Ranking Unstructured Documents in IR

(A Comparative Study of Vector Space Model and Latent Semantic Indexing Model)

A Dissertation

Submitted to
Central Department of Computer Science and Information Technology,
Institute of Science and Technology
Tribhuvan University

In Partial Fulfillment of the Requirements for the degree of
Master of Science in Computer Science and Information Technology

By
Jamir Rana
December 2012

Date:………………..

Recommendation

I hereby recommend that the dissertation prepared under my supervision by **Mr. Jamir Rana,** entitled "**Ranking Unstructured Documents in IR (A Comparative Study of Vector Space Model and Latent Semantic Indexing Model)**" be accepted as fulfilling in part requirements for the degree of Masters of Science. In my best knowledge this is an original work in computer science.

Prof. Dr. Shashidhar Ram Joshi

Department of Computer and Electronic Engineering

Institute of Engineering, Pulchowk, Nepal

(Supervisor)

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

We certify that we have read this dissertation work and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Master of Science in Computer Science and Information Technology.

Evaluation Committee

Asst. Prof. Nawaraj Poudel
Central Department of Computer
Science and Information Technology
Tribhuvan University
(Act. Head)

Prof. Dr. Shashidhar Ram Joshi
Department of Computer and
Electronics Engineering
Institute of Engineering
Pulchowk, Nepal
(Supervisor)

(External Examiner)

(Internal Examiner)

Date: ------------------------

ABSTRACT

For thousands of years people have realized the importance of archiving and finding information. With the advent of computers, it became possible to store large amounts of information; and finding useful information from such collections became a necessity. The field of Information Retrieval (IR) was born in the 1950s out of this necessity. Over the last fifty years, the field has matured considerably. Several IR systems are used on an everyday basis by a wide variety of users. The goal of information retrieval (IR) is to provide users with those documents that will satisfy their information need.

Various Models of Information retrieved have been implemented like Boolean Model, Vector Space Model, Probabilistic Model and so on, among these models Vector Space Model (VSM) and Latent Semantic Indexing Model (LSI) are also promising models being used till date. The main concern of the study is to rank the documents and find out whether LSI Model overcomes the problems of VSM when the problems are attached with synonyms and polysemys while ranking documents.

The implemented features of these models like how to represent documents and query as vectors in $R^{|v|}$, term-document matrix, term-weighting, cosine similarity, SVD decomposition, dimensionality reduction and its effect in results of LSI have been presented.

Precision and recall have been implemented to know the effectiveness of the system. Conclusions have been drawn and future recommendation has been provided for better improvement.

ACKNOWLEDGEMENT

It is my great opportunity to complete this thesis under the supervision of Prof. **Dr. Shashidhar Ram Joshi**, Department of Computer and Electronics Engineering, Institute of Engineering Pulchowk, Nepal. I would like to express my profound gratitude to my supervisor for his generous encouragements and undertaking of the supervision of my entire research work. This form of outcome is due to his continuous encouragement, helpful suggestions and comments.

With this regard, I wish to extend my sincere appreciation to respected Head of Department of Central Department of Computer Science and Information Technology, **Assoc. Prof. Dr. Tanka Nath Dhamala** for his kind help.

I am also grateful to all staffs of Central Department of Computer Science and Information Technology, Tribhuvan University and all other staffs of Central Library of Tribhuvan University.

Lastly, let me offer my profound gratitude to my parents, family members and friends who inspired and helped me to uplift me at every steps of my life. This study is an outcome of their heartily blessing.

Jamir Rana

TABLE OF CONTENTS

Page No.

CHAPTER-II: REVIEW OF LITERATURE

CHAPTER-III: RESEARCH METHODOLOGY

CHAPTER-IV: IMPLEMENTATION & EVALUATION

LIST OF FIGURES

Figure No. Title **Page No.**

LIST OF TABLES

ABBREVIATIONS

IR	:	Information Retrieval.
LSI	:	Latent Semantic Indexing.
PLSI	:	Probabilistic Latent Semantic Indexing.
SVD	:	Singular Value Decomposition
VSM	:	Vector Space Model.

CHAPTER-I

INTRODUCTION

1.1 Background Information

For thousands of years people have realized the importance of archiving and finding information. With the advent of computers, it became possible to store large amounts of information; and finding useful information from such collections became a necessity. The field of Information Retrieval (IR) was born in the 1950s out of this necessity. Over the last fifty years, the field has matured considerably. Several IR systems are used on an everyday basis by a wide variety of users. The goal of information retrieval (IR) is to provide users with those documents that will satisfy their information need.

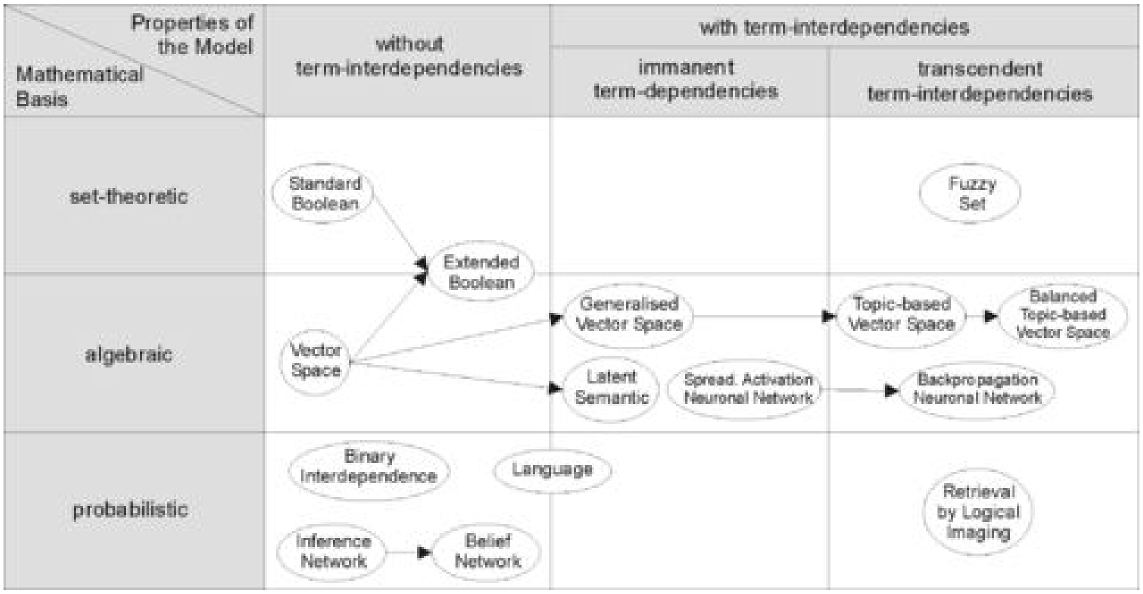
“What is information retrieval?” Information retrieval (IR) is the area of study concerned with searching any kind of relevant information like web-pages, news events, answers, images etc. According to [1], The meaning of the term information retrieval can be very broad. However academic field of study, information retrieval might be defined as the finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections, (usually stored on computer).

The need for the organization of data is a must for a quick and efficient retrieval of information. A robust means for organization of data in any organization has been the use of databases. Databases like the relational, object-oriented or object-relational databases, all have well structured format to keep data. Not all information that an organization generates is kept or can be kept in databases. Information are stored in the huge amount in form of unstructured or semi-structured documents. Organizing these documents into meaningful groups is a typical sub-problem of Information Retrieval, in which there is need to learn about the general content of data, Cutting, D. et al. [2].

1.2 IR Models

What is a model? There are two good reasons for having models of information retrieval. The first is that models guide, research and provide the means for academic discussion. The second reason is that models can serve as a blueprint to implement an actual retrieval system [3].

For the information retrieval to be [4] efficient, the documents are typically transformed into a suitable representation. There are several representations. The picture on the below illustrates the relationship of some common models. In the picture, the models are categorized according to two dimensions: the mathematical basis and the properties of the model.



Properties of the Model / Mathematical Basis	without term-interdependencies	with term-interdependencies: immanent term-dependencies	with term-interdependencies: transcendent term-interdependencies
set-theoretic	Standard Boolean Extended Boolean		Fuzzy Set
algebraic	Vector Space	Generalised Vector Space Latent Semantic Spread. Activation Neuronal Network	Topic-based Vector Space Balanced Topic-based Vector Space Backpropagation Neuronal Network
probabilistic	Binary Interdependence Language Inference Network Belief Network		Retrieval by Logical Imaging

Figure 1.1: Categorization of IR-Models [4]

a. First Dimension: Mathematical Basis

- Set-theoretic models represent documents as sets of words or phrases. Similarities are usually derived from set-theoretic operations on those sets. Common models are:
    - Standard Boolean model
    - Extended Boolean model
    - Fuzzy retrieval

- Algebraic models represent documents and queries usually as vectors, matrices, or tuples. The similarity of the query vector and document vector is represented as a scalar value.

  - Vector space model
  - Generalized vector space model
  - (Enhanced) Topic-based Vector Space Model
  - Extended Boolean model
  - Latent semantic indexing

- Probabilistic models treat the process of document retrieval as a probabilistic inference. Similarities are computed as probabilities that a document is relevant for a given query. Probabilistic theorems like the Bayes' theorem are often used in these models.

  - Binary Independence Model
  - Probabilistic relevance model on which is based the okapi (BM25) relevance function
  - Uncertain inference
  - Language models
  - Divergence-from-randomness model
  - Latent Dirichlet allocation

b. Second Dimension: Properties of the Model

- Models without term-interdependencies treat different terms/words as independent. This fact is usually represented in vector space models by the orthogonality assumption of term vectors or in probabilistic models by an independency assumption for term variables.
- Models with immanent term interdependencies allow a representation of interdependencies between terms. However the degree of the interdependency between two terms is defined by the model itself. It is usually directly or indirectly derived (e.g.

by dimensional reduction) from the co-occurrence of those terms in the whole set of documents.

- Models with transcendent term interdependencies allow a representation of interdependencies between terms, but they do not allege how the interdependency between two terms is defined. They relay an external source for the degree of interdependency between two terms. (For example a human or sophisticated algorithms.)

1.3 Information Retrieval Methods

Broadly speaking [5], information retrieval methods fall into two categories: They generally either view the retrieval problem as a document selection problem or as a document ranking problem.

1.3.1 Document Selection Method

In document selection methods, the query is regarded as specifying constraints for selecting relevant documents. A typical method of this category is the Boolean retrieval model, in which a document is represented by a set of keywords and a user provides a Boolean expression of keywords, such as “car or repair ships “, “tea or coffee” or “database systems but not Oracle”. The retrieval system would take such a Boolean query and return documents that satisfy the Boolean expression. Because of the difficulty in prescribing a user’s information need exactly with a Boolean query, the Boolean retrieval method generally only works well when the user knows a lot about the document collection and can formulate a good query in this way.

1.3.2. Document Ranking Method

Document ranking methods use the query to rank all documents in the order of relevance. For ordinary users and exploratory queries, these methods are more appropriate than document selection methods. Most modern information retrieval systems present a ranked list of documents in response to a user’s keyword query. There are many different ranking methods based on a large spectrum of mathematical foundations, including algebra, logic, probability and statistics. The common intuition behind all of these methods is that it may match the keywords in a query with those in the documents and score each document based on how well it matches the query. The goal is to approximate the degree of relevance of a document with a score computed based on information such as the frequency of words in the document and the whole collection.

1.4 Information Retrieval Process

There [3] are three basic processes an information retrieval system has to support: the representation of the content of the documents, the representation of the user's information need, and the comparison of the two representations. The processes are visualized in Figure 1.2. In the figure, rectangular boxes represent data and rounded boxes represent processes.

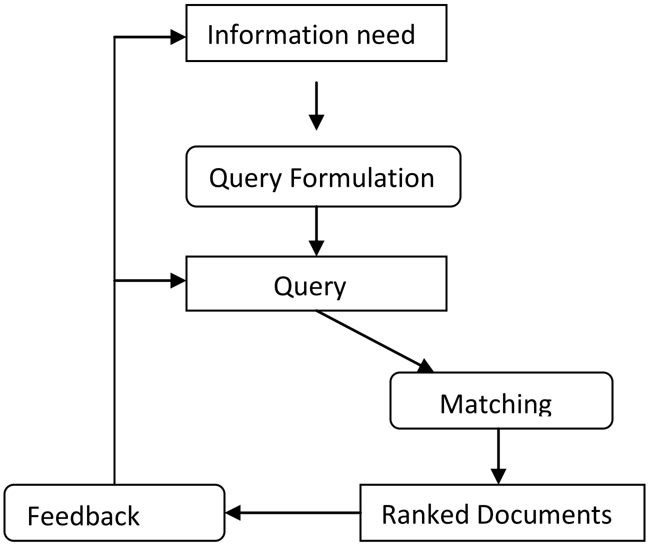


Figure 1.2: Information Retrieval Processes [3].

Users do not search just for fun; they have a need for information. The process of representing their information need is often referred to as the query formulation process. The resulting representation is the query. In a broad sense, query formulation might denote the complete interactive dialogue between system and user, leading not only to a suitable query but possibly also to the user better understanding his/her information need: This is denoted by the feedback process in Figure 1.2.

The comparison of the query against the document representations is called the matching process. The matching process usually results in a ranked list of documents. Users will walk down this document list in search of the information they need. Ranked retrieval will hopefully put the relevant documents towards the top of the ranked list, minimizing the time the user has to invest in reading the documents. Simple but effective ranking algorithms use the frequency distribution of terms over documents, but also statistics over other information, such as the number of hyperlinks that point to the document. Ranking algorithms based on statistical approaches easily halve the time the user has to spend on reading documents. The theory behind ranking algorithms is a crucial part of information retrieval.

In short, the main points of information retrieval process have been concluded as:

1. User has some information needs

2 Information Need → Query using Query Representation

3 Documents → Document Representation

4. IR system matches the two representations to determine the documents that satisfy user’s information needs.

1.5 Structured and Unstructured Data

Unstructured data is a generic level for describing any corporate information that is not in a database. Unstructured data can be textual or non textual. Textual unstructured data is generated in media like email messages, word documents, instant messages. Non-textual unstructured data is generated in media like JPEG images, MP3 audio and flash video files etc.

Even data that is unstructured such as free text of an image or a video clip typically has some associated information such as timestamp or author information that contributes partial structuring [6]. Data with partial structure is referred as semi-structured data.

Structured data on the other hand, which is really that, is organized in a structure so that it is identifiable. The most universal form of structured data is a database like SQL or Access. For example structure query language allows us to select specific piece of information based on columns and rows in a field. It might be looked for all the rows containing a particular data or zip code or name, this is structured data and it is organized and searchable by data type within the actual content.

1.6 An Overview of VSM & LSI Model

Various mathematical models have been proposed to represent Information Retrieval Systems and procedures as defined in **section 1.2**; the Boolean model, the probabilistic model, the vector space model, the latent semantic indexing model etc. Of the above models, this study focuses on the comparative study of the two information retrieval models viz. the vector space model and the latent semantic indexing model for ranking unstructured documents in IR. So, the brief descriptions of these two models have been presented below.

1.6.1 Vector Space Model

A vector space model is proposed by Gerard Salton and his colleagues. A vector space model is a mathematical structure formed by a collection of vectors. In the vector space model text is represented by a vector of terms.

