrator. Because the contents that we use in this step are stemmed, the frequently appearing terms we get from Rainbow software are also in their stemmed forms. Then we merge the index term list and frequent term list into the final index term list for the term-by-document matrix.

4.1.4 Build the term-by-document matrix.

A term-by-document matrix is used to represent a text collection. To represent a document collection with *n* documents and *m* index terms, an $m \times n$ matrix will be used. The elements in the matrix are the weights. See Section 2.2.1 and Section 2.2.4 for a more detailed explanation of the construction of a term-by-document matrix and index-terms weighting schemes. After the construction, all the semantic content of the text collection is contained in the term-by-document matrix. In this step, the contents and the index term list are used. In our LSI system, we use a string-matching algorithm to match the index term and the content. Therefore, the index term can be phrased with several words. For example, *Graphical User Interfaces* can be one index term in our retrieval system instead of three.

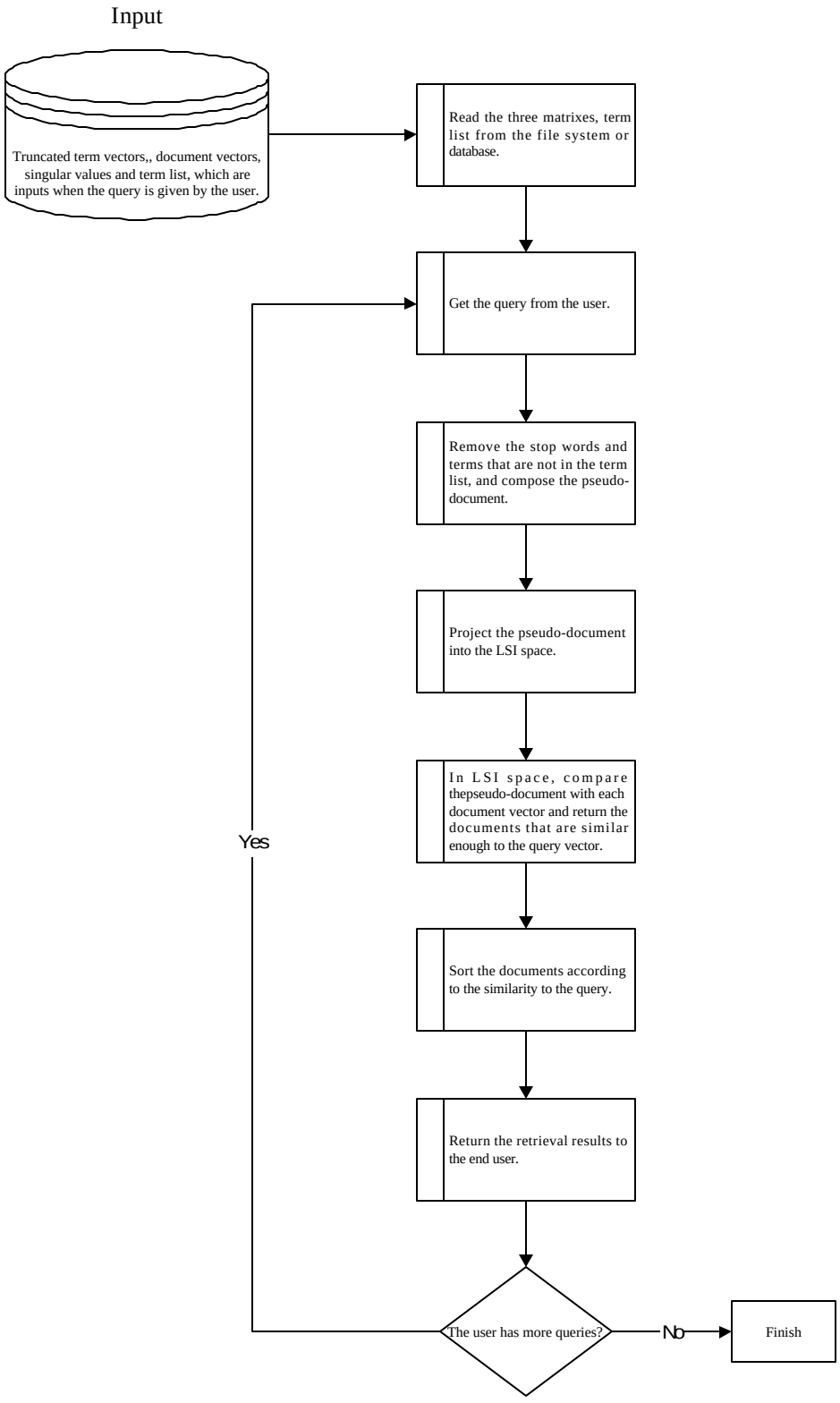
4.1.5 Singular Value Decomposition (SVD)

During step 4, an $m \times n$ matrix A is built. In step 5, singular value decomposition (SVD) [GL98] is used to decompose the original term-by-document matrix into three truncated matrixes, which are term matrix, document matrix and singular values. (See Section 3.2 for more details) The administrator can set the rank used in SVD. In Section 3.4.1, we have discussed how to choose an appropriate rank.

4.1.6 Write the truncated term matrix, document matrix, singular values, term list and records of nuggets back to the file system.

In this step, the truncated term matrix, document matrix, singular values, term list and records of nuggets are written back to the file system for later use during the query phase.

4.2 Implementation of query phase:



4.2.1 Obtain the three matrixes, term list and record of nuggets from file system.

In step 1, the three matrixes, term list and record of nuggets are obtained for the file system for use during the query phase.

4.2.2 Obtain the query from the user and preprocess the query.

In this step, the query is obtained from the user. Stop words that appeared in the query are eliminated. We also use the Porter stemmer [Porter80] to convert the query terms into their stems because the term list and contents of documents in the LSI are also in their stem forms.

4.2.3 Project the query to the LSI space

We can then treat the query as a pseudo-document, which is a *m*-dimensional vector (*m* is the number of terms used in that document collection), and project it into the *k*-dimensional vector by $\vec{q} = q^T U \sum^{-1}$ [DDF+90]. After the projection, the query (or pseudo-document) is represented by a *k*-dimensional vector in the LSI space. See Section 3.3 for a more detailed explanation of the pseudo-document and query projection.

4.2.4 Return the documents that are similar enough to the pseudo-document

As we discussed in Section 2.2.2, several different measures can be used to determine the similarity between two vectors in the vector space. The most commonly used similarity measure is the cosine similarity, which calculates the angle between two vectors in the vector space. The higher the cosine between two vectors, the more similar the two vectors are, regardless of their Euclidean distance. In our implementation, the cosines between the pseudo-document vector and each document vector are calculated and each document with a higher cosine than the pre-defined threshold will be put into the answer set. The administrator can set the similarity threshold. Next, all the documents in the answer set are sorted according to their cosine similarity to the pseudo-document and returned to the

user, so that the user can first read the documents that are supposed to be most relevant to the query. For each returned document, its similarity to the pseudo-document is also displayed to indicate how similar it is to the original query.

4.3 Using information retrieval techniques to improve retrieval results

We use query expansion and relevance feedback techniques to improve the retrieval results of the system.