The set of all n-tuples $(x_1, x_2,\ldots x_n)$ of n real numbers is known as n-space where n being a positive integer. It is denoted by R^n. Each n-tuples $(x_1, x_2,\ldots x_n)$ belongs to R^n is called a point or element in n-space. The real number $x_1,x_2\ldots\ldots\ldots x_n$ corresponding to a given point or element in R^n are called it's first, second, …n^{th} coordinate respectively.

All the documents are represented by point in a space of n dimension formed by n term coordinates. Each dimension represents element weight defined by tf-idf for one term. Queries are treated like documents. Documents are ranked by closeness to the query. Closeness is determined by a similarity score calculation.

The vector space retrieval model addresses the partial matching, supporting non-binary weights both for documents and queries and producing **continuous similarity** measures in [0,1]. The similarity measure is derived from the geometrical relationship of vectors in the v-dimensional space of document/query vectors. The vector space retrieval model is the standard retrieval technique used both on the Web and for classical text retrieval.

Key idea of Vector Space Retrieval is to represent both the document and the query by a weight vector in the supposing n=v, v-dimensional. Keyword space assigning non-binary weights, determine their distance in the v-dimensional keyword space. It is a today's standard text retrieval technique used by the web search engines [7]. The vector model is usually as good as the known ranking alternatives.

1.6.2 Latent Semantic Indexing Model

Latent Semantic Indexing is a technique that projects queries and documents into a space with "latent" semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms- as long as their terms are semantically similar in a sense. The latent semantic space that projects into has fewer dimensions than the original space (which has as many dimensions as terms). It is thus a method for dimensionality reduction. A dimensionality reduction technique takes a set of objects that exist in a high-dimensional space and represents them in a low-dimensional space, often in a two dimensional or three-dimensional space for the purpose of visualization.

Latent semantic indexing is the application of a particular mathematical technique, called Singular Value Decomposition or SVD, to a word-by-document matrix. SVD (and hence LSI) is a least-squares method. The projection into the latent semantic space is chosen such that the representations in the original space are changed as little as possible when measured by the sum of the squares of the differences. SVD takes a matrix A and represents it as $\hat{A}$ in a lower dimensional space such that the "distance" between the two matrices as measured by the 2-norm is minimized:

$$\Delta = \left\| A - \hat{A} \right\|_2 \text{..(i)}$$

The 2-norm for matrices is the equivalent of Euclidean distance for vectors. SVD project an n-dimensional space onto a k-dimensional space where t > > k. In this study (word-document matrices), t is the number of word types in the collection. The projection transforms a document's vector in n-dimensional word space into a vector in the k-dimensional reduced space.

Latent Semantic Indexing chooses the mapping that is optimal in the sense that it minimizes the distance Δ [8].

*“LSI tries to ove*rcome the problems of lexical matching by using statistically derived conceptual indices instead of individual words for retrieval. LSI assumes that there is some underlying or latent structure in word usage that is partially obscured by variability in word choice. A truncated Singular Value Decomposition (SVD) is used to estimate the structure in word usage across documents. Retrieval is then performed using a database of singular values and vectors obtained from the truncated SVD. Performance data shows that these statistically derived vectors are more robust indicators of meaning t*han individual terms.”* Berry et al. [9].

LSI incorporates VSM and also poses the negative vectors. Application of Latent Semantic Indexing with results can be found in Berry et al. [9] and Landauer et al. [10].

1.7 Focus of the Study

In the information retrieval, there are several models discussed earlier that assist in retrieving relevant information as desired by the users. These models have their own advantages and limitations and the desired information may satisfy the user’s need or not because sometimes the user’s query may match to the irrelevant documents accidentally and sometimes relevant documents may be missed due to no term in the document occurs in the query given by the users. The main focus of the study is to compare the two selected information retrieval models namely vector space model and latent semantic indexing model, find out their advantages and limitations and evaluate their comparative performance.

1.8 Statement of the Problem

As in [9,11], Typically, information is retrieved by literally matching terms in documents with those of a query. However, VSM can be inaccurate when they are used to match a user's query. Since there are usually many ways to express a given concept (synonymy), the literal terms in a user's query may not match those of a relevant document. In addition, most words have multiple meanings (polysemy), so terms in a user's query will literally match terms in irrelevant documents. A better approach would allow users to retrieve information on the basis of a conceptual topic or meaning of a document which is provided by LSI.

VSM would return documents only when documents share one or more terms with the query. But LSI can extract additional documents which are relevant to the query yet share no common terms because it is based on meaning rather than literal term usage.

Therefore, there are basically two problems that can be pointed out:

1. The problem of synonyms.

2. The problem of polysemys.

1. Problem of Synonyms

Example

Query term: "hum*or*"

D1: Joke clean humor and humor.

D2: Comedy circus.

D3: Comedy song funny.

D4: Humor story.

In the above example, ***"humor"*** has the synonyms (i.e. identical or similar meaning) as ***joke, comedy, funny.*** Will LSI be able to extra relevant documents D2 and D3 ? Because they contain synonyms for ***humor*** yet share no common term with the query.

2. Problem of Polysemys

Example

Query term: “*rock*”

D1: The rock marble.
D2: The rock music.
D3: Type of loud modern music with music guitars and music drums.

In the above example, the word **“*rock*”** means ***rock marble*** or ***rock music***. These two documents D1 and D2 have used **“rock”** as two different meanings. Also D3 gives the meaning of ***rock music*** though no ***rock*** word found. Will LSI be able to extra relevant documents D3 ? Because it gives the sense of ***rock music*** yet share no common term with the query.

1.9 Objectives of the Study

The study primarily focuses on the comparative study of two information retrieval models viz. Vector Space Model and Latent Semantic Model for retrieving information (documents) and ranking those unstructured documents. However, the specific objectives are as follows:

- To do comparative study of the Vector Space Model (VSM) and Latent Semantic Indexing (LSI) Model.
- To exhibit how these models are used for retrieving relevant documents and ranked them properly.
- To examine whether LSI Model overcomes the problems of VSM or not.

1.10 Limitations of the Study

The present study has some limitations. The study is presented to partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Information Technology. To do the comparative study of the two information retrieval models viz. Vector Space Model and Latent Semantic Indexing Model, few numbers and small size documents are taken as samples and only positive scores are taken in LSI to compare with VSM. These documents have been tested and comparatively analyzed the results.

1.11 Organization of the Study

The present study has adopted the following organization of the thesis, which is briefly described below.

Chapter-I; Introduction

This chapter includes the following components: Background Information, IR Models, Information Retrieval Methods, Information Retrieval Process, Structured and Unstructured Data, Overview of VSM and LSI Models, Focus of the study, Statement of the Problem, Objectives of the Study and Limitations of the Study.

Chapter-II; Review of Literature

The main components of this chapter are: VSM Vs. LSI Models, Components of VSM, Components of LSI Model, Geometrical Visualization, and Documents Ranking.

Chapter-III; Research Methodology

This chapter deals with Research Methodology adopted by the present study. In this chapter, the general implementation process in IR has been described and tools for the effectiveness measurements of the system (result) has also been described.

Chapter-IV; Implementation & Evaluation

The fourth chapter shows the picture of implementation of both the models under study. It also consists of the comparative study of the results of both models as well as their precision and recall have been evaluated and analyzed.

Chapter-V; Summary, Conclusion and Future Recommendation

It states summary, conclusion and future recommendation of the study. This chapter consists of the summary of the whole study and points out the conclusion of the analysis. It also provides future recommendation for further study which will be the fruitful for the new researcher in the field of IR.

CHAPTER-II

REVIEW OF LITERATURE

2.1 Vector Space Model Vs. Latent Semantic Indexing Model

The properties of vector space model are:

– Ranking of documents according to similarity value.

– Documents can be retrieved even if they don't contain some query keyword.

Despite its success the vector model suffers some problems. Unrelated documents may be retrieved simply because terms occur accidentally in it, and on the other hand related documents may be missed because no term in the document occurs in the query (consider synonyms, there exists a study that different people use the same keywords for expressing the same concepts only 20% of the time). Thus it would be an interesting idea to see whether the retrieval could be based on concepts rather than on terms, by mapping first terms to a "concept space" (and queries as well) and then establish the ranking with respect to similarity within the concept space. This idea is explored in the following.

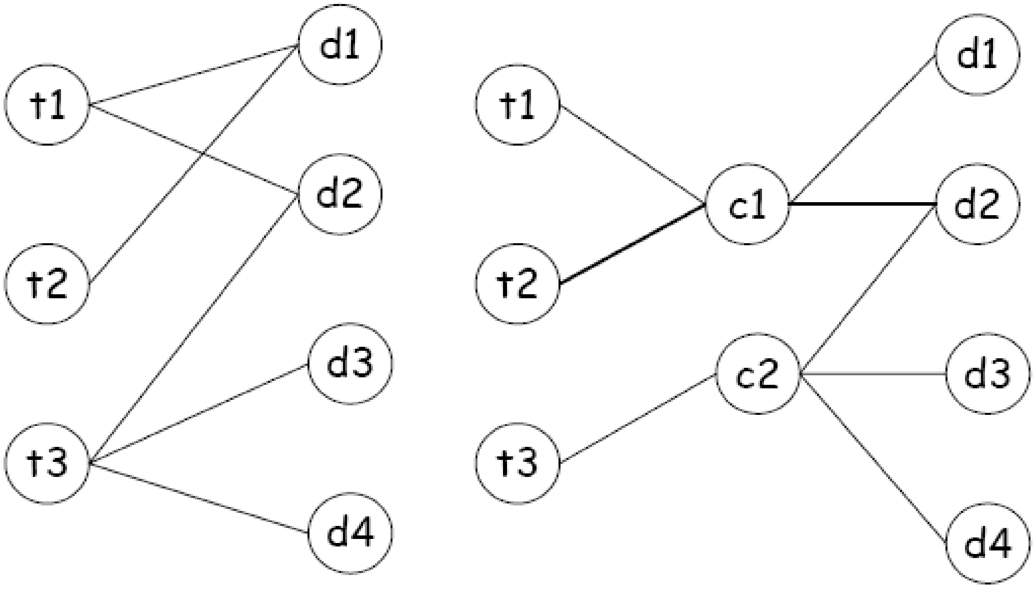


Figure 2.1: LSI Model:-Using Concepts for Retrieval [11]

This illustrates the approach: rather than directly relating documents and terms as in vector retrieval, there exists a middle layer into which both queries and documents map. The space of concepts can be of smaller dimension. For example, we could determine that the query t3 returns d2, d3, d4 in the answer set based on the observation that they relate to concept c2, without requiring that the document contains term d3. The question is, of how to obtain such a concept space. Much simpler, it would be to try to use mathematical properties of the term-document matrix, i.e. determine the concepts by matrix computation [11].

The differences between vector space model and Latent semantic indexing model can be summarized in the table below as:

IR Models	Partial Match	Full Match	No Match at all
Vector Space Model	Yes	Yes	No
LSI Model	Yes	Yes	Yes

Table 2.1: Comparison Table for Two IR Models [11].

2.2 Components of Vector Space Model

The vector space model procedure can be divided in to three stages. The first stage is the document indexing where content bearing terms are extracted from the document text. The second stage is the weighting of the indexed terms to enhance retrieval of document relevant to the user. The last stage ranks the document with respect to the query according to a similarity measure. The three stages are described briefly below:

1. Document Indexing

It is obvious that many of the words in a document do not describe the content, words like the, is. By using automatic document indexing those non significant words (function words) are removed from the document vector, so the document will only be represented by content bearing words [12]. This indexing can be based on term frequency, where terms that have both high and low frequency within a document are considered to be function words

2. Term-Document Matrix and Term Weighting

In VSM, a collection of d documents described by t terms can be represented as a t×d matrix A, commonly called term-document matrix. The column vectors are called document vector representing the documents in the collection and the row vectors are called term vectors representing the indexed terms from the documents. Each component of the document vector reflects a particular term which may or may not be in the document. The value of each component depends on the degree of relationship between its associated term and the respective document, commonly called term weighting. As the Vector Space Model requires that the relationship be described by a single numerical value, let a_{ij} represent the degree of relationship between term i and document j [13].

The use of various weighting schemes to discriminate one document from the other. In general this factor is called collection frequency document. The most used term weighting schemes are described briefly below.

a. Binary Weighting

This is the simplest term weighting method where if $a_{ij}=1$ means term i occurs in document j, $a_{ij}=0$ meaning otherwise. The binary weighting informs about the fact that a term is somehow related to a document but carries no information on the strength of the relationship.

b. Term Frequency Weighting

The simplest approach is to assign the weight to be equal to the number of occurrences of term t in document d. This weighting scheme is referred to as term frequency and is denoted $tf_{t,d}$ with the subscripts denoting the term and the document in order [1].

In this scheme $a_{ij} = tf_{ij}$ where tf_{ij} denotes how many times term i occurs in document j. This scheme is more informative than the binary weighting but, nevertheless it suffers from shortcoming as it focuses only on the local weight, neglecting the global weight.

c. Term Frequency-Inverse-Document Frequency Weighting (tf-idf)

This scheme overcomes the drawback of term frequency model by including the global weight. Thus have the term frequency tf (t,d), defined in the simplest case as the occurrence count of a term in a document. The inverse document frequency is a measure of the general importance of the term (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient). Hence an inverse document frequency factor is incorporated which diminishes the weight of terms that occur very frequently in the collection and increases the weight of terms that occur rarely.

$$\mathrm{idf}(t) = \log\left(\frac{D}{\mathrm{df_i}}\right) \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (1)$$

$$w = \mathrm{tf}\text{–}\,\mathrm{idf}(t, d) = \mathrm{tf}(t, d) \times \mathrm{idf}(t) \quad \dots\dots.\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots..(2)$$

where,

- tf_i = term frequency (term counts) or number of times a term i occurs in a document.
- df_i = document frequency or number of documents containing term i
- D = number of documents in the database.

To understand the global weight and the local weight following table can be used.

	Doc1	Doc2	Doc3	Doc4
Term 1	2	1		3
Term 2		2		1

Table 2.2: Term-document Matrix and Term Weighting.

The term1 occurs in total of 3 documents while term2 occurs in total of 2 documents. Now, the global weight of term1 = number of document(s) with the term1/total documents = log (4/3) = 0.124. Similarly, the global weight of term2 = log (4/2) = 0.30. Thus, the global weight is the overall importance of the term which decreases as the number of document containing the term increases. The tf-idf scheme aims at balancing the local and the global term occurrences in the documents.

d. Document Length Normalization

This is also one of the component of weighting scheme is the normalization factor, which is used to correct inconsistency in the document lengths. It is useful to normalize the document vectors so that documents are retrieved independent of their lengths [14]. The most popular normalization factor is Cosine normalization, given by

$$\frac{1}{\sqrt{\sum_{i=0}^{v}\left(G_i l_{ij}\right)^2}} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (ii)$$

Which divides by the magnitude of the weighted document vector, therefore forcing the magnitude of the weighted document vectors to be one.

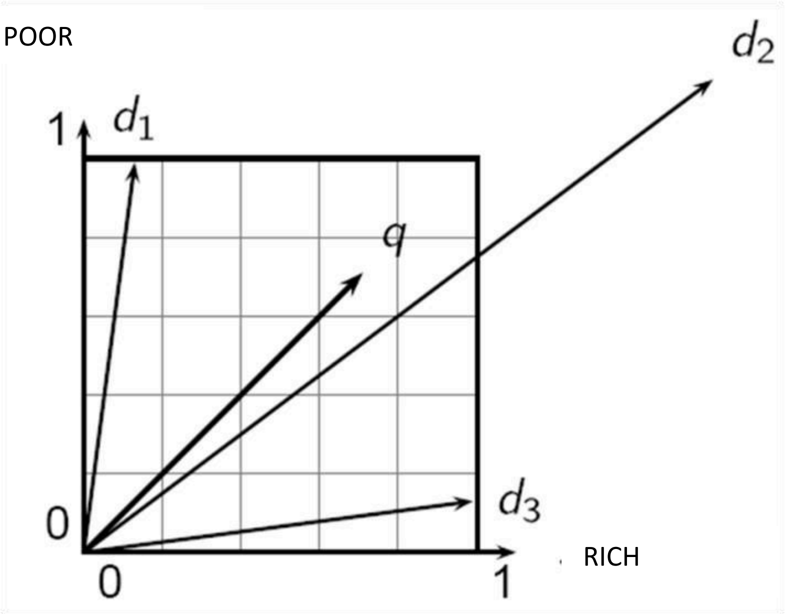


Figure 2.2: An Inconsistent Vector Space before Normalization [14].

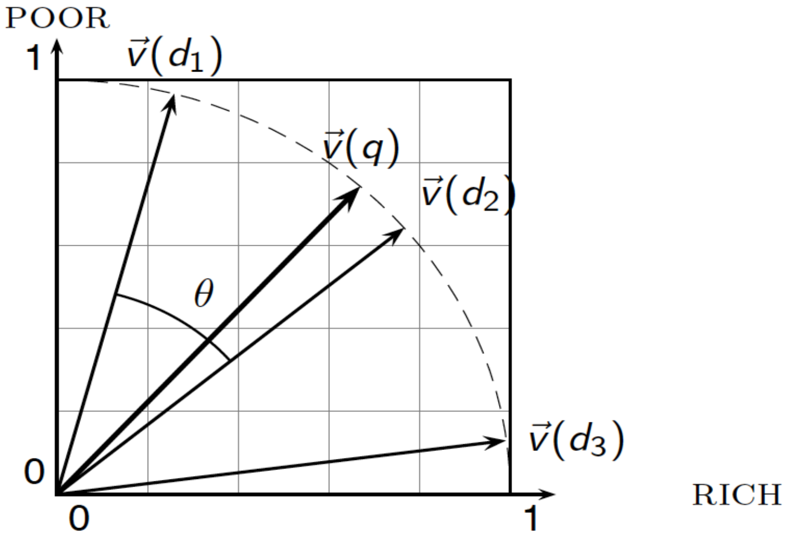


Figure 2.3: A Consistent Vector Space after Normalization [14].

3. Similarity Coefficients

The similarity in vector space models is determined by using associative coefficients based on the inner product of the document vector and query vector, where word overlap indicates similarity. The inner product is usually normalized. The most popular similarity measure is the cosine coefficient, which measures the angle between the document vector and the query vector[15].

Cosine Similarity score measures the similarity between two document vectors. The similarity between the two vectors is defined by the angle between them. If the two vectors are exactly similar then the angle between the two vectors is zero and thus cosine equal to 1 representing the perfect match. Similarly, if the two vectors are perfectly dissimilar then the angle between the two vectors is perfectly a 90^o and thus cosine equal to zero representing the perfect dissimilar vectors [16].

$$\text{sim(Q,D)} = \text{cosine } \Theta = \frac{V(Q) \cdot V(D)}{|V(Q)|\,|V(D)|} \qquad \text{...(3)}$$

Where,

V(Q) = Query Vector.

V(D) = Document Vector

|V(Q)| = Magnitude of Query Vector.

|V(D)| = Magnitude of Document Vector.

Or

$$\mathrm{Sim}(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2}\sqrt{\sum_i w_{i,j}^2}} \quad \text{...(iii)}$$

where the sigma symbol means "the sum of", Q is a query, D is a document relevant to Q and w are weights.

Figure 2.4: The Cosine Similarity (Cosine Angle) between Query and Documents.

2.3 Components of Latent Semantic Indexing Model

1. SVD (Singular Value Decomposition)

In 1965 G. Golub and W. Kahan introduced Singular Value Decomposition (SVD) as a decomposition technique for calculating the singular values, pseudo-inverse and rank of a matrix [17]. In order to implement LSI, the SVD forms the foundation for LSI. The technique decomposes a matrix $\mathbf{A_{t\times d}}$ into three new matrices. SVD is performed on the matrix to determine patterns in the relationships between the terms and concepts contained in the text. It computes the term and document vector spaces by transforming the single term-frequency matrix, $A_{t\times d}$, into the product of three other matrices— a **t** by **n** term-concept vector matrix U, an **n** by **n** singular values matrix S, and a **d** by **n** concept-document vector matrix, V, which satisfy the following relations [18].

$$\mathbf{A_{t\times d} = U_{t\times n} S_{n\times n} (V_{nxd})^T} \quad \text{..(4)}$$

Where **t** is the number of terms, **d** is the number of documents, **n = min(t,d)** , **U** and **V** have orthonormal columns, i.e. $\mathbf{U^T U = V^T V = I}$, **rank(A) = r** , **S=diag** $\mathbf{(S_{1,1}, S_{2,2}, \ldots\ldots\ldots\ldots\ldots S_{r,r})}$ and $\mathbf{S_{1,1} \geq S_{2,2} \geq, \ldots\ldots\ldots S_{r,r} > 0, S_{i,j} = 0}$ where **i≠j.**

Also,

U = is a matrix whose columns are the eigenvectors of the $\mathbf{AA^T}$ matrix. These are termed the left eigenvectors.

S = is a matrix whose diagonal elements are the singular values of **A**. This is a diagonal matrix, so its non-diagonal elements are zero by definition.

V is a matrix whose columns are the eigenvectors of the $\mathbf{A^TA}$ matrix. These are termed the right eigenvectors.

$\mathbf{V^T}$ is the transpose of **V**.

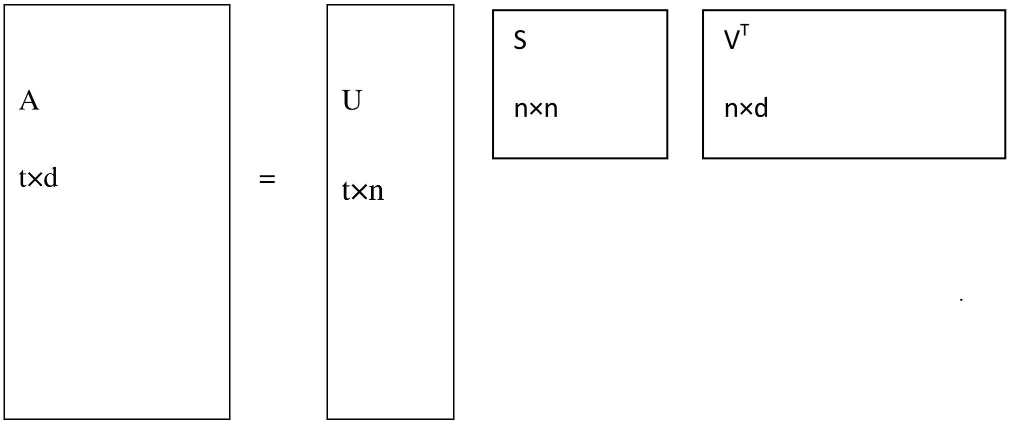


Figure 2.5: Mathematical Representation of the Matrix A [18].

The decomposition not only provides a direct method for computing the rank of a matrix, but exposes other equally interesting properties and features of matrices. Decomposing a matrix with **Equation 4** is often referred to as computing its "Full SVD".

Example of Full SVD

The full SVD of a matrix **A** can be computed by the following algorithm:

1. Compute its transpose $\mathbf{A^T}$ and $\mathbf{A^TA}$.
2. Determine the eigenvalues of $\mathbf{A^TA}$ and sort these in descending order, in the absolute sense. Square roots these to obtain the singular values of **A**.

3. Construct diagonal matrix **S** by placing singular values in descending order along its diagonal. Compute its inverse, $\mathbf{S}^{-1}$.
4. Use the ordered eigenvalues from step 2 and compute the eigenvectors of $\mathbf{A}^{\mathbf{T}}\mathbf{A}$. Place these eigenvectors along the columns of **V** and compute its transpose, $\mathbf{V}^{\mathbf{T}}$.
5. Compute **U** as $\mathbf{U} = \mathbf{AVS}^{-1}$. To complete the proof, compute the full SVD using $\mathbf{A} = \mathbf{USV}^{\mathbf{T}}$.

Let take a matrix A of order m×m and a transpose matrix $\mathbf{A}^{\mathbf{T}}$ is obtained by converting rows into columns and columns into rows. An example is given below.

$$\text{if } A = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \quad \text{then } A^T = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \qquad \text{..............................} \ (\text{iv})$$

Figure 2.6: A Matrix and its Transpose.

Using these matrices to construct two new matrices. This is done by pre-multiplying **A** by $\mathbf{A}^{\mathbf{T}}$ and $\mathbf{A}^{\mathbf{T}}$ by **A**.

$$\text{"left matrix"} \quad A\,A^T = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \qquad \text{"right matrix"} \quad A^T A = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

$$A\,A^T = \begin{bmatrix} 16 & 12 \\ 12 & 34 \end{bmatrix} \qquad A^T A = \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix} \qquad \text{..................} \ (\text{v})$$

Figure 2.7:"Left" and "Right" Matrices.

This matrix is also obtained by making the transformation:

$$(\mathbf{A}^{\mathbf{T}}\mathbf{A})^{\mathbf{T}} = \mathbf{A}^{\mathbf{T}}(\mathbf{A}^{\mathbf{T}})^{\mathbf{T}} = \mathbf{A}^{\mathbf{T}}\mathbf{A} \qquad \textbf{..(5)}$$

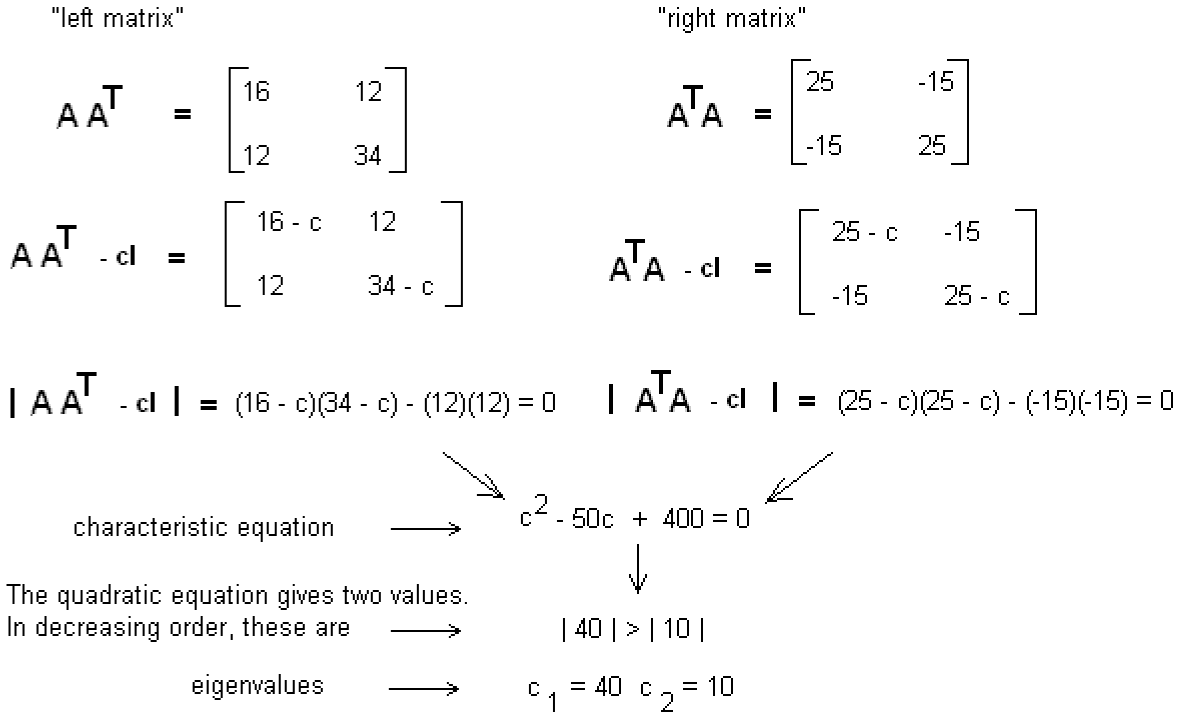
Now for the sake of consistency, this is to refer to their eigenvectors as the left and right eigenvectors. This left-right labeling system is trivial but very convenient. When plotted in the same space, these eigenvectors end at the left and right of each other. Examples have been provided.

|A - c*I| = 0 ....(6)

Here, **c** is a scalar matrix, **I** is an identity matrix and **A** has been transformed into a singular matrix. The problem of transforming a regular matrix into a singular matrix is referred to as the **eigenvalue problem**

Calculating eigenvalues for AA^T and A^TA

Note that $\mathbf{A^TA}$ and $\mathbf{AA^T}$ have the same characteristic equation and eigenvalues, too. Below showing eigenvalues are calculated from **equation 6**. This is not surprising -- considering the transformation given in **Equation 5**.



"left matrix"

$$AA^T = \begin{bmatrix} 16 & 12 \\ 12 & 34 \end{bmatrix}$$

$$AA^T - cI = \begin{bmatrix} 16 - c & 12 \\ 12 & 34 - c \end{bmatrix}$$

$$|\, AA^T - cI \,| = (16 - c)(34 - c) - (12)(12) = 0$$

"right matrix"

$$A^TA = \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$$

$$A^TA - cI = \begin{bmatrix} 25 - c & -15 \\ -15 & 25 - c \end{bmatrix}$$

$$|\, A^TA - cI \,| = (25 - c)(25 - c) - (-15)(-15) = 0$$

characteristic equation $\longrightarrow$ $c^2 - 50c + 400 = 0$

The quadratic equation gives two values.
In decreasing order, these are $\longrightarrow$ $|\,40\,| > |\,10\,|$

eigenvalues $\longrightarrow$ $c_1 = 40 \quad c_2 = 10$

Figure 2.8: Characteristic Equation and Eigenvalues for AA^T and A^TA.

Computing S of SVD

Algorithm for computing S

Given a matrix **A** of order m×n

1. Compute its transpose $\mathbf{A}^\mathbf{T}$ and $\mathbf{A}^\mathbf{T}\mathbf{A}$.
2. Determine the eigenvalues of $\mathbf{A}^\mathbf{T}\mathbf{A}$ and sort these in descending order, in the absolute sense. Square roots these to obtain the singular values of **A**.
3. Construct diagonal matrix **S** by placing singular values in descending order along its diagonal.

In the SVD technique, once eigenvalues from either the left or right matrix are obtained these are ordered in decreasing order (in the absolute sense) like this

$| c_1 | > | c_2 | > | c_3 | \ldots > | c_n |$

Then, taking their square roots yields

$s_1 > s_2 > s_3 \ldots > s_n$

Singular values can be zero or nonzero values. The number of nonzero singular values is defined as the **Rank of a Matrix**.

Retaking the example at hand, the singular values of **A** are

$s_1 = (c_1)^{1/2} = (40)^{1/2} = 6.32\ldots$
$s_2 = (c_2)^{1/2} = (10)^{1/2} = 3.16\ldots$

Thus, by virtue of Figure 2.8 it has been demonstrated that one can safely compute the singular values of **A** with either $\mathbf{A}^\mathbf{T}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^\mathbf{T}$.