4.3.1 Query Expansion

In Section 2.1.3.2 and Section 3.4.3, we have discussed query expansion. To implement query expansion, we need only to modify step 2 of the query phase. In step 2, after we get rid of the stop words from the query and do the stemming, we calculate the cosine similarity between each term vector and each query term vector. Every term with a higher cosine similarity to any query term than the pre-defined threshold will be put into the query as an additional query term. The expanded query will be used in step 3 as the pseudo-document and projected into the LSI space. The administrator can set the term similarity threshold for best retrieval results.

4.3.2 User's Relevance Feedback

In Section 2.1.3.3 and Section 3.4.2, we have discussed the user's relevance feedback. For it, we need to modify step 4 of the query phase. After the user gets the retrieval results from the retrieval system, the user selects the most relevant document, and the system will retrieve documents again, but this time, the system will use the most relevant document vector as the query vector instead of using the original query vector. Relevance feedback has great potential for improving the retrieval results. Relevance feedback can be used iteratively until satisfactory retrieval results are obtained.

5 Experimental Evaluation

In our experiments, we used the three well-known standard test collections shown in Table 5.1. The test collections consist of documents, human-language queries and relevance information about the queries as answers.

Test Collection	Brief Description	Number of Documents	Number of Queries
TIMES	Articles from Time magazine's world news section in 1963	425	83
CFC	The Cystic Fibrosis Database (CF) consists of documents published from 1974 to 1979 discussing Cystic Fibrosis Aspects	1239	100
CRAN	Document abstracts in aeronautics and related areas originally used for tests at the Cranfield Institute of Technology in Bedford, England.	1400	225

Table 5.1 Summary information about these three test collections.

Recall and precision are always used to measure the performance of information retrieval systems. "Recall" is the fraction of the relevant documents that are returned by the system. "Precision" is the fraction of relevant documents in the set of returned documents [BR99]. We use the recall-precision curve to evaluate the performance of the system. The recall-precision curve reflects precision at different levels of recall.

5.1 Number of index terms

We have discussed index terms in Section 2.1. Before we performed this set of experiments, we asked the question, "How many index terms should be chose from the text collections". Our hypothesis was that our results would show that the more index terms are used, the better the retrieval results is.

In this experiment, we use index term sets of different numbers to index the same document collection. The index term sets are composed of different numbers of the most frequently appearing terms generated from Rainbow software [Rainbow]. The three sets have the 500, 1000 and 2000 most frequently appearing terms, respectively. No stop words are in the term list, and all the terms are listed in their stems by using the Porter Stemmer [Porter80].

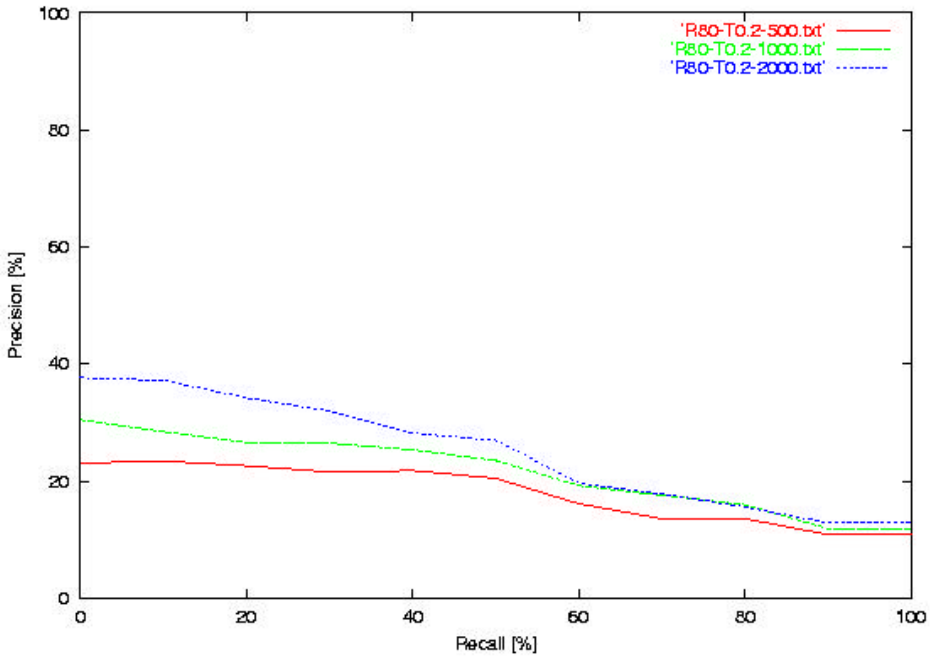


Figure 7 The recall-precision curves of index term sets with different number of index terms used in the CFC text collection. Rank is 80 and similarity threshold is set to 0.2.

Figure 7 shows that the more index terms used in the system, the better the retrieval results. We also did experiments on different text collections with different settings, and

the results were quite consistent. Figure 8 shows another two sets of recall-precision curves.