Proceeding with the decomposition, now as it has been computed the singular values and this construct matrix **S** by placing values in descending order along its main diagonal, like this:

$$\mathbf{S} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} = \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix}$$

Figure 2.9: Singular Matrix of A and its Singular Values, Ordered in Descending Order.

'A' is a matrix of **Rank 2** since only two nonzero singular values are obtained, which turned out to be the only singular values of **A**. Since this is a diagonal matrix, by definition its non-diagonal elements are zero.

Computing "Right" Eigenvectors, V, and V^T of SVD

Eigenvectors for each eigenvalue, c_1 = 40 and c_2 = 10 are computed and converted eigenvectors to unit vectors. This is done by normalizing their lengths. Figure 2.10 illustrates these steps.

To compute the eigenvectors of $\mathbf{A^TA}$. This is done by solving

$$\mathbf{(A - c_iI)X_i = 0} \quad \text{..}\mathbf{(7)}$$

$$\text{Magnitude of a vector (L)} = \sqrt{X^2 + Y^2 + \cdots} \qquad \dots\dots\dots\dots\dots\dots.\ (8)$$

Note below that it has constructed **V** by preserving the order in which singular values were placed along the diagonal of **S**. That is, placed the largest eigenvector in the first column and the second eigenvector in the second column. These end paired with singular values placed along the diagonal of **S**. Preserving the order in which singular values, eigenvalues and eigenvectors are placed in their corresponding matrices is very important. Otherwise it would have end with the wrong SVD.

for $c_1 = 40$

$$A^TA - cI = \begin{bmatrix} 25 - 40 & -15 \\ -15 & 25 - 40 \end{bmatrix} = \begin{bmatrix} -15 & -15 \\ -15 & -15 \end{bmatrix}$$

$$(A^TA - cI)\ X_1 = 0$$

$$\begin{bmatrix} -15 & -15 \\ -15 & -15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-15x_1 + -15x_2 = 0$$

$$-15x_1 + -15x_2 = 0$$

Solving for x_2 for either equation: $x_2 = -x_1$

$$X_1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ -x_1 \end{bmatrix}$$

Dividing by its length,

$$L = \sqrt{x_1^2 + x_2^2} = x_1\sqrt{2}$$

$$X_1 = \begin{bmatrix} x_1 / L \\ -x_1 / L \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\sqrt{2}} \\ \dfrac{-1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0.7071 \\ -0.7071 \end{bmatrix}$$

for $c_2 = 10$

$$A^TA - cI = \begin{bmatrix} 25 - 10 & -15 \\ -15 & 25 - 10 \end{bmatrix} = \begin{bmatrix} 15 & -15 \\ -15 & 15 \end{bmatrix}$$

$$(A^TA - cI)\ X_2 = 0$$

$$\begin{bmatrix} 15 & -15 \\ -15 & 15 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$15x_1 + -15x_2 = 0$$

$$-15x_1 + 15x_2 = 0$$

Solving for x_2 for either equation: $x_2 = x_1$

$$X_2 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_1 \end{bmatrix}$$

Dividing by its length,

$$L = \sqrt{x_1^2 + x_2^2} = x_1\sqrt{2}$$

$$X_2 = \begin{bmatrix} x_1 / L \\ x_1 / L \end{bmatrix} = \begin{bmatrix} \dfrac{1}{\sqrt{2}} \\ \dfrac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

Figure 2.10: Right Eigenvectors of A^TA

Now to construct **V** by placing vectors along its columns and compute $\mathbf{V}^T$

$$V = \begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} 0.7071 & 0.7071 \\ -0.7071 & 0.7071 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$$

Figure 2.11: V and its Transpose V^T

Computing "Left" Eigenvectors and U of SVD

As **equation 4** is post-multiplying by **V** then equation **9** is arrived.

$$\mathbf{AV = USV^TV = US} \quad \text{......................................(9)}$$

To compute **U,** it can be reused eigenvalues and compute in exactly the same manner the eigenvectors of $\mathbf{AA^T}$. Once these are computed and place these along the columns of **U**. However, with large matrices this is time consuming. In practice, it is common to use the following shortcut. Post-multiply **Equation 9** by $\mathbf{S^{-1}}$ to obtain

$$\mathbf{AVS^{-1} = USS^{-1}} \quad \text{...(10)}$$

$$\mathbf{U = AVS^{-1}} \quad \text{...(11)}$$

and then compute **U**. Since **A** and **V** are known already, it is just need to invert **S**. Since **S** is a diagonal matrix, it must follow that

$$S^{-1} = \begin{bmatrix} \dfrac{1}{s_1} & 0 \\ 0 & \dfrac{1}{s_2} \end{bmatrix}$$

Figure 2.12: Inverted Singular Matrix.

Since $s_1 = 40^{1/2} = 6.3245$ and $s_2 = 10^{1/2} = 3.1622$ (expressed to four decimal places), then

$$\frac{1}{s_1} = \frac{1}{6.3245} = 0.1581 \qquad \frac{1}{s_2} = \frac{1}{3.1622} = 0.3162$$

$$U = AVS^{-1} = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} 0.7071 & 0.7071 \\ -0.7071 & 0.7071 \end{bmatrix} \begin{bmatrix} 0.1581 & 0 \\ 0 & 0.3162 \end{bmatrix}$$

$$U = AVS^{-1} = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} 0.1118 & 0.2236 \\ -0.1118 & 0.2236 \end{bmatrix}$$

$$U = AVS^{-1} = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix}$$

Figure 2.13: "Left" Eigenvectors and U.

The orthogonal nature of the **V** and **U** matrices is evident by inspecting their eigenvectors. This can be demonstrated by computing dot products between column vectors. All dot products are equal to zero. A visual inspection is also possible in this case. In Figure 2.14 eigenvectors have been plotted. Observe that they are all orthogonal and end to the right and left of each other, from here the reference to these as "right" and "left" eigenvectors

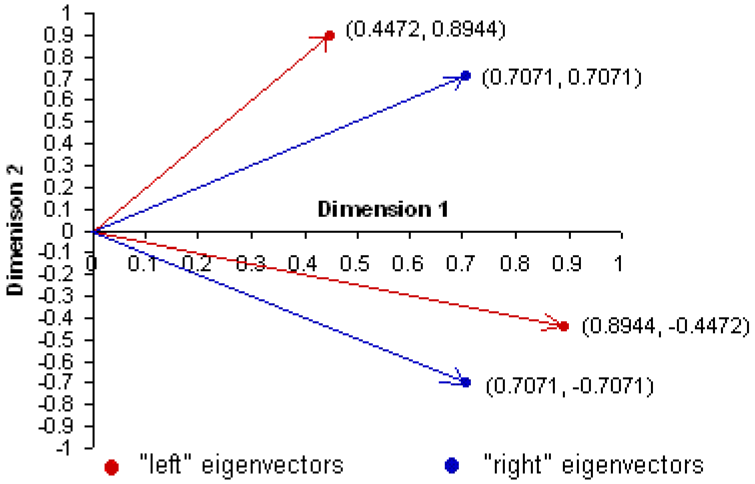


Figure 2.14: "Right" and "Left" Eigenvectors.

Computing the Full SVD

So, finally known **U,S, V** and $\mathbf{V}^{\mathbf{T}}$. To complete the proof, it is reconstructed **A** by computing its full SVD.

$$A = USV^T = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix} \begin{bmatrix} 6.3245 & 0 \\ 0 & 3.1622 \end{bmatrix} \begin{bmatrix} 0.7071 & -0.7071 \\ 0.7071 & 0.7071 \end{bmatrix}$$

$$A = USV^T = \begin{bmatrix} 0.4472 & 0.8944 \\ 0.8944 & -0.4472 \end{bmatrix} \begin{bmatrix} 4.4721 & -4.4721 \\ 2.2360 & 2.2360 \end{bmatrix}$$

$$A = USV^T = \begin{bmatrix} 3.9998 & 0 \\ 2.9999 & -4.9997 \end{bmatrix} \approx \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

Figure 2.15: Computing the Full SVD.

So as it can easily be seen, SVD is a straightforward matrix decomposition and reconstruction technique.

2 Dimensionality Reduction and LSI

When [18] computing the SVD of a matrix is desirable to reduce its dimensions by keeping its first k singular values. Since these are ordered in decreasing order along the diagonal of **S** and this ordering is preserved when constructing **U** and $\mathbf{V}^{\mathbf{T}}$, keeping the first k singular values is equivalent to keeping the first k rows of $\mathbf{V}^{\mathbf{T}}$ (gives documents vectors)**,** first k rows and columns of S and the first k columns of **U**. **Equation 4** reduces to

$$\mathbf{A^*_{t\times d} = U^*_{\ t\times k}\, S^*_{k\times k}\, (V_{k\times d})^{\ T*}} \ \ \mathbf{..............................(12)}$$

This process is termed **dimensionality reduction**, and **A*** is referred to as the **Rank k Approximation of A** or the "Reduced SVD" of **A**. The top k singular values are selected as a mean for developing a **"latent semantics"** representation of **A** that is now free from noisy dimensions. This "latent semantics" representation is a specific data structure in low-dimensional space in which documents, terms and queries are embedded and compared. This hidden or "latent" data structure is masked by noisy dimensions and becomes evident after the SVD.

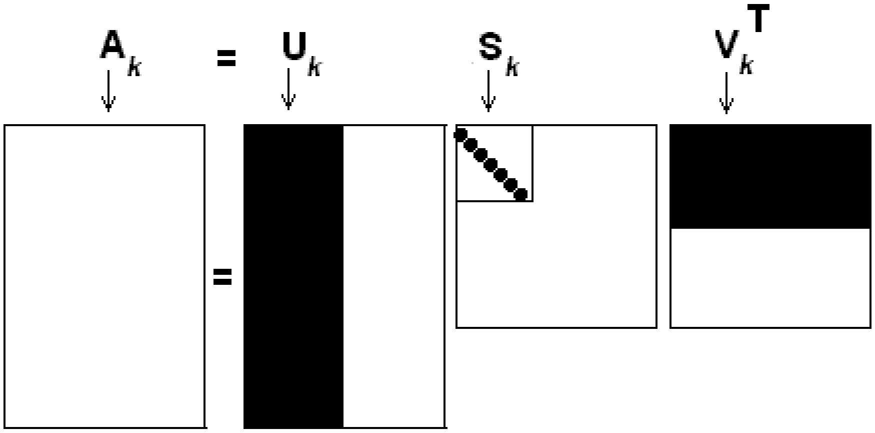


Figure 2.16: The Reduced SVD or Rank k-Approximation [18].

Where, U_k = term vectors
V_k^t = document vectors
S_k = Singular Values

3. Choosing the Number of Factors

Choosing [9] the number of dimensions (k) for A_k shown in Figure 2.16 is an interesting problem. While a reduction in k can remove much of the noise, keeping too few dimensions or factors may lose important information. Eventually performance must approach the level of performance attained by standard vector methods, since with k = n factors A_k will exactly reconstruct the original term by document matrix A. That LSI works well with a relatively small (compared to the number of unique terms) number of dimensions or factors k shows that these dimensions are, in fact, capturing a major portion of the meaningful structure.

4. Queries

For purposes of information retrieval, a user's query must be represented as a vector in k-dimensional space and compared to documents. A query (like a document) is a set of words. For example, the user query can be represented by

$$\hat{q} = q^T U_k S_k^{-1} \quad \text{..}(13)$$

Where q is simply the vector of words in the users query, multiplied by the appropriate term weights (a_{ij} = L(i; j) ×G(i); where a_{ij} denotes the frequency in which term i occurs in document j.). The sum of these k-dimensional terms vectors is reflected by the $q^T U_k$ term in **Equation 13**, and the right multiplication by S_k^{-1} differentially weights the separate dimensions. Thus, the query vector is located at the weighted sum of its constituent term vectors. The query vector can then be compared to all existing document vectors, and the documents ranked by their similarity (nearness) to the query. One common measure of similarity is the cosine between the query vector and document vector. Typically, the z closest documents or all documents exceeding some cosine threshold are returned to the user [9].

2.4 A Geometrical Visualization of LSI

The geometrical interpretation [18] of SVD has been described below. This is to use a simple analogy. Depicts an arrow rotating in two dimensions and describing a circle as in Figure 2.17.

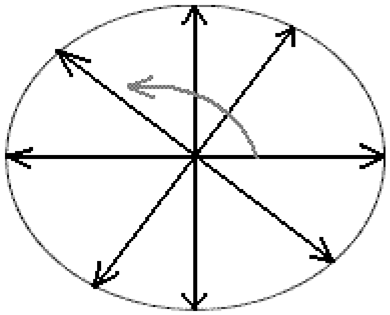


Figure 2.17: A Rotating Arrow of Fixed Length Describing a Circle.

Now suppose that when rotating this arrow, stretch and squeeze it in a variable manner and up to a maximum and minimum length. Instead of a circle the arrow now describes a two-dimensional ellipsoid.

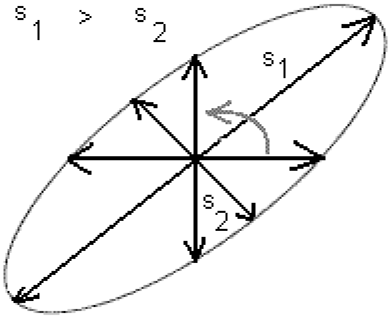


Figure 2.18: A Rotating Arrow of Variable Length Describing an Ellipsoid.

This is exactly what a matrix does to a vector.

When multiplying a vector **X** by a matrix **A** to create a new vector **AX = Y**, the matrix performs two operations on the vector: rotation (the vector changes coordinates) and scaling (the length of the vector changes). In terms of SVD, the maximum stretching and minimum squeezing is determined by the singular values of the matrix. These are values inherent, unique or "singular" to **A** that can be uncovered by the SVD algorithm. In the figure the effect of the two largest singular values, s_1 and s_2 has been labeled.

For now, the important thing to remember is this: singular values describe the extent by which the matrix distorts the original vector and, thus, can be used to highlight which dimension(s) is/are affected the most by the matrix. Evidently, the largest singular value has the greatest influence on the overall dimensionality of the ellipsoid. In geometric terms Figure 2.17 and Figure 2.18 show that multiplying a vector by a matrix is equivalent to transforming (mapping) a circle into an ellipsoid.

Now instead of a rotating vector depicts a set of m-dimensional vectors, all perpendiculars (orthogonal), with different lengths and defining an m-dimensional ellipsoid. The ellipsoid is embedded in an m-dimensional space, where m = k, with k being the number of nonzero singular values. If eliminate dimensions by keeping the three largest singular values, a three-dimensional ellipsoid is obtained. This is a Rank 3 Approximation.

Now suppose that compared with the two largest singular values the third one is small enough that ignores it. This is equivalent to removing one dimension. Geometrically, the 3-dimensional ellipsoid collapses into a 2-dimensional ellipsoid, and this obtain a Rank 2 Approximation. Obviously, if keeping only one singular value the ellipsoid collapses into a one-dimensional shape (a line). Figure 2.19 illustrates roughly the reduction.

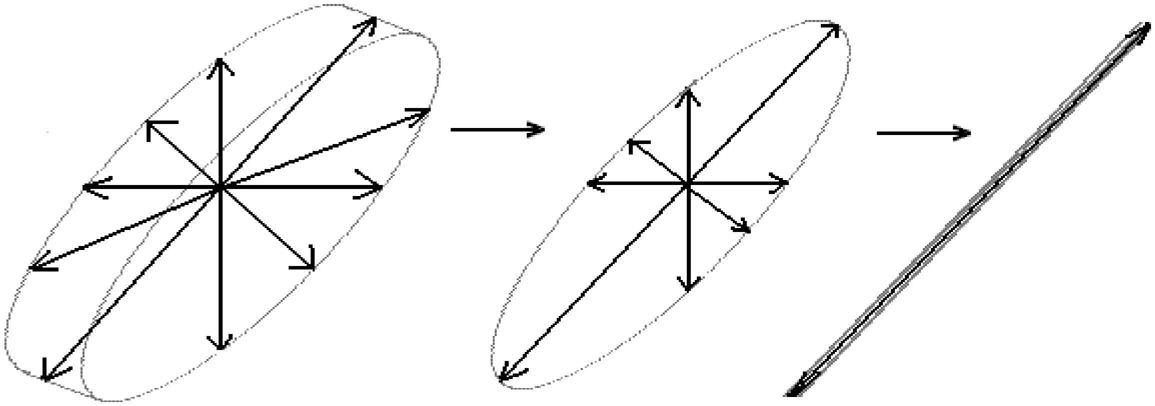


Figure 2.19: Dimensionality Reduction.

This is why it says that keeping the top k singular values is a dimensionality reduction process. The main idea is this: singular values can be used to highlight which dimensions are affected the most when a vector is multiplied by a matrix. Thus, the decomposition allows to determine which singular values can be retained, from here the expression "singular value decomposition".

In the case of LSI, SVD unveils the hidden or "latent" data structure, where terms, documents, and queries are embedded in the same low-dimensional and critical space. This is a significant departure from term vector models.

2.5 Documents Ranking

Both the models under study are used to rank the documents according to their scores. Finally sort the documents in descending order according to the similarity values as given below:

Example of ranking the documents

Rank 1: Doc 2

Rank 2: Doc 10

Rank 3: Doc 1

………………

………………

CHAPTER-III

Research Methodology

3.1 General Implementation Process in IR

1. Defining Data Storage

The process flow of implementation of IR models begins with maintaining a folder, which holds the documents collection. Each document is in the form of .txt file as new document is created the content of the folder increased by one. Each document contains a sentence of words that give some sense about some subject or topic.

2. Counting No. of Documents

From the document folders the number of the document contained is counted and returned value is assigned.

3. Tokenization

It is [19] the process of breaking a stream of text up into words, phrases, symbols or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining. During this phase, all remaining text is parsed, lowercased and all punctuation removed.

Example:

Input: Friends, Romans, Countrymen, lend me your ears;

Output:

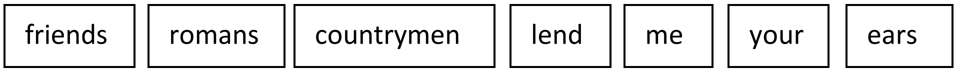


Figure 3.1: Tokenization

4. Filtration

Filtration refers [19] to the process of deciding which terms should be used to represent the documents so that these can be used for

1. Describing the document's content.
2. Discriminating the document from the other documents in the collection.

Frequently used terms cannot be used for this purpose for two reasons. First, the number of documents that are relevant to a query is likely to be a small proportion of the collection. A term that will be effective in separating the relevant documents from the non-relevant documents, then, is likely to be a term that appears in a small number of documents. This means that high frequency terms are poor discriminators. The second reason is that terms appearing in many contexts do not define a topic or sub-topic of a document. As **Ruthven and Lalmas** state that:

"The more documents in which a term appears (the more contexts in which it is used) then the less likely it is to be a content-bearing term. Consequently it is less likely that the term is one of those terms that contribute to the user's relevance assessment. Hence, terms that appear in many documents are less likely to be the ones used by a searcher to discriminate between relevant and non-relevant documents."

For these reasons, frequently used terms or **stopwords** are removed from text streams. This can be accomplished with a stopword library --a stop-list of terms to be removed. These lists can be either generic (applied to all collections) or specific (created for a given collection). Stoplist: contain stopwords, not to be used as index like prepositions, articles, pronouns, some adverbs and adjectives etc. The removal of stopwords usually improves IR effectiveness

5. Stemming

Stemming refers [19] to the process of reducing terms to their stems or root variant. Thus, "computer", "computing", "compute" is reduced to "comput" and "walks", "walking" and "walker" is reduced to "walk".

6. Weighting

Weighting [19] is the final stage in most IR indexing applications. Terms are weighted according to a given weighting model which may include local weighting, global weighting or both. If local weights are used, then term weights are normally expressed as term frequencies, tf. If global weights are used, the weight of a term is given by IDF values. The most common (and basic) weighting scheme is one in which local and global weights are used (weight of a term = tf*IDF). This is commonly referred to as tf*IDF weighting.

During **indexing** documents are prepared for use by an IR system. This means preparing the raw document collection into an easily accessible representation of documents. This transformation from a document text into a representation of text- is known as **indexing** the documents. So, document indexing involves (3) through (6).

7. Building an Inverted Index

Inverted index [20] (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches at a cost of increased processing when a document is added to the database. The inverted file is may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines. An index always maps back from terms to the parts of a document where they occur. The basic idea of an inverted index is shown in fig 3.2.

A dictionary of terms is kept. Then for each term, a list that records is associated in which documents the term occurs in. Each item in the list-which records that a term appeared in a document is conventionally called a posting. The list is then called a postings list (or), and all the postings lists taken together are referred to as the postings. The dictionary has been stored alphabetically and each postings list is sorted by document ID. Example of drawing an inverted index is shown below.

D1: new home sales top forecasts
D2: home sales rise july

D3: increase in home sales july
D4: july new home sales rise

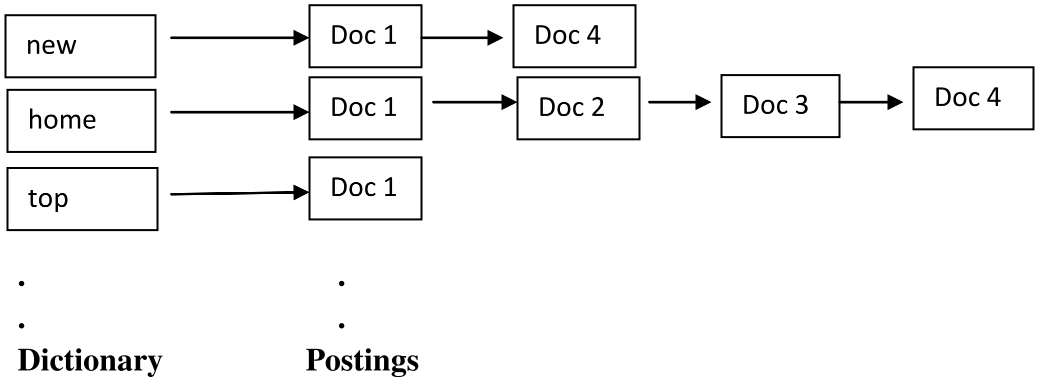


Figure 3.2: Two Parts of an Inverted Index

8. Insertion and Defining Cell Position of a Term-Document Matrix

```
human Di is 4

human in document 0
human in document 1
human in document 2
human in document 17

Frequency for human in doc is 1
0 0.
Frequency for human in doc is 2
0 1.
Frequency for human in doc is 1
0 2.
Frequency for human in doc is 1
0 17
```

Figure 3.3: Frequency of Terms in Each document and Cell Position in Term Document Matrix.

For the term **"human"**, it is contain in 4 documents, in documents ID 0, 1, 2 and 17 respectively. The term **"human"** has frequency 1 in document ID 0, frequency 2 in document ID 1, frequency 1 in document ID 2 and frequency 1 in document ID 17. For each occurrence of the term in document ID defines the cell position of the term-document matrix as 0 0, 0 1, 0 2 and 0 17 as shown in figure 3.3.

3.2 Basic Measures for Text Retrieval: Precision and Recall

"Suppose **[5] that a text retrieval system has just retrieved a number of documents for based on input in the form of a query. How can user asses how accurate or correct the** ***system was?"*** Let the set of documents relevant to a query be denoted as {Relevant}, and the set of documents retrieved be denoted as {Retrieved}. The set of documents that are both relevant and retrieved is denoted as {Relevant} ∩ {Retrieved}, as shown in the Venn diagram of figure 3.4. There are two basic measures for assessing the quality of text retrieval:

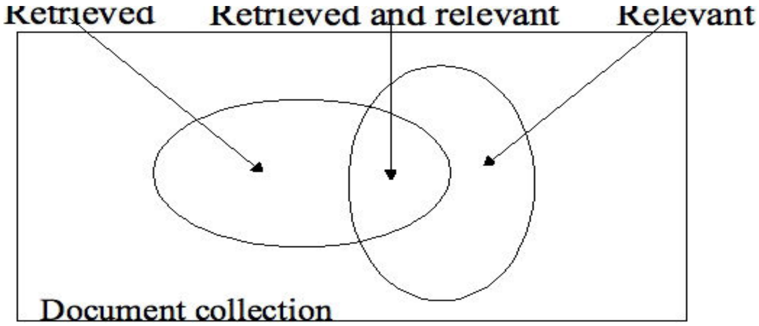


Figure 3.4: Relationship between Relevant and Retrieved Documents [5]

3.2.1 Precision

This is the percentage of retrieved documents that are in fact relevant to the query (i.e. "correct" responses). It is formally defined as

$$\text{Precision} = \frac{\{\text{ Relevant}\} \cap \{\text{ Retrieved}\}}{\{\text{Retrieved}\}}$$

$$= \frac{\#\text{ Relevant items retrieved}}{\#\text{retrieved items}} \quad \dots\dots\dots\dots\dots..(14)$$

3.2.2 Recall

This is the percentage of documents that are relevant to the query and were in fact, retrieved. It is formally defined as

$$\text{recall} = \frac{\{\text{Relevant}\} \cap \{\text{Retreived}\}}{\{\text{Relevant}\}} = \frac{\#\text{relevant items retrieved}}{\#\text{relevant items}} \ldots\ldots\ldots.\ldots\ldots\ldots(15)$$

CHAPTER-IV

Implementation & Evaluation

4.1 Algorithms

To implement both the IR models under study, the following algorithms have been presented below.

4.1.1 Algorithm for VSM

Basic Algorithm to Implementation the Vector Space Model for Unstructured Documents Ranking is as follow.

1. Construction of a term-document matrix ‘**A**’.

2. Defining weight for each element of the term-document matrix using a specific term weight scoring system. Mostly preferable weighting system is local and global weightings as:
   $a_{i\,j}= L_{ij}\times G_{ij}$
   where,
   L_{ij} = Local weight of a term i in the document j.
   G_{ij} = Global weight of a term in the entire collection.

3. Similarly defining query term/s matrix ‘**Q**’.

4. Normalized Similarity Cosine is computed between each document vector and a query vector for similarity comparison.

5. Finally sort the documents in descending order according to the similarity values.

End of algorithm.

4.1.2 Algorithm for LSI Model

Basic Algorithm to Implementation the Latent Semantic Indexing Model for Unstructured Documents Ranking is as follow.

1. Construction of a term-document matrix '**A**'.

2. Defining weight for each element of the term-document matrix using a specific term weight scoring system. Mostly preferable weighting system is local and global weightings as:

$a_{ij}=L_{ij}\times G_{ij}$
where,
L_{ij} = Local weight of a term i in the document j.
G_{ij} = Global weight of a term in the entire collection.

3. Perform Singular Value Decomposition (SVD) on '**A**'. This will result in the product of three matrices U, S and V^T respectively.

4. Reduced the matrices obtained from (3) to K-approximation.

5. Find new coordinates of query and document vectors in the reduced k-dimensional space.

6. Similarity Cosine is computed between each document vector and query vector in the reduced k-dimensional space.

7. Finally sort the documents in decreasing order of cosine similarity values.

End of Algorithm.

4.2 Implementation of VSM & LSI Model

4.2.1 A Demonstration of Full Matching to the Query Terms

List of Input Documents:

This study has taken altogether nine documents of two groups, five about Human Computer Interaction (HCI), and four about Mathematical Graph Theory, these two groups are conceptually disjoint. Indexing is done to these input documents as described in **section 3.1** and all the underlined words in Table 4.1 denote keywords which are used as referents to the documents.

Doc Id	**Input Documents**
D1	Human machine interface for ABC computer applications.
D2	A survey of user opinion of computer system response time.
D3	The EPS user interface management system.
D4	System and human system engineering testing of EPS.
D5	Relation of user perceived response time to error measurement.
D6	The generation of random, binary, ordered trees.
D7	The intersection graph of paths in trees.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D9	Graph minors: A survey.

Table 4.1: List of Input Documents with Underlined Keywords.

Experiment 1

Applying VSM

The term-document matrix with simple term- frequency (tf_i) weighting is presented below for the input data set mentioned in Table 4.1.

Terms	Q	D1	D2	D3	D4	D5	D6	D7	D8	D9
human	1	1	0	0	1	0	0	0	0	0
interface	0	1	0	1	0	0	0	0	0	0
computer	1	1	1	0	0	0	0	0	0	0
user	0	0	1	1	0	1	0	0	0	0
system	0	0	1	1	2	0	0	0	0	0
response	0	0	1	0	0	1	0	0	0	0
time	0	0	1	0	0	1	0	0	0	0
EPS	0	0	0	1	1	0	0	0	0	0
survey	0	0	1	0	0	0	0	0	0	1
tree	1	0	0	0	0	0	1	1	1	0
graph	1	0	0	0	0	0	0	1	1	1
minor	0	0	0	0	0	0	0	0	1	1

Table 4.2: Term-Document Matrix and Term Weighting (tf_i)

The term-document matrix with term-frequency-inverse-document frequency ($tf\text{-}idf_i$) weighting of the above Table 4.2 is shown below.

df_i	D/df_i	idf_i= log(D/df_i)	Q	D1	D2	D3	D4	D5	D6	D7	D8	D9
2	9\2	0.6532	0.6532	0.6532	0	0	0.6532	0	0	0	0	0
2	9\2	0.6532	0	0.6532	0	0.6532	0	0	0	0	0	0
2	9\2	0.6532	0.6532	0.6532	0.6532	0	0	0	0	0	0	0
3	9\3	0.4771	0	0	0.4771	0.4771	0	0.4771	0	0	0	0
3	9\3	0.4771	0	0	0.4771	0.4771	0.9542	0	0	0	0	0
2	9\2	0.6532	0	0	0.6532	0	0	0.6532	0	0	0	0
2	9\2	0.6532	0	0	0.6532	0	0	0.6532	0	0	0	0
2	9\2	0.6532	0	0	0	0.6532	0.6532	0	0	0	0	0
2	9\2	0.6532	0	0	0.6532	0	0	0	0	0	0	0.6532
3	9\3	0.4771	0.4771	0	0	0	0	0	0.4771	0.4771	0.4771	0
3	9\3	0.4771	0.4771	0	0	0	0	0	0	0.4771	0.4771	0.4771
2	9\2	0.6532	0	0	0	0	0	0	0	0	0.6532	0.6532

Table 4.3: Term-Document Matrix and Term Weighting ($w_i = tf_i \times idf_i$)

<u>**Query terms**</u>**: *“human computer* tree graph*”***

<u>**Query vector coordinates**</u>**:**

```
0.6532 0 0.6532 0 0 0 0 0 0 0.4771 0.4771 0
```

<u>**Output Generated**</u>

For the query ***“human computer tree graph”***, following output result is obtained using the input data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.6593	Yes
D2	0.2537	Yes
D3	0	Not Retrieved
D4	0.2808	Yes
D5	0	Not Retrieved
D6	0.4171	Yes
D7	0.5898	Yes
D8	0.4238	Yes
D9	0.1914	Yes

Table 4.4: Computing Cosine Scores and Marking Document Relevancy.