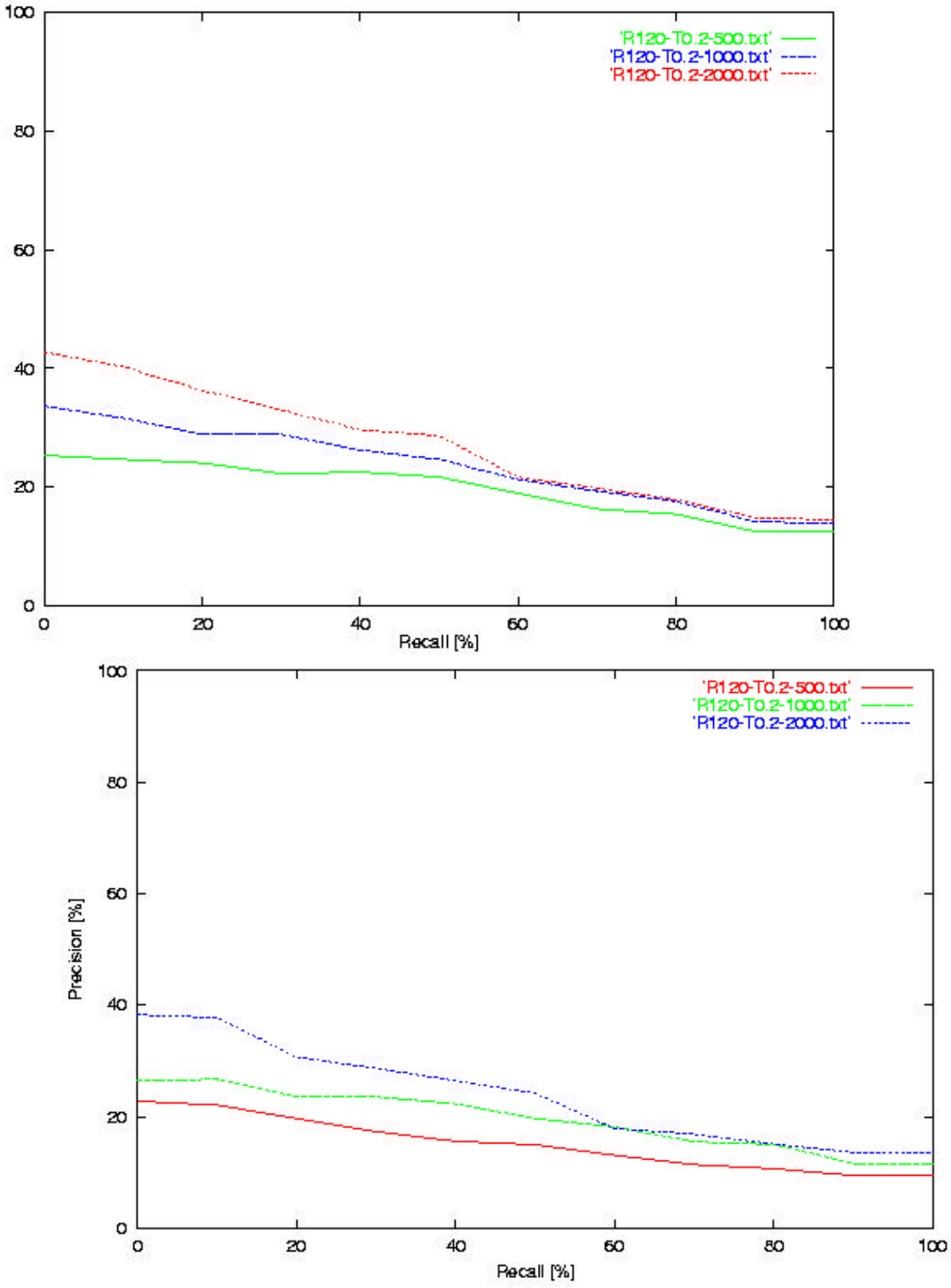


Figure 8 The recall-precision curves of index term sets with different number of index terms. Rank is 120 and similarity threshold is set to 0.2. Text collection for the top one is the TIME, for the bottom one is the Cran.

The experiment results are reasonable because the more index terms used, the more useful the information is in the document collection captured by the retrieval system, which therefore, yields better retrieval results. Consequently, to yield acceptable retrieval results from a practical retrieval system, we must use a set of index terms large enough to capture the system's useful information. However, if too many index terms are used in the system, preprocessing the documents and executing the queries will require more time, and the efficiency of the retrieval system will be compromised.

5.2 Quality of index terms.

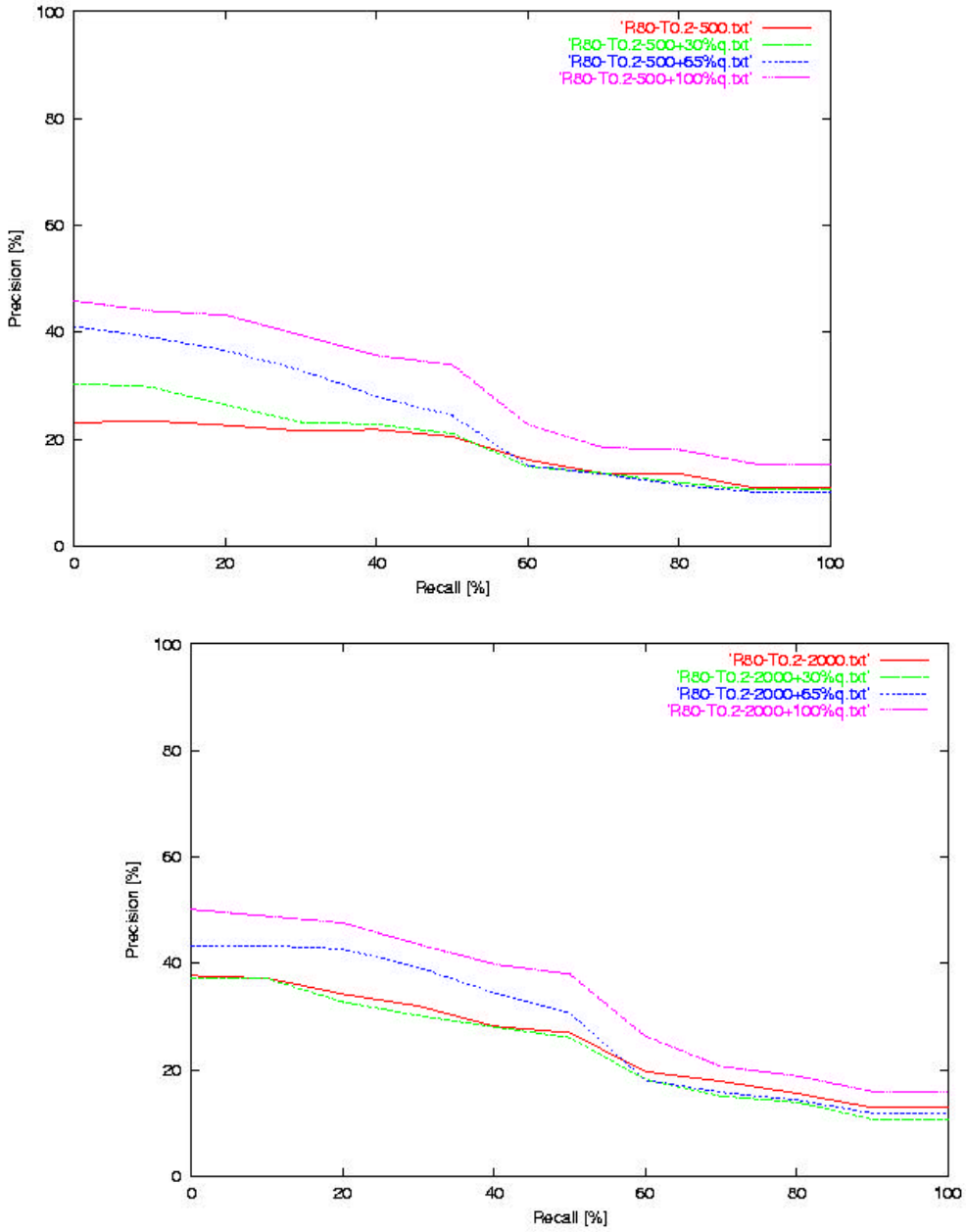


Figure 9: The recall-precision curves of index term sets with different composition terms used in the TIME text collection. Rank is 80 and similarity threshold is set to 0.2. The number of frequent appearing terms is used in the top one is 500 and in the bottom one is 2000.

In this set of experiments, we tried to find out how the quality of the index terms can affect the retrieval results. As we discussed in Section 2.1.1, some words may carry more meaning than others. In some document collections, the terms of the specific domain can describe the concepts of the documents much better than other terms in the document

collection. Therefore, it is useful to add more useful terms to the index terms list or get rid of the terms that cannot be used to distinguish different concepts. Before we performed this set of experiments, we expected that the experimental results would show that better index terms would yield better results on the same text collection.

We use the terms that appeared in the queries as terms of good quality, because these terms are useful for describing the concepts in that specific document collection, and the users (or experts) are more interested in them than in the collection's other terms. We select all the terms from the queries and get rid of the stop words. As well, all the query terms are in their stem forms.

We constructed 4 sets of index terms of each document collection by using different setting; that is, the top N most frequently appearing terms, top N most frequently appearing terms and 30% query terms, top N most frequently appearing terms and 65% query terms, and the top N most frequently appearing terms and all the query terms. The query terms that are added into the index term list are selected randomly.

In Figure 8, in the left one, the top 500 frequently appearing terms are used as the base of the index terms. In the right one, the top 2000 frequently appearing terms are used as base. In both situations, the experiments with index terms of better quality yield better results. We also did the experiments on different text collection with different settings, and the results show that using carefully chosen index terms can noticeably enhance the retrieval results of the retrieval system. Figure 10 shows the results on the CFC text collection with different settings.