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.4 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D7	The intersection graph of paths in trees.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D6	The generation of random, binary, ordered trees.
D4	System and human system engineering testing of EPS.
D2	A survey of user opinion of computer system response time.
D9	Graph minors: A survey.

Table 4.5: Ranked Result of a Query where Non-Ranked Documents are Not Displayed.

Applying LSI Model

1. A_2 Approximation (i.e. 2D-Representation)

The LSI model has been applied for the same query as applied in VSM. The term-document matrix **"A"** (i.e. 12×9 matrix) is decomposed into the products of three separate matrices as defined and given by **equation 4**. The result is presented below:

Singular Value Decomposition (SVD)

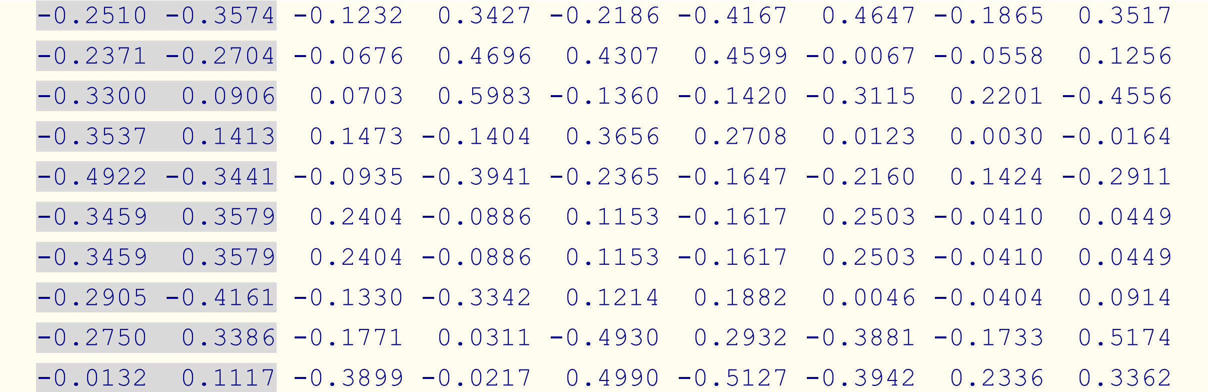
Query terms: ***"human computer*** **tree graph*"***

$$\mathbf{A_{t\times d} = U_{t\times n}S_{n\times n}(V_{n\times d})^T}$$

Input Matrix A =

D1	D2	D3	D4	D5	D6	D7	D8	D9
0.6532	0	0	0.6532	0	0	0	0	0
0.6532	0	0.6532	0	0	0	0	0	0
0.6532	0.6532	0	0	0	0	0	0	0
0	0.4771	0.4771	0	0.4771	0	0	0	0
0	0.4771	0.4771	0.9542	0	0	0	0	0
0	0.6532	0	0	0.6532	0	0	0	0
0	0.6532	0	0	0.6532	0	0	0	0
0	0	0.6532	0.6532	0	0	0	0	0
0	0.6532	0	0	0	0	0	0	0.6532
0	0	0	0	0	0.4771	0.4771	0.4771	0
0	0	0	0	0	0	0.4771	0.4771	0.4771
0	0	0	0	0	0	0	0.6532	0.6532

Matrix U =



```
-0.2510 -0.3574 -0.1232  0.3427 -0.2186 -0.4167  0.4647 -0.1865  0.3517
-0.2371 -0.2704 -0.0676  0.4696  0.4307  0.4599 -0.0067 -0.0558  0.1256
-0.3300  0.0906  0.0703  0.5983 -0.1360 -0.1420 -0.3115  0.2201 -0.4556
-0.3537  0.1413  0.1473 -0.1404  0.3656  0.2708  0.0123  0.0030 -0.0164
-0.4922 -0.3441 -0.0935 -0.3941 -0.2365 -0.1647 -0.2160  0.1424 -0.2911
-0.3459  0.3579  0.2404 -0.0886  0.1153 -0.1617  0.2503 -0.0410  0.0449
-0.3459  0.3579  0.2404 -0.0886  0.1153 -0.1617  0.2503 -0.0410  0.0449
-0.2905 -0.4161 -0.1330 -0.3342  0.1214  0.1882  0.0046 -0.0404  0.0914
-0.2750  0.3386 -0.1771  0.0311 -0.4930  0.2932 -0.3881 -0.1733  0.5174
-0.0132  0.1117 -0.3899 -0.0217  0.4990 -0.5127 -0.3942  0.2336  0.3362
```

```
  -0.0443  0.2038 -0.5418 -0.0127  0.1127 -0.0471  0.0409 -0.6922 -0.4088
  -0.0556  0.2336 -0.5785 -0.0071 -0.1121  0.2355  0.4553  0.5698 -0.0879
```

Matrix S =

```
         1.8798 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
         0.0000 1.4713 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
         0.0000 0.0000 1.3334 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
         0.0000 0.0000 0.0000 1.0247 0.0000 0.0000 0.0000 0.0000 0.0000
         0.0000 0.0000 0.0000 0.0000 0.8460 0.0000 0.0000 0.0000 0.0000
         0.0000 0.0000 0.0000 0.0000 0.0000 0.7626 0.0000 0.0000 0.0000
         0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.5251 0.0000 0.0000
         0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.3069 0.0000
         0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.2189
```

Matrix V^T =

```
  -0.2842 -0.6653 -0.3980 -0.4380 -0.3302 -0.0033 -0.0146 -0.0339 -0.1261
  -0.2385  0.4426 -0.3706 -0.5666  0.3636  0.0362  0.1023  0.2060  0.3201
  -0.0591  0.2025 -0.0790 -0.1924  0.2883 -0.1395 -0.3334 -0.6168 -0.5640
   0.8992  0.0394 -0.1626 -0.3616 -0.1783 -0.0101 -0.0160 -0.0205  0.0094
   0.0587 -0.2349  0.4991 -0.3418  0.3842  0.2814  0.3450  0.2585 -0.4036
  -0.0846 -0.0812  0.6216 -0.4018 -0.1076 -0.3208 -0.3502 -0.1484  0.4234
   0.1823 -0.4327 -0.1876  0.1913  0.6338 -0.3581 -0.3210  0.2453  0.1207
  -0.0474  0.1508  0.0212 -0.0402 -0.1701  0.3632 -0.7129  0.4999 -0.2322
   0.0647 -0.2174 -0.0226  0.0532  0.2324  0.7327 -0.1581 -0.4203  0.3908
```

<u>Query Vector:</u>

The query vector for the query terms: ***“human computer*** **tree graph*”*** is presented below:

```
                      0.6532 0 0.6532 0 0 0 0 0 0 0.4771 0.4771 0
```

Now the Reduced **SVD** (with k = 2) has been computed for the 12×9 matrix **A** to obtain the rank-2 approximation or A_2 as defined in **equation 12**. For easy to understand, the selected parts have been highlighted in the above **U** and **S** matrices. Also the highlighted parts of $\mathbf{V^T}$ give the coordinates of document vectors as defined in **section 2.3** (**2**). So, the coordinates of document vectors are presented below:

```
     D1           D2      D3       D4       D5       D6      D7        D8       D9
  -0.2842 -0.6653 -0.3980 -0.4380 -0.3302 -0.0033 -0.0146 -0.0339 -0.1261
  -0.2385  0.4426 -0.3706 -0.5666  0.3636  0.0362  0.1023  0.2060  0.3201
```

Query Vector Coordinates:

The query vector coordinates is obtained from the following equation as:

$$\hat{q} \;=\; q^T U_k S_k^{-1}$$

which is defined above and given by **Equation 13** and the query vector coordinates is:

```
-0.2165 -0.0161
```

Output Generated

For the query ***"human computer tree graph"***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.8116	Yes
D2	0.7892	Yes
D3	0.7804	Yes
D4	0.6686	Yes
D5	0.6155	Yes
D6	0.0167	Yes
D7	0.0675	Yes
D8	0.0888	Yes
D9	0.2965	Yes

Table 4.6: Computing Cosine Scores and Marking Document Relevancy.

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.6 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D2	A survey of user opinion of computer system response time.
D3	The EPS user interface management system.
D4	System and human system engineering testing of EPS.

D5	Relation of user perceived response time to error measurement.
D9	Graph minors: A survey.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D7	The intersection graph of paths in trees.
D6	The generation of random, binary, ordered trees.

Table 4.7: Ranked Result of a Query where Non-Ranked Documents are Not Displayed.

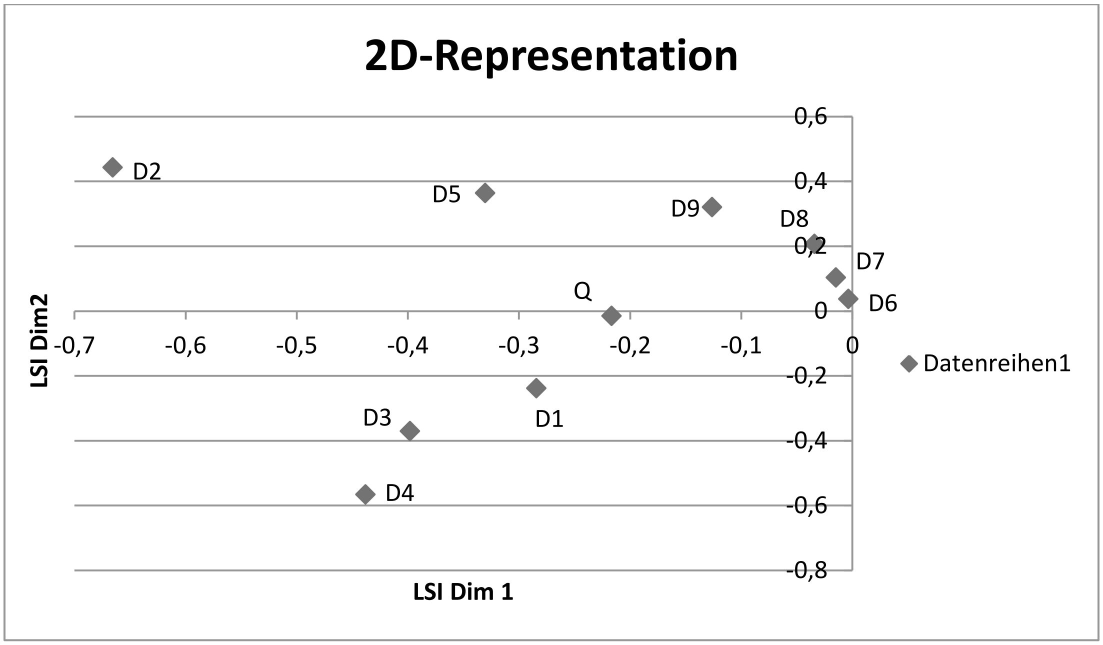


Figure 4.1: Documents Projected into 2D Semantic Space

<u>Analysis of the Experiment Result</u>

<u>Query terms</u>: ***“human computer*** **tree graph*”***

Since the present study has taken altogether nine documents of two groups (i.e. conceptually disjoint), five about HCI and four about graph theory documents. The above query is formulated considering the whole nine documents but not only considering the certain group.

To find documents relevant to the query: ***“human computer tree graph”***, both the IR models i.e. VSM and LSI models have been applied in **Experiment 1**.

As depicted in Table 4.5, VSM has returned documents D1,D2, D4, D6, D7, D8 and D9 since they share one or more terms with the query. But documents D3 and D5 have not returned since they share no terms at all with the query given though these documents are also relevant to the query in the meaning.

As depicted in table 4.7, LSI has been applied using two factors, i.e. A_2 is used to approximate the original 12×9 term-document matrix **A**. All the 9 documents relevant to the query have been returned from D1 to D9. Hence, the LSI approach can extract two additional documents (D3 and D5) which are relevant to the query yet share no common terms. This ability to retrieve relevant information based on meaning rather than literal term usage is the main property of LSI.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various tests on different number of k factors for the same query (i.e. ***“human computer*** **tree graph*”***). The choosing of different number of k factors and their results are presented below.

2. A_4 Approximation (i.e. 4D-Representation)

Query terms: ***“human computer*** **tree graph*”***

Query Vector:

```
0.6532 0 0.6532 0 0 0 0 0 0 0.4771 0.4771 0
```

Query Vector Coordinates:

```
-0.2165 -0.0161 -0.3593  0.5838
```

Output Generated

For the query ***"human computer tree graph"***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.8727	Yes
D2	0.1469	Yes
D3	0.0621	Yes
D4	-0.064	Yes but not taken
D5	-0.331	Yes but not taken
D6	0.4269	Yes
D7	0.4456	Yes
D8	0.4561	Yes
D9	0.4847	Yes

Table 4.8: Computing Cosine Scores and Marking Document Relevancy.

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.8 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D9	Graph minors: A survey.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D7	The intersection graph of paths in trees.
D6	The generation of random, binary, ordered trees.
D2	A survey of user opinion of computer system response time.
D3	The EPS user interface management system.

Table 4.9: Ranked Result of a Query where Non-Ranked Documents are Not Displayed.

As depicted in Table 4.9, LSI has been applied using four factors, i.e. A_4 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 7 documents D1, D2, D3, D6, D7, D8 and D9 have been returned. The LSI approach has missed 2 documents D4 and D5 which are also relevant to the query.

3. A_6 Approximation (i.e. 6D-Representation)

<u>Query terms</u>: *“human computer* tree graph*”*

<u>Query Vector</u>:

```
0.6532  0  0.6532  0  0  0  0  0  0  0.4771  0.4771  0
```

<u>Query Vector Coordinates:</u>

```
-0.2165 -0.0161 -0.3593  0.5838  0.0712 -0.8287
```

<u>Output Generated</u>

For the query ***“human computer tree graph”***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.6365	Yes
D2	0.1453	Yes
D3	-0.421	Yes but not taken
D4	0.2513	Yes
D5	-0.032	Yes but not taken
D6	0.6667	Yes
D7	0.6435	Yes
D8	0.4507	Yes
D9	-0.154	Yes but not taken

Table 4.10: Computing Cosine Scores and Marking Document Relevancy.

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.10 and non-ranked documents are not presented.

D6	The generation of random, binary, ordered trees.
D7	The intersection graph of paths in trees.
D1	Human machine interface for ABC computer applications.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D4	System and human system engineering testing of EPS.
D2	A survey of user opinion of computer system response time.
	.

Table 4.11: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.11, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 6 documents D1, D2, D4, D6, D7 and D8 have been returned. The LSI approach has missed 3 documents D3, D5 and D9 which are also relevant to the query.