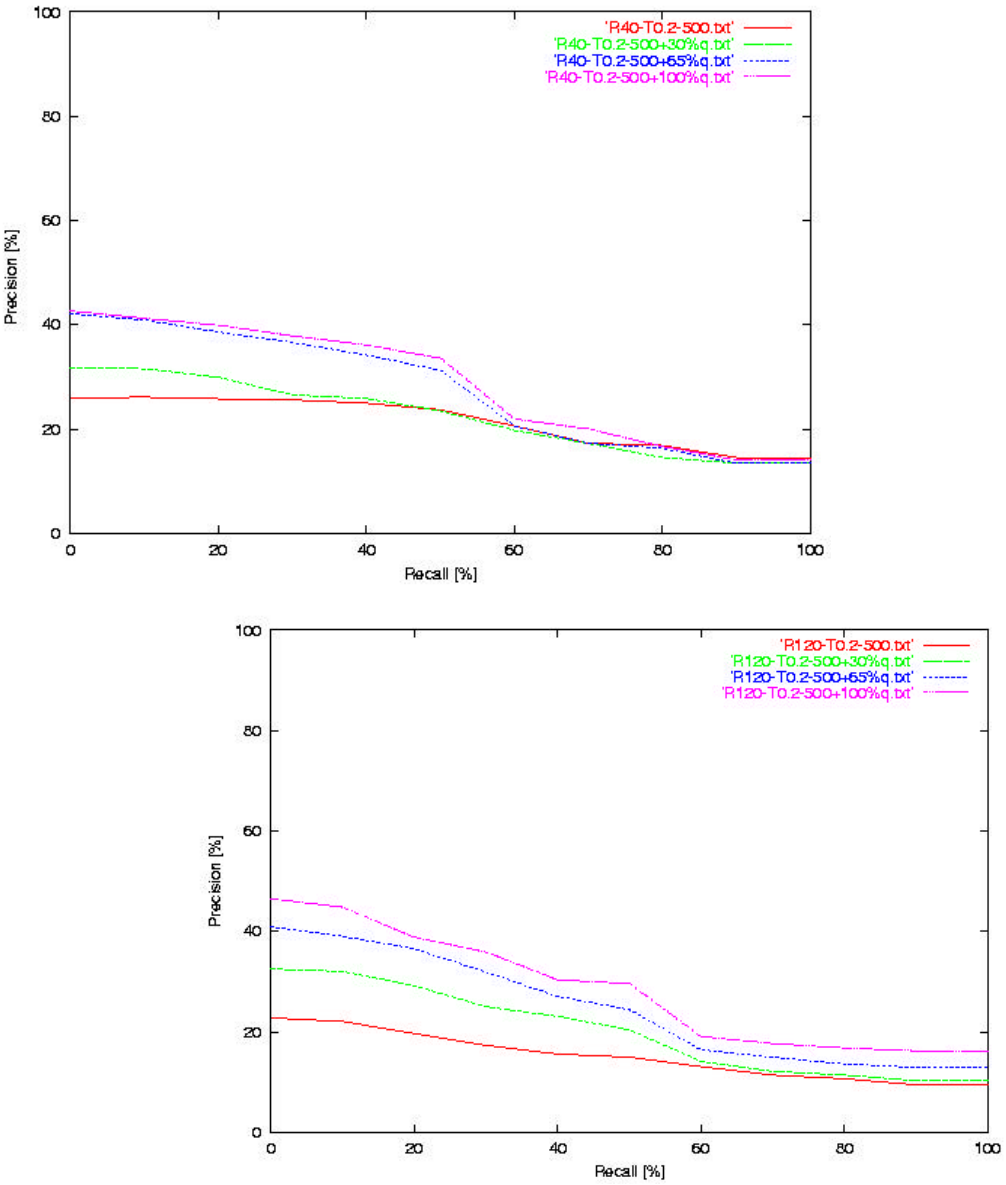


Figure 10 The recall-precision curves of index term sets with different composition terms used in the CFC text collection. The number of frequently appearing term is 500. Similarity threshold is 0.2. Rank is 40 for the top one and rank is 120 for the bottom one.

To sum up, index terms of better quality can yield better retrieval results. Therefore, in a domain-specific document collection, we can add the terms of that specific area as index terms to improve the retrieval performance. In our retrieval system, the index term list can be modified by the administrator.

5.3 Rank of SVD

The retrieval quality of the LSI system heavily depends on its number of dimensions. As we discussed in Section 3.4.1, using an appropriately chosen dimension number (rank) can remove much of the noise, but retain enough useful information to capture the latent structure in the term-by-document matrix. If too few dimensions are used, too much statistical information will be lost, and the latent structure cannot be reflected sufficiently. However, if too many dimensions are used, the noise or irrelevant detail cannot be removed. Experiments show that LSI works well when using a relatively small rank compared with the number of documents and the number of terms [Dum91][DDF+90]. We performed this set of experiments in order to learn how to choose an appropriate rank for then LSI retrieval system.

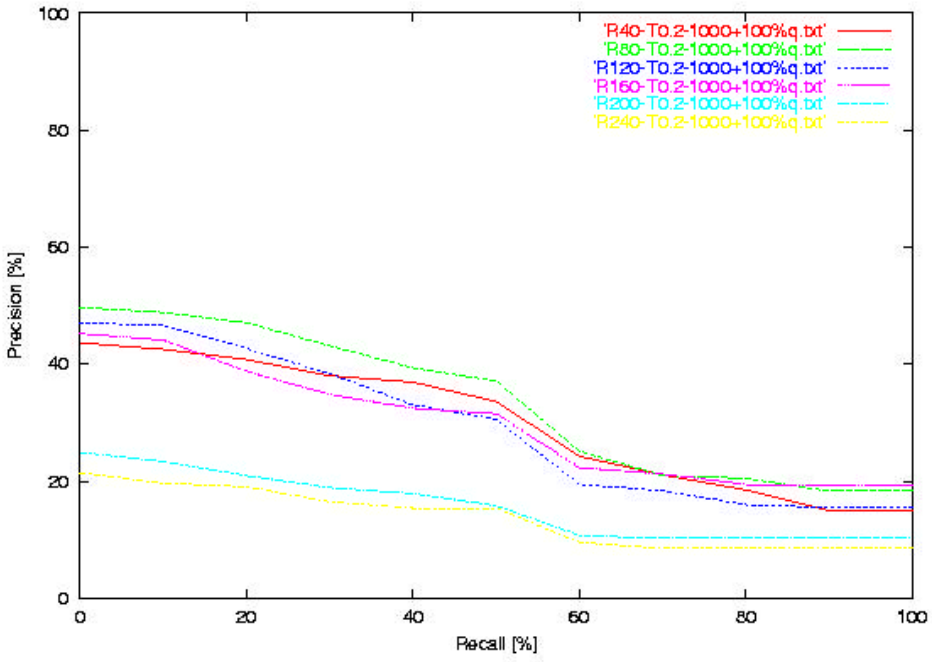


Figure 11: The recall-precision curves of different rank. The text collection is CRAN. Similarity threshold is set to 0.2 and the index terms are 1000 most frequently appearing terms and all query terms.

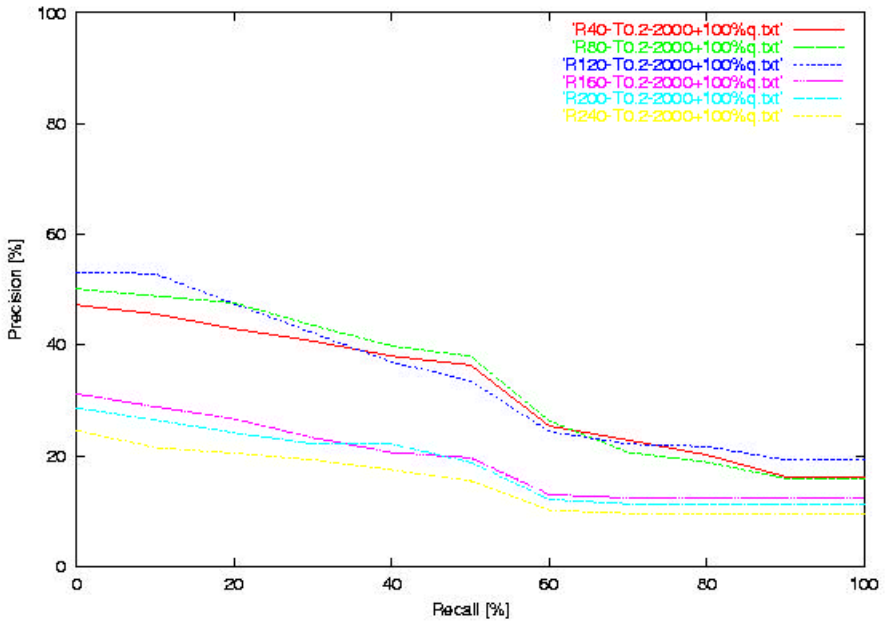


Figure 12 : The recall-precision curves of different rank. The text collection is CRAN. Similarity threshold is set to 0.2 and the index terms are 2000 most frequently appearing terms and all query terms.

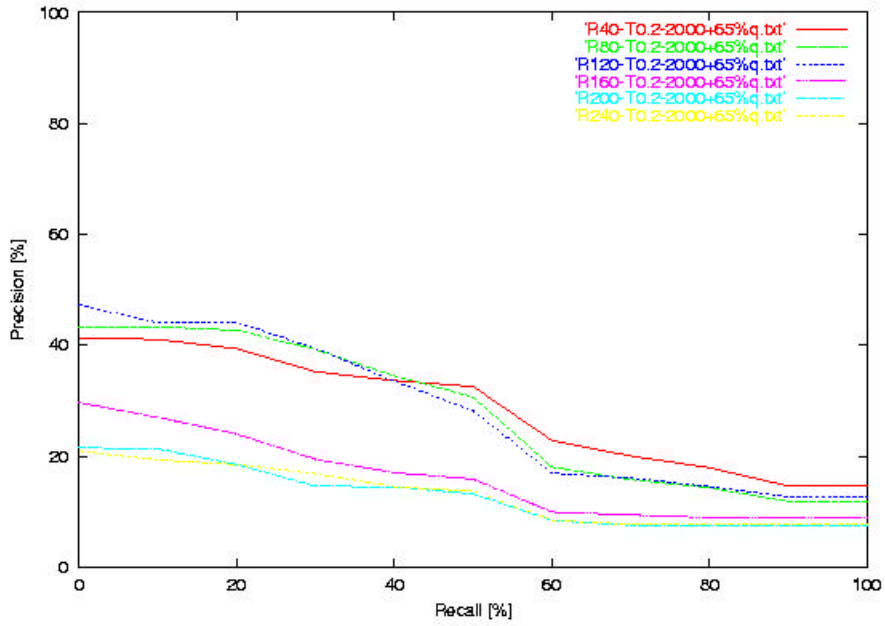


Figure 13 The recall-precision curves of different rank. The text collection is CRAN. Similarity threshold is set to 0.2 and the index terms are 2000 most frequently appearing terms and 65% of the query terms.

Figure 11, 12 and 13 shows the recall-precision curves of different rank on the CRAN text collections with different setting. These figures reveal that the overall performance is best when the rank is set to 80 or 120. When rank is set to 160 and 40, the performance

decreases, but the retrieval results are still acceptable. However, when the rank is set to 200 and 240, the performance is much worse than that obtained by using other rank settings. In LSI systems, using more dimensions does not mean a better performance. We also tested the system on other text collections and obtained similar results. That is, the performance is not good when the rank is too small or too large. In the experiments on different text collections of different settings, the rank that yields best results may vary, but when the rank is around 100, the results are always good.

5.4 Relevance feedback

As we discussed in Section 2.1.3.3 and Section 3.4.2, retrieval effectiveness can be improved significantly by using relevance feedback. "Relevance feedback" is a process that uses the user's relevance information to enrich or reformulate the query interactively and recursively. A beginner user always has difficulty expressing his/her information needs, but the terms appearing in relevance documents can be used to enrich the query.

In this experiment, we use the first relevant document as the refined query, instead of using a combination of the original query and relevant documents. We use this experiment to imitate the situation in which the user finds the first retrieved document that is really relevant to his/her information request and wants to have more documents like it. We ran these experiments in order to find out whether the relevance feedback can improve the retrieval results. We expected that a noticeable enhancement could be obtained by using relevance feedback.

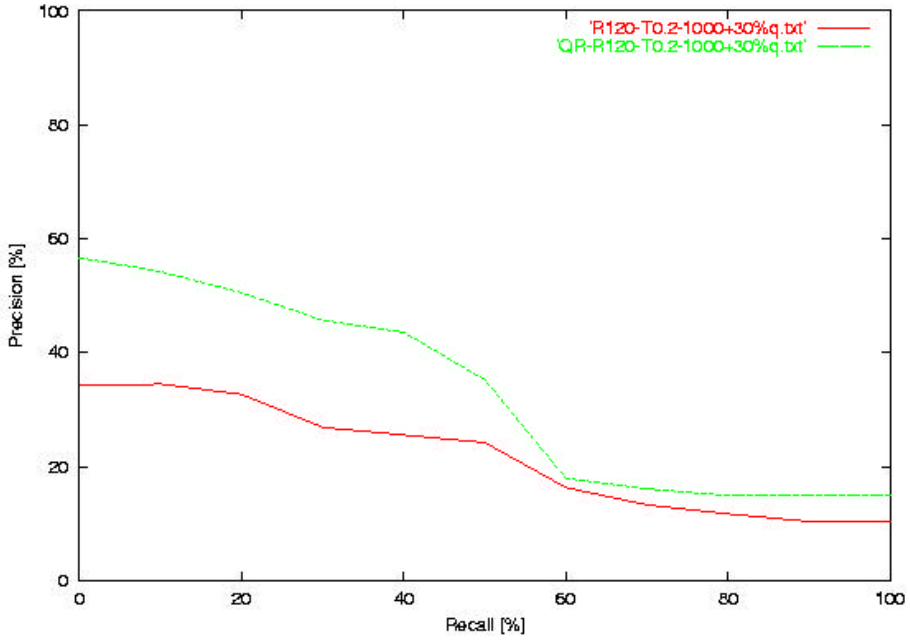


Figure 14: The recall-precision curves of original query and refined query. Text collection is the CFC. Similarity threshold is set to 0.2 and the index terms are 1000 most frequently appearing terms and 30% query terms.