4. A_8 Approximation (i.e. 8D-Representation)

<u>Query terms</u>: *“human computer* tree graph*”*

<u>Query Vector</u>:

```
                0.6532  0  0.6532  0  0  0  0  0  0  0.4771  0.4771  0
```

<u>Query Vector Coordinates:</u>

```
      -0.2165 -0.0161 -0.3593   0.5838   0.0712 -0.8287 -0.1305 -0.6416
```

<u>Output Generated</u>

For the query ***“human computer tree graph”***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.5423	Yes
D2	0.078	Yes
D3	-0.346	Yes but not taken
D4	0.2124	Yes
D5	0.0007 *	Yes but not taken
D6	0.1653	Yes
D7	0.7329	Yes
D8	0.002 *	Yes but not taken
D9	-0.014	Yes but not taken

Table 4.12: Computing Cosine Scores and Marking Document Relevancy.

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.12 and non-ranked documents are not presented.

D7	The intersection graph of paths in trees.
D1	Human machine interface for ABC computer applications.
D4	System and human system engineering testing of EPS.
D6	The generation of random, binary, ordered trees.
D2	A survey of user opinion of computer system response time.
	.

Table 4.13: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.13, LSI has been applied using 8 factors, i.e. A_8 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 7 documents D1, D2, D4, D6 and D7 have been returned. The LSI approach has missed 4 documents D3, D5, D8 and D9 which are also relevant to the query. Here, * sign indicates that though D5 and D8 have positive value but they are almost equal to zero. So D5 and D8 are ignored.

Comparison Table

Table 4.14 lists the LSI-ranked documents with different numbers of factors (k). The documents presented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors			
K = 2	K = 4	K = 6	K = 8
D1	D1	D6	D7
D2	D9	D7	D1
D3	D8	D1	D4
D4	D7	D8	D6
D5	D6	D4	D2
D9	D2	D2	
D8	D3		
D7			
D6			

Table 4.14: Returned Documents Based on Different Numbers of LSI Factors.

From the above table, it can be concluded that k =2 is the best approximation for the given query terms ***“human computer* tree graph*”***

4.2.2 Demonstration of Partial Matching to the Query Terms

Experiment 2

Applying VSM

For the same input data set mentioned in Table 4.1, another query is tested and its result is presented below.

Query terms: ***“human computer interaction”***

Query Vector:

```
0.6532  0  0.6532  0  0  0  0  0  0  0  0  0
```

Query vector coordinates:

```
0.6532  0  0.6532  0  0  0  0  0  0  0  0  0
```

Output Generated

For the query **"human computer interaction"**, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.8165	Yes
D2	0.3141	Yes
D3	0	Not Retrieved
D4	0.3478	Yes
D5	0	Not Retrieved
D6	0	Not Retrieved
D7	0	Not Retrieved
D8	0	Not Retrieved
D9	0	Not Retrieved

Table 4.15: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.15 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D4	System and human system engineering testing of EPS.
D2	A survey of user opinion of computer system response time.

Table 4.16: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

Applying LSI Model

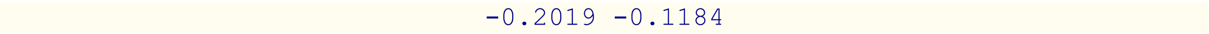
1. A_2 Approximation (i.e. 2D-Representation)

Query terms: ***"human computer interaction"***

Query Vector:

```
0.6532 0 0.6532 0 0 0 0 0 0 0 0 0
```

Query Vector Coordinates:



```
-0.2019 -0.1184
```

Output Generated

For the query ***"human computer interaction"***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.986	Yes
D2	0.438	Yes
D3	0.976	Yes
D4	0.9278	Yes
D5	0.2054	Yes
D6	-0.425	No
D7	-0.379	No
D8	-0.359	No
D9	-0.154	No

Table 4.17: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.17 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D3	The EPS user interface management system.
D4	System and human system engineering testing of EPS.
D2	A survey of user opinion of computer system response time.
D5	Relation of user perceived response time to error measurement.

Figure 4.18: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

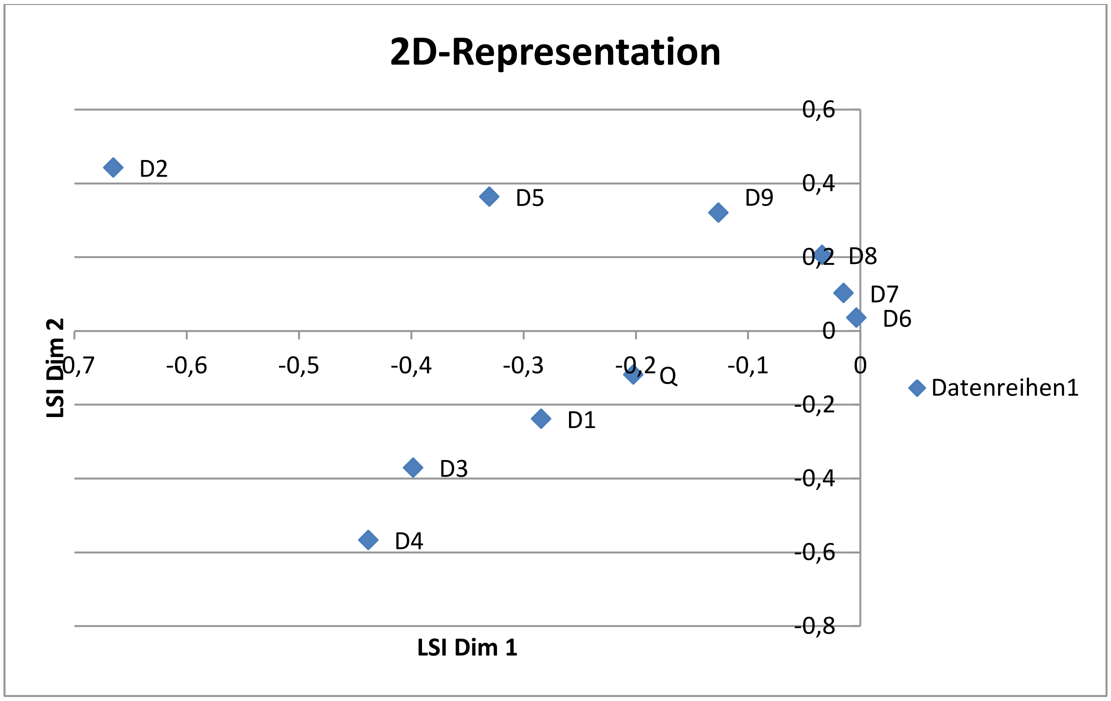


Figure 4.2: Documents Projected into 2D Semantic Space

Analysis of the Experiment Result

Query terms: ***"human computer* interaction"**

Since the present study has taken altogether nine documents of two groups (i.e. conceptually disjoint), five about HCI and four about graph theory documents. The above query is formulated considering the first five documents (i.e. HCI group) only.

To find documents relevant to the query: ***"human computer interaction"***, both the IR models i.e. VSM and LSI models have been applied in **Experiment 2**.

As depicted in Table 4.16, VSM has returned documents D1, D4 and D2 since they share one or more terms with the query. But documents D3 and D5 have not returned since they share no terms at all with the query given though these documents are also relevant to the query in the meaning.

As depicted in Table 4.18, LSI has been applied using two factors, i.e. A_2 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), all the 5 documents relevant to the query have been returned from D1 to D5. Hence, the LSI approach can extract two additional documents (D3 and D5) which are relevant to the query yet share no common terms. This ability to retrieve relevant information based on meaning rather than literal term usage is the main property of LSI.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various test on different number of k factors for the same query (i.e. **“*huma*n computer interaction”**). The choosing of different number of k factors and their result are presented below.

2. A_4 Approximation (i.e. 4D-Representation)

Query terms: *“human computer* interaction”

Query Vector:

```
0.6532 0 0.6532 0 0 0 0 0 0 0 0 0
```

Query Vector Coordinates:

```
-0.2019 -0.1185 -0.0260  0.5999
```

Output Generated

For the query ***“human computer interaction”***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.9977	Yes
D2	0.1885	Yes
D3	0.0779	Yes
D4	-0.106	Yes but not taken
D5	-0.236	Yes but not taken
D6	-0.065	No
D7	-0.045	No
D8	-0.033	No
D9	0.0184	No

Table 4.19: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.19 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D2	A survey of user opinion of computer system response time.
D3	The EPS user interface management system.
D9	Graph minors: A survey.

Table 4.20: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.20, LSI has been applied using 4 factors, i.e. A_4 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 4 documents D1, D2, D3 and D9 have been returned. The LSI approach has missed 2 documents D4 and D5 which are also relevant to the query and it has also retrieved one document D9 which is not relevant to the query as expected.

3. A_6 Approximation (i.e. 6D-Representation)

Query terms: *“human computer* interaction*”*

Query Vector:

```
0.6532 0 0.6532 0 0 0 0 0 0 0 0 0
```

Query Vector Coordinates:

```
-0.2019 -0.1185 -0.0260  0.5999 -0.2738 -0.4785
```

Output Generated

For the query ***“human computer interaction”***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.7833	Yes
D2	0.2783	Yes
D3	-0.487	Yes but not taken
D4	0.2763	Yes
D5	-0.237	Yes but not taken
D6	0.1842	No
D7	0.1232	No
D8	-0.022	No
D9	-0.113	No

Table 4.21: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.21 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D2	A survey of user opinion of computer system response time.
D4	System and human system engineering testing of EPS.
D6	The generation of random, binary, ordered trees.
D7	The intersection graph of paths in trees.

Table 4.22: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.22, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 5 documents D1, D2, D4, D6 and D7 have been returned. The LSI approach has missed 2 documents D3 and D5 which are also relevant to the query and it has also retrieved 2 documents D6 and D7 which are not relevant to the query as expected.

4. A8 Approximation (i.e. 8D-Representation)

Query terms: **“*human computer* interaction”**

Query Vector:

```
                    0.6532  0  0.6532  0  0  0  0  0  0  0  0  0
```

Query Vector Coordinates:

```
      -0.2019 -0.1185 -0.0260  0.5999 -0.2738 -0.4785  0.1906  0.0714
```

Output Generated

For the query ***“human computer interaction”***, following result is obtained using the data set mentioned in Table 4.1 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.784	Yes
D2	0.1547	Yes
D3	-0.504	Yes but not taken
D4	0.3021	Yes
D5	-0.042	Yes but not taken
D6	0.0473	No
D7	-0.057	No
D8	0.087	No
D9	-0.097	No

Table 4.23: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.23 and non-ranked documents are not presented.

D1	Human machine interface for ABC computer applications.
D4	System and human system engineering testing of EPS.
D2	A survey of user opinion of computer system response time.
D8	Graph minors IV: Widths of trees and well-quasi-ordering.
D6	The generation of random, binary, ordered trees.

Table 4.24: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.24, LSI has been applied using 8 factors, i.e. A_8 is used to approximate the original 12×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 5 documents D1, D2, D4, D6 and D8 have been returned. The LSI approach has missed 2 documents D3 and D5 which are also relevant to the query and it has also retrieved 2 documents D6 and D8 which are not relevant to the query as expected.

Comparison Table

Table 4.25 lists the LSI-ranked documents with different numbers of factors (k). The documents presented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors			
K = 2	K = 4	K = 6	K = 8
D1	D1	D1	D1
D3	D2	D2	D4
D4	D3	D4	D2
D2	D9	D6	D8
D5		D7	D6

Table 4.25: Returned Documents Based on Different Numbers of LSI Factors.

From the above table, it can be concluded that k =2 is the best approximation for the given query terms ***"human computer* interaction".**

4.2.3 Demonstration of Synonyms Test

Experiment: 3

List of Input Documents:

The present study has taken nine documents of two distinct groups. The first group i.e. first five documents is related to the **"humor"** and the second group i.e. last four documents is related to the **"car"**. Here, **"humor"** has the synonyms (i.e. identical or similar meaning) as **joke, comedy, funny** and **"car"** has the synonyms as **automobile, vehicle**. This study has been trying to focus whether these two models under study could really solve the problems of synonyms.

Indexing is done to these input documents as described in **section 3.1** and all the underlined words in Table 4.26 denote keywords which are used as referents to the documents. Also for simplicity, simple term weighting i.e. term-frequency (tf_i) is used in this experiment.

D1	Joke clean humor and humor.
D2	Funny picture with joke page.
D3	Comedy circus.
D4	Comedy song funny.
D5	Humor story.
D6	BMW automobile vehicle service.
D7	Car vehicle.
D8	Car and car automobile center.
D9	Vehicle, vehicle and vehicle.

Table 4.26: List of Input Documents with Underlined Keywords.

Applying VSM

The term-document matrix with simple term- frequency (tf_i) weighting is presented below for the input data set mentioned in Table 4.26.

Terms	Q	D1	D2	D3	D4	D5	D6	D7	D8	D9
Humor	1	2	0	0	0	1	0	0	0	0
Joke	0	1	1	0	0	0	0	0	0	0
Funny	0	0	1	0	1	0	0	0	0	0
Comedy	0	0	0	1	1	0	0	0	0	0
Automobile	0	0	0	0	0	0	1	0	1	0
Vehicle	0	0	0	0	0	0	1	1	0	3
Car	0	0	0	0	0	0	0	1	2	0

Table 4.27: Term-Document Matrix and Term Weighting (tf_i)

Query term: “*hu*mor”

Query Vector Coordinates:

```
1  0  0  0  0  0  0
```

Output Generated

For the query **“*hu*mor”**, following output result is obtained using the input data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.8944	Yes
D2	0	Not Retrieved
D3	0	Not Retrieved
D4	0	Not Retrieved
D5	1	Yes
D6	0	Not Retrieved
D7	0	Not Retrieved
D8	0	Not Retrieved
D9	0	Not Retrieved

Table 4.28: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.28 and non-ranked documents are not presented.

D5	Humor story.
D1	Joke clean humor and humor.

Table 4.29: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

Applying LSI Model

$$A_{t\times d} = U_{t\times n}S_{n\times n}(V_{n\times d})^T$$

Singular Value Decomposition (SVD)

Input Matrix A =

Terms	D1	D2	D3	D4	D5	D6	D7	D8	D9
humor	2	0	0	0	1	0	0	0	0
joke	1	1	0	0	0	0	0	0	0
funny	0	1	0	1	0	0	0	0	0
comedy	0	0	1	1	0	0	0	0	0
automobile	0	0	0	0	0	1	0	1	0
vehicle	0	0	0	0	0	1	1	0	3
car	0	0	0	0	0	0	1	2	0

Matrix U =

```
 0.0000   0.8776   0.0000  -0.2342  -0.3061   0.0000   0.2852
 0.0000   0.4628   0.0000   0.2153   0.5346   0.0000  -0.6735
 0.0000   0.1215   0.0000   0.7183   0.3485   0.0000   0.5897
 0.0000   0.0300   0.0000   0.6187  -0.7065   0.0000  -0.3424
-0.1460   0.0000  -0.4130   0.0000   0.0000  -0.8989   0.0000
-0.9692   0.0000   0.2420   0.0000   0.0000   0.0462   0.0000
-0.1984   0.0000  -0.8780   0.0000   0.0000   0.4357   0.0000
```

Matrix S =

```
3.3698 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 2.4606 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 2.3802 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.7779 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 1.2275 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.9896 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.5267 0.0000 0.0000
```

Matrix V^T =

```
 0.0000  0.0000  0.0000  0.0000  0.0000 -0.3309 -0.3465 -0.1611 -0.8628
 0.9014  0.2375  0.0122  0.0616  0.3566  0.0000  0.0000  0.0000  0.0000
 0.0000  0.0000  0.0000  0.0000  0.0000 -0.0719 -0.2672 -0.9113  0.3050
-0.1423  0.5251  0.3480  0.7520 -0.1317  0.0000  0.0000  0.0000  0.0000
-0.0632  0.7194 -0.5755 -0.2917 -0.2493  0.0000  0.0000  0.0000  0.0000
 0.0000  0.0000  0.0000  0.0000  0.0000 -0.8617  0.4869 -0.0279  0.1402
-0.1957 -0.1591 -0.6500  0.4697  0.5416  0.0000  0.0000  0.0000  0.0000
 1.6051  1.1612 -0.8069 -0.2377 -0.7767 -0.3407  0.0306 -0.0317  0.4259
 0.0018 -0.1307 -0.0017  0.0030  0.0662  0.0591 -0.0145  0.0090 -0.0148
```

1. A_2 Approximation (i.e. 2D-Representation)

Query term: "humor"

Query Vector:

```
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
0.0000 0.3567
```

Output Generated

For the query **"humor"**, following result is obtained using the data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	1	Yes
D2	1	Yes
D3	1	Yes
D4	1	Yes
D5	1	Yes
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.30: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.30 and non-ranked documents are not presented.