Figure 14 shows how this kind of relevance feedback improves the performance of the retrieval systems. The lower curve is the recall-precision curve of the original queries. The upper one is the recall-precision curve obtained by using the user's relevance feedback as refined query.

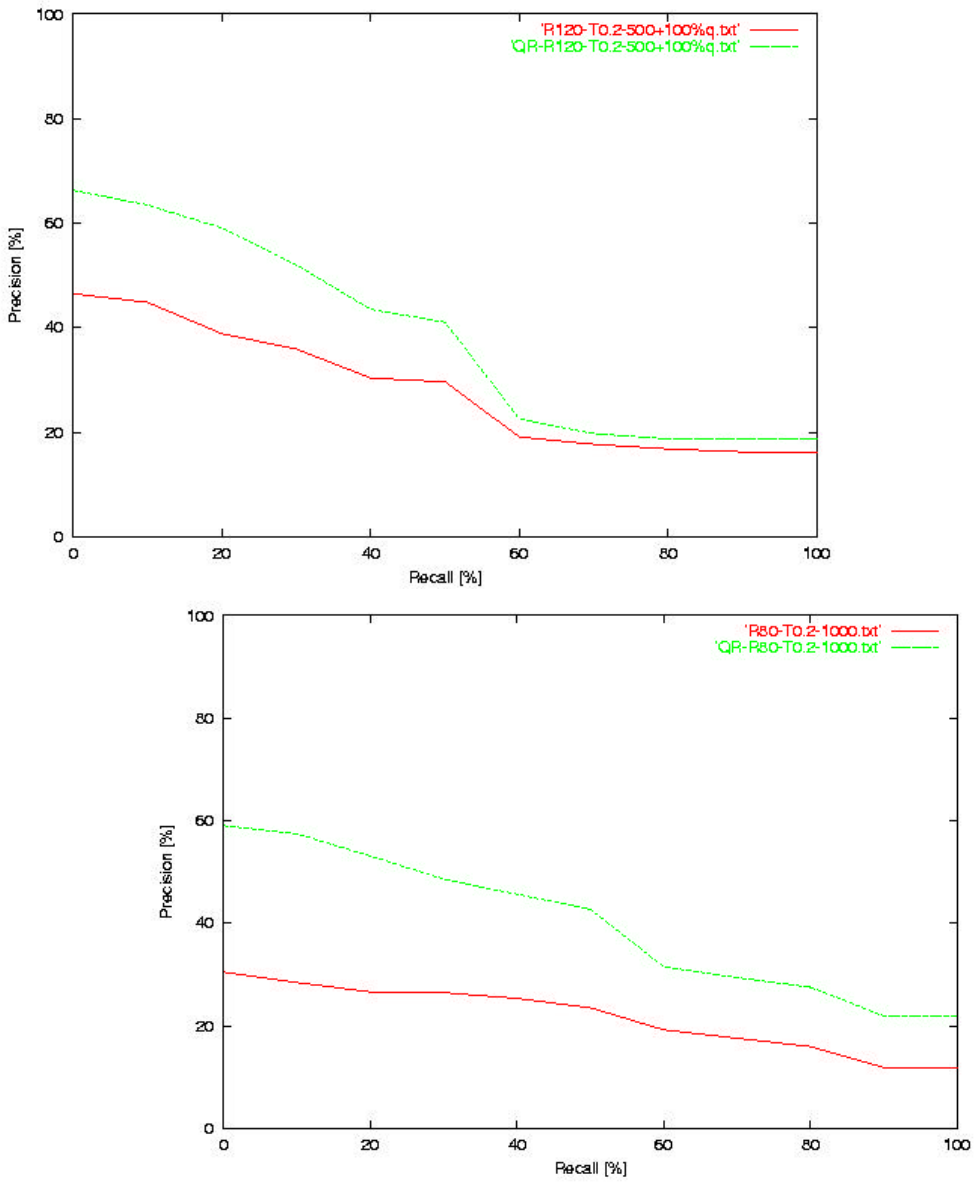


Figure 15 The recall-precision curves of original query and refined query. The top curve shows the experiment results of the refined query. Text collection is the TIME. Similarity threshold is set to 0.2. The index terms are 500 most frequently appearing terms and all query terms for the top one, and 1000 most frequently appearing terms only for the bottom one.

Figure 15 shows the experimental results on the TIME text collection.

The user's relevance feedback can significantly enhance the performance of the retrieval systems. We tested using the user's relevance feedback to refine the user's queries on different test collections and different composition of index terms. In all cases, the relevance feedback improved the performance noticeably.

5.5 Query Expansion

In Section 2.1.3.2 and Section 3.4.3, we have discussed query expansion. We can use LSI as an online thesaurus to expand the original query automatically by adding the terms that are semantically closely related to the query terms. We performed these experiments to learn whether the query expansion can improve the quality of an LSI retrieval system. Our hypothesis was that our results would show a large improvement on the retrieval results when using query expansion.

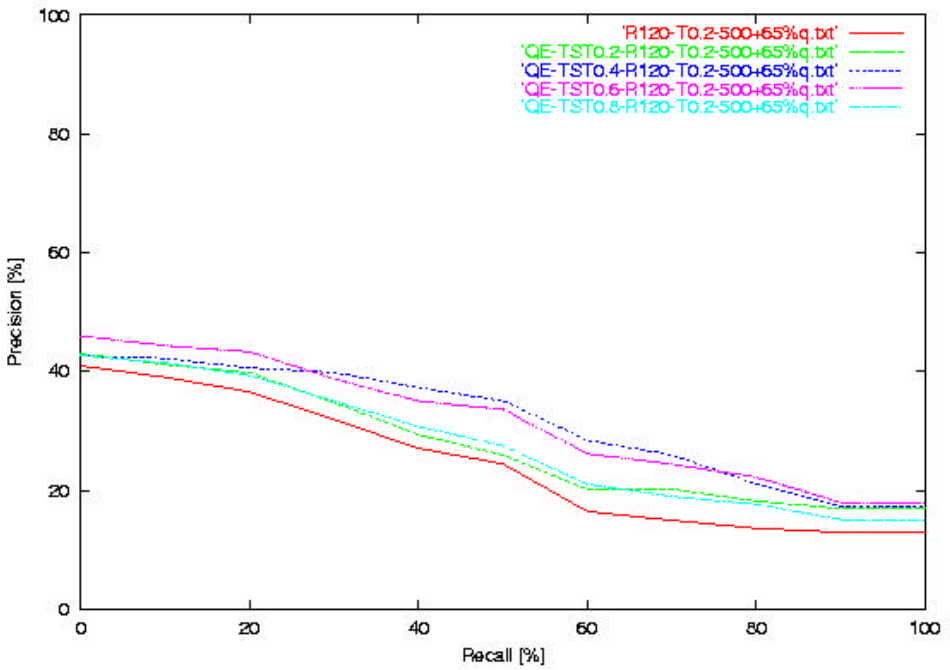


Figure 16: The recall-precision curves of the original query and expanded queries. Text collection is the CRAN. Rank is 120. Similarity threshold is set to 0.2 and the index terms are 500 most frequently appearing terms and 65% query terms. TST means term similarity threshold.

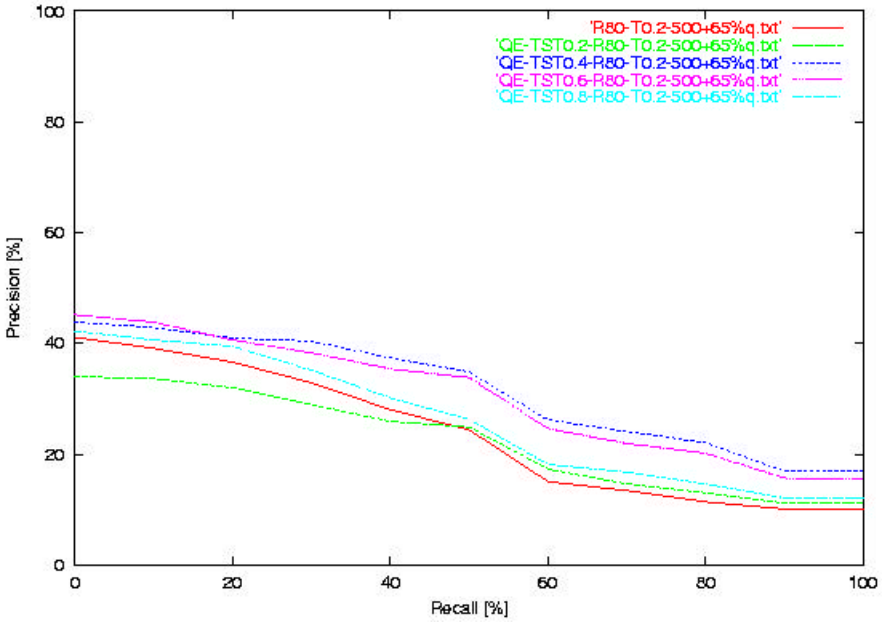


Figure 17 The recall-precision curves of the original query and expanded queries. Text collection is the TIME. Rank is 80. Similarity threshold is set to 0.2 and the index terms are 500 most frequently appearing terms and 65% query terms. TST means term similarity threshold

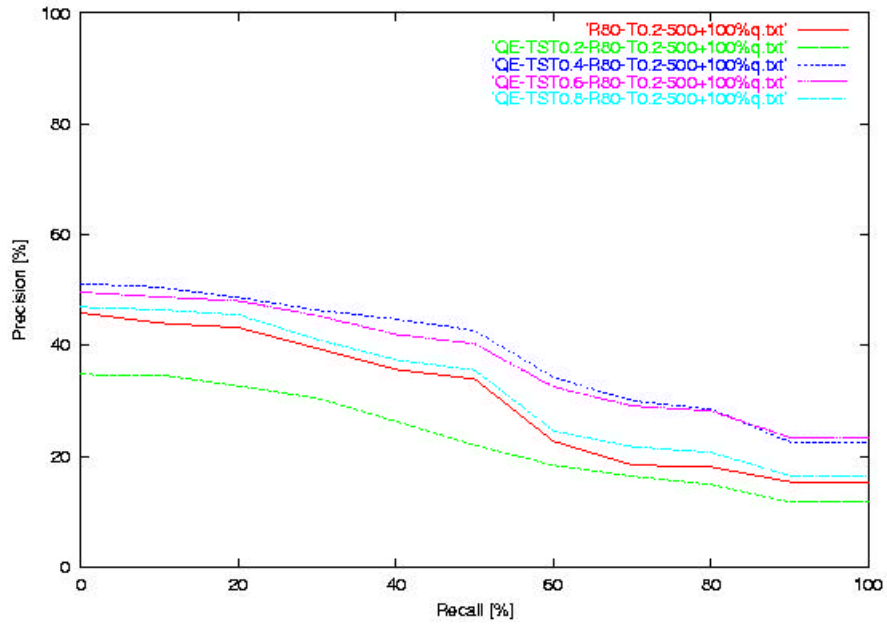


Figure 18 The recall-precision curves of the original query and expanded queries. Text collection is the TIME. Similarity threshold is set to 0.2 and the index terms are 500 most frequently appearing terms and all query terms. TST means term similarity threshold

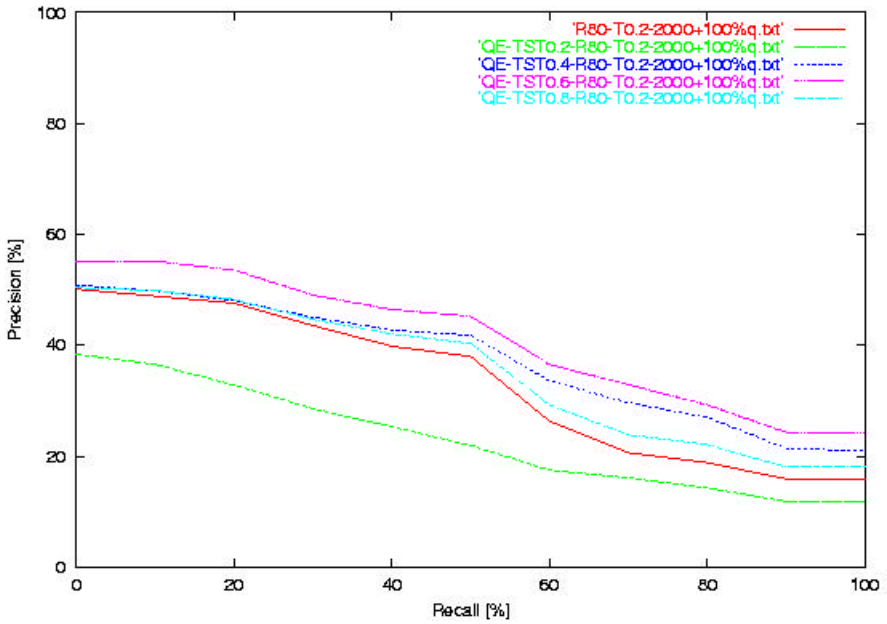


Figure 19 The recall-precision curves of the original query and expanded queries. Text collection is the CFC. Similarity threshold is set to 0.2 and the index terms are 2000 most frequently appearing terms and 100% query terms. TST means term similarity threshold

Figure 16, 17, 18 and 19 reveal that when the similarity threshold is set to 0.6 or 0.4, the expanded query yields the best results, which are much better than those from the original query. When the similarity threshold is too high or too low, the retrieval result is not as good as those when the similarity threshold is appropriately set, but is still better than the result of the original query. The reason is that when the similarity threshold is too high, few terms can be expanded to the query, and therefore, the performance cannot be improved much. On the other hand, if the similarity threshold is too low, some irrelevant terms might be expanded to the query, and the performance may decrease. We did experiments on different text collections of different settings, and the experiment results show that if an appropriate threshold is set, the automatic query expansion can always enhance the performance significantly.

5.6 Folding-in method

In Section 3.3.4, we discussed the folding-in method. Folding-in is very inexpensive in time and space, but the new vectors do not affect the existing vectors. Therefore, after the

folding-in, the updated SVD is not an exact representation of the updated text collection and can possibly result in a worse performance. We performed these experiments in order to learn how good the retrieval results could be after folding-in new documents into the original document collection. We were expecting that our results would show that the retrieval results would be acceptable after fold-in new documents.

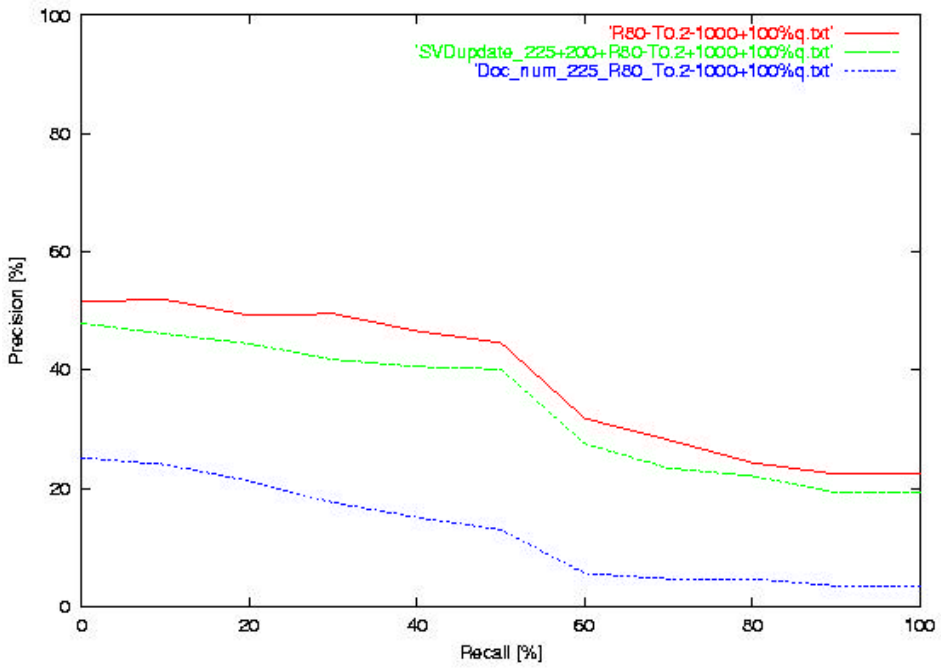


Figure 20: The recall-precision curves of original text collection and updated text collection after folding-in. Text collection is TIME. Similarity threshold is set to 0.2 and the index terms are 1000 most frequently appearing terms and all query term.

In Section 3.3.4, we discussed the folding-in method. Folding-in is very inexpensive in time and space, but the new vectors do not affect the existing vectors. Therefore, after the folding-in, the updated SVD is not an exact representation of the updated text collection and can possibly result in a worse performance.

We did the updated experiments by using folding-in method. Figure 20 shows the folding-in results on TIME. There are 425 documents in the TIME text collection. We first used LSI to index only 225 of them. Because some documents were not indexed and

could not be retrieved, the performance was very poor, as the bottom curve indicates. Then we used the folding-in method to append the remaining 200 documents into the system. The middle curve shows the retrieval results after the folding-in, which enhances the performance significantly. The top curve indicates the performance when all 425 documents are indexed by LSI. The experiment results show that the retrieval result of folding-in is almost as good as that achieved by SVD-recomputing even when about half of the documents are appended to the text collection. For a relatively constant document collection, in which, for example, less than 20% of documents are added in, the result of the folding-in method should be indistinguishable from the results of using LSI to index all the documents.

6 Conclusions

Latent Semantic Indexing is a conceptual indexing technique that has been used to overcome the problems of lexical matching. The LSI model uses statistical technique singular value decomposition (SVD) to reveal the “latent” semantic structure and eliminate much of variability of the word choice.

In this essay, we implemented a LSI information retrieval system. The experimental results show that the system can retrieve the documents effectively. We did experiments using different settings and different performance enhancing techniques to improve the results. The experimental results demonstrate that relevance feedback and query-expansion techniques can improve the retrieval effectiveness significantly. We also used the folding-in method to append new documents and new index terms to the collection to save the time and effort required by frequent SVD recomputing.

References:

[BDL95] M. Berry, S. Dumais and T. Letsche Computational Methods for Intelligent Information Access *Proceedings of the 1995 conference on Supercomputing*,1995

[BDO95] M. Berry, S. Dumais, and G. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4): 573-595, 1995.

[BR99]R. Baeza-Yates and B. Ribeiro-*Neto Modern Information Retrieval*. Addison-Wesley, 1999. 6

[Brown_Corpus] http://www.edict.com.hk/lexiconindex/frequencylists/words2000.htm

[CCW95]W. B. Croft, R. Cook, and D. Wilder. 1995. "Providing Government Information on the Internet: Experiences with THOMAS". *Proceedings of Digital Libraries '95*.pp 19-24. 1995

[Croft86] W.B. Croft User-specified domain knowledge for document retrieval *Proceedings of 1986 ACM conference on Research and development in information retrieval* **,** pp 201-206, 1986

[DDF+90] S. Deerwester, S.Dumais, G.Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6):391-407, 1990.

[Dum91] S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments*, Computers, 23(2):229-236, 1991

[Dum95a] Dumais, S. Using LSI for information filtering: TREC-3 experiments. *In Proc. of the Third Text Retrieval Conference (TREC-3), National Institude of Standards and Technology*, 1995

[Dum95b] Dumais, S. Latent Semantic Indexing(LSI): TREC-3 report. In Proceeding of TREC 3 conference, 1995.

[FBY92] W.B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[GL89] G. Golub and C.Van Loan. Matrix Computations. Johns-Hopkins, Baltimore, Maryland, second edition, 1989.

[KKC94] O. Kwon, M. Kim and K. Choi Query Expansion Using Domain-Adapted, Weighted Thesaurus in an Extended Boolean Model, *CIKM94* pp. 140-146 1994

[Miy90] S. Miyamoto, Information Retrieval based on Fuzzy Association. Fuzzy Sets and Systems, pp. 191-205

[O'Brien94] Gavin W. O'Brien Information Management Tools for Updating an SVD-Encoded Indexing Scheme. Master thesis. Univeristy of Tennessee, Knoxville. 1994

[Oga91] Y. Ogawa et al, A Fuzzy Document Retrieval System Using the Keyword Connection Matrix and a Learning Method. *Fuzzy Sets and Systems.*

[Porter80] Porter, M.F. 1980, An algorithm for suffix stripping. *Program*, 14(3) 130-137.

[PW91] H.J. Peat, and P. Willett The limitations of term co-occurrence data for query expansion in document retrieval system. *Journal of the ASIS 42(5)*, pp. 378-383, 1991

[QF93] Y. Qiu and H.P. Frei Concept Based Query Expansion *ACM-SIGIR'93* pp 160-169, 1993

[Rainbow] http://www-2.cs.cmu.edu/~mccallum/bow/rainbow/

[Rocchio71] J.J. Rocchio. Relevance feedback in information retrieval. *In the SMART Retrieval System-Experiments in Automatic Document Processing*, pp. 313-323, 1971.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4): 288-297, 1990.

[SP99] A. Singhal, F.Pereira Document Expansion for Speech Retrieval *Proceedings on the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* pp 34-41, 1999

[TG97] L. Tauscher, S. Greenberg "How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems ". *Int. J. Human-Computer Studies*, 47, 97-137.

[Voorhees94]E. M. Voorhees. Query Expansion using Lexical-Semantic Relations. In Proceedings of the 94 Annual International CMSIGIR conference on Research and *Development in Information Retrieval*, pp. 61-69. 1994

[XC96] J. Xu and W. B. Croft. Query expansion using local and global document analysis. *In Proc.ACM SIGIR,* 1996.