D1	Joke clean humor and humor.
D2	Funny picture with joke page.
D3	Comedy circus.
D4	Comedy song funny.
D5	Humor story.

Table 4.31: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

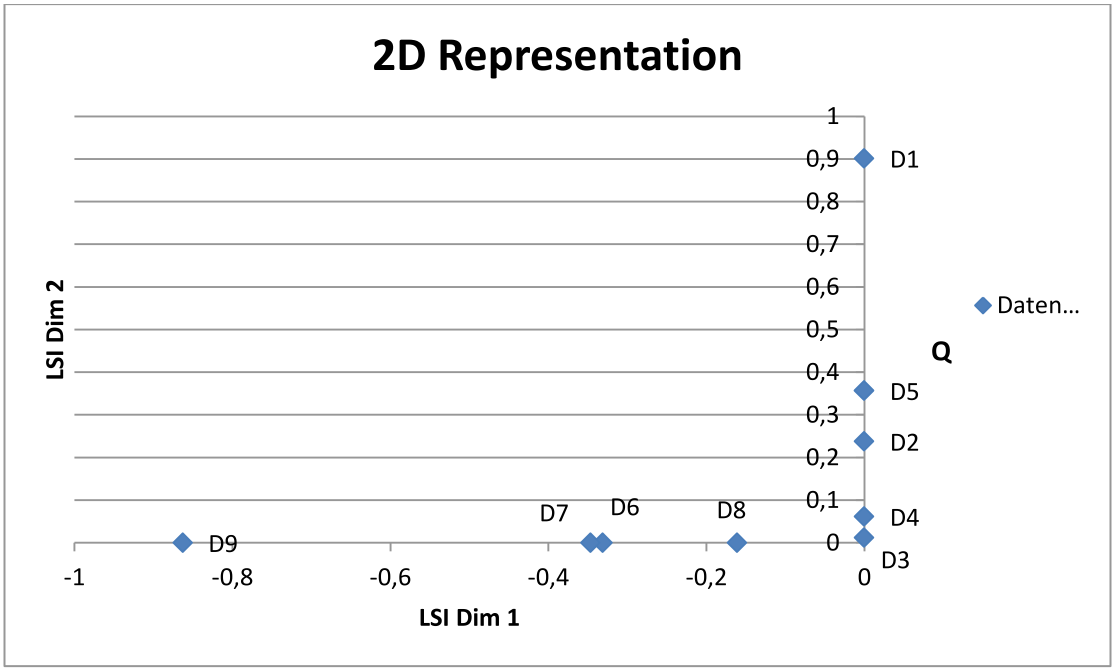


Figure 4.3: Documents Projected into 2D Semantic Space

Analysis of the Experiment Result

Query term: *“humor”*

Since the present study has taken altogether nine documents of two groups (i.e. conceptually disjoint), five about synonyms or identical meaning to **“humor”** and four about synonyms or identical meaning to **“car”** documents. The above query is formulated considering the first five documents only because all those documents is somewhat related to the synonymous or identical meaning to **“humor”**.

To find documents relevant to the query: ***“*humor*”***, both the IR models i.e. VSM and LSI models have been applied in **Experiment 3**.

From Table 4.29, When VSM is applied; it has returned documents D1 and D5 since they share one or more terms with the query. But documents D2, D3 and D4 have not been returned since they share no terms at all with the query given but these documents are also relevant to the query in the meaning (i.e. synonymous).

As depicted in Table 4.31, LSI has been applied using two factors, i.e. A_2 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), all the 5 relevant documents to the query have been returned from D1 to D5. Hence, the LSI approach can extract 3 additional documents (D2, D3 and D4) which are relevant to the query yet share no common terms. It can also be concluded that LSI works very well for the synonyms while observing the cosine scores.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various test on different number of k factors for the same query (i.e.***“humor”***). The choosing of different number of k factors and their result are presented below.

2. A_4 Approximation (i.e. 4D-Representation)

<u>Query terms</u>: “humor”

<u>Query Vector:</u>

```
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

<u>Query Vector Coordinates:</u>

```
0.0000  0.3567  0.0000 -0.1317
```

Output Generated

For the query **"humor"**, following result is obtained using the data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.9806	Yes
D2	0.071	Yes
D3	-0.313	Yes but not taken
D4	-0.269	Yes but not taken
D5	1	Yes
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.32: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.32 and non-ranked documents are not presented.

D5	Humor story.
D1	Joke clean humor and humor.
D2	Funny picture with joke page.

Table 4.33: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.33, LSI has been applied using 4 factors, i.e. A_4 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 3 documents D1, D2, and D5 have been returned. The LSI approach has missed 2 documents D3 and D4 which are also relevant to the query in meaning (i.e. synonyms).

3. A_6 Approximation (i.e. 6D-Representation)

Query term: "humor"

Query Vector:

```
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
0.0000  0.3567  0.0000 -0.1317 -0.2494  0.0000
```

Output Generated

For the query **"humor"**, following result is obtained using the data set mentioned in Table 4.26 is presented in the below.

Doc. Id	Cosine Scores	Relevancy
D1	0.8559	Yes
D2	-0.391	Yes but not taken
D3	0.3336	Yes
D4	-0.012	Yes but not taken
D5	1	Yes
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.34: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.34 and non-ranked documents are not presented.

D5	Humor story.
D1	Joke clean humor and humor.
D3	Comedy circus.

Table 4.35: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.35, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 3 documents D1, D3, and D5 have been returned. The LSI approach has missed 2 documents D2 and D4 which are also relevant to the query in meaning (i.e. synonyms).

Comparison Table

Table 4.36 lists the LSI-ranked documents with different numbers of factors (k). The documents presented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors.		
K = 2	K = 4	K = 6
D1	D5	D5
D2	D1	D1
D3	D2	D3
D4		
D5		

Table 4.36: Returned Documents Based on Different Numbers of LSI Factors.

From the above table, it can be concluded that k =2 is the best approximation for the given query term **"humor"**.

1. A_2 Approximation (i.e. 2D-Representation)

Applying LSI

Query term: "car"

Query Vector:

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
```

Query Vector Coordinates:

```
-0.0589   0.0000
```

Output Generated

For the query ***"car"***, following result is obtained using the data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	1	Yes
D7	1	Yes
D8	1	Yes
D9	1	Yes

Table 4.37: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.37 and non-ranked documents are not presented.

D6	BMW automobile vehicle service.
D7	Car vehicle.
D8	Car and car automobile center.
D9	Vehicle, vehicle and vehicle.

Table 4.38: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

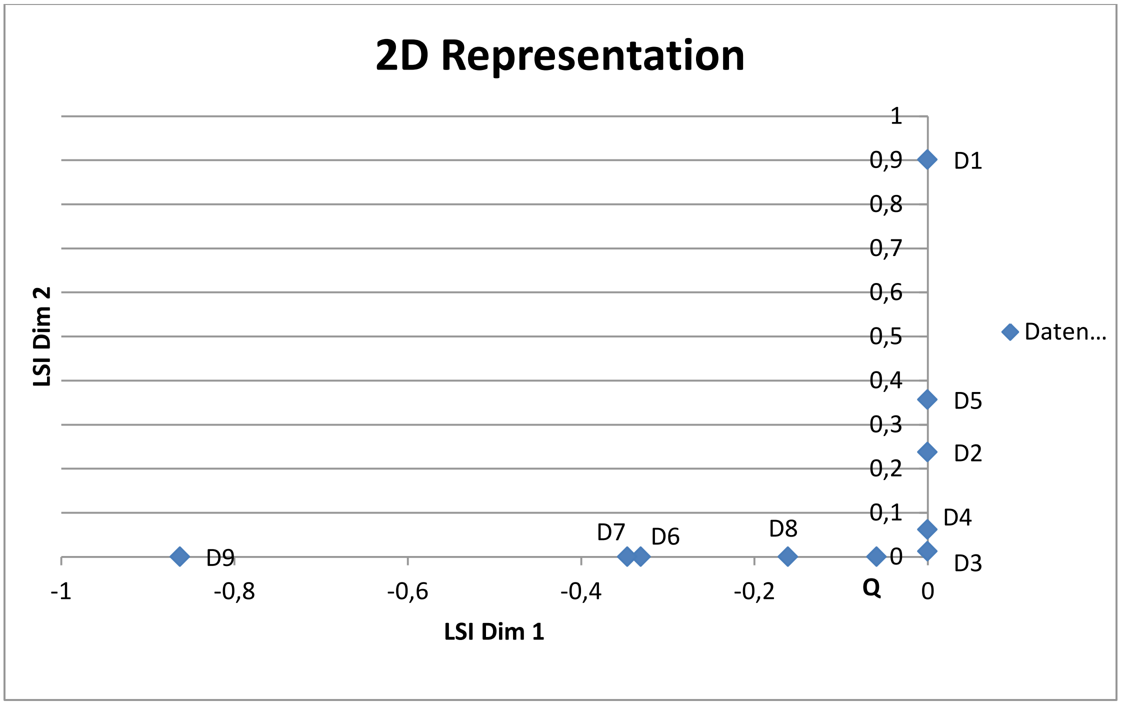


Figure 4.4 : Documents Projected into 2D Semantic Space

Analysis of the Experiment Result

Query term: "car"

From **experiment 1** and **experiment 2**, it is clear that VSM would return documents only when documents share one or more terms with the query given. So, it is obvious that when applying VSM for this query. It would have returned documents D7 and D8 because they share the term **"car"** with the query and missed D6 and D9 which are also relevant in meaning (i.e synonyms)

As depicted in Table 4.38, LSI has been applied using 2 factors, i.e. A_2 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), all 4 documents from D6 to D9 have been returned. They all are relevant to the query. It can be concluded that LSI approach works well for the synonyms or identical meaning while observing the cosine values.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various test on different number of k factors for the same query (i.e. ***"car"***). The choosing of different number of k factors and their result are presented below.

2. A$_4$ Approximation (i.e. 4D-Representation)

<u>Query term</u>: "car"

<u>Query Vector:</u>

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
```

<u>Query Vector Coordinates:</u>

```
-0.0589  0.0000 -0.3688  0.0000
```

<u>Output Generated</u>

For the query **"car"** following result is obtained using the data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	0.3638	Yes
D7	0.7279	Yes
D8	0.9999	Yes
D9	-0.18	Yes but not taken

Table 4.39: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.39 and non-ranked documents are not presented.

D8 D7 D6	Car and car automobile center. Car vehicle. BMW automobile vehicle service.

Table 4.40: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.40, LSI has been applied using 4 factors, i.e. A_4 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 3 documents D6, D7 and D8 have been returned. The LSI approach has missed 1 document D9 which is also relevant to the query in meaning (i.e. synonyms).

3. A_6 Approximation (i.e. 6D-Representation)

Query term: "car"

Query Vector:

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
```

Query Vector Coordinates:

```
-0.0589  0.0000 -0.3688  0.0000  0.0000  0.4403
```

Output Generated

For the query **"car"**, following result is obtained using the data set mentioned in Table 4.26 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	-0.624	Yes but not taken
D7	0.8819	Yes
D8	0.6235	Yes
D9	0.0001 *	Yes but not taken

Table 4.41: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.41 and non-ranked documents are not presented.

D7	Car vehicle.
D8	Car and car automobile center.

Table 4.42: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in Table 4.42, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 7×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 2 documents D7 and D8 have been returned. The LSI approach has missed 2 documents D6 and D9 which are also relevant to the query in meaning (i.e. synonyms). Here, * sign indicates that though D9 has positive value but it is almost equal to zero. So D9 is ignored.

Comparison Table

Table 4.43 lists the LSI-ranked documents with different numbers of factors (k). The documents presented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors.		
K = 2	K = 4	K = 6
D6	D8	D7
D7	D7	D8
D8	D6	
D9		

Table 4.43: Returned Documents Based on Different Numbers of LSI Factors

From the above table, it can be concluded that k =2 is the best approximation for the given query term ***"car"***.

4.2.4 Demonstration of Polysemys Test

List of Input Documents:

Most words have more than one distinct meaning. These words are known as **polysemy** words. From table 4.44, the word **“rock”** means **rock marble** or **rock music**. The two documents D1 and D3 have used **“rock”** as two different meanings. Also document D4 gives the meaning of **rock** as **rock music** though it doesn’t contain word **rock.**

Again, the word **“bank”** may be **bank finance institute** or **river bank** (i.e D6 and D8). So the word **“bank”** also holds the property of polysemy. Also document D9 gives the meaning of **bank** though it doesn’t contain word **bank**. This study has been trying to focus whether both the models under study would be successful to retrieve the relevant documents when the case is about the polysemy.

The present study has taken nine documents of two distinct groups. The first group i.e. first five documents is related to the polysemy word **“rock”** and the second group i.e. last four documents is related to the polysemy word **“bank”**.

Doc ID	**Input Documents**
D1	The rock marble.
D2	The rock potato.
D3	The rock music.
D4	Type of loud modern music with music guitars and music drums.
D5	The rock music, marble and potato.
D6	The river bank.
D7	Short river or long river and tuition institute.
D8	The bank finance institute.
D9	Finance institute or sides of a river.

Table 4.44: List of Input Documents with Underlined Keywords

Indexing is done to these input documents as described in **section 3.1** and all the underlined words in Table 4.44 denote keywords which are used as referents to the documents. Also for simplicity, simple term weighting i.e. term-frequency (tf_i) is used in this experiment.

Experiment: 4

Applying VSM

The term-document matrix with simple term- frequency (tf_i) weighting is presented below for the input data set mentioned in Table 4.44

Terms	Q	D1	D2	D3	D4	D5	D6	D7	D8	D9
rock	1	1	1	1	0	1	0	0	0	0
marble	0	1	0	0	0	1	0	0	0	0
potato	0	0	1	0	0	1	0	0	0	0
music	0	0	0	1	3	1	0	0	0	0
bank	0	0	0	0	0	0	1	0	1	0
river	0	0	0	0	0	0	1	2	0	1
institute	0	0	0	0	0	0	0	1	1	1
finance	0	0	0	0	0	0	0	0	1	1

Table 4.45: Term-Document Matrix and Term Weighting (tf_i)

Query term: “rock”

Query vector coordinates:

```
1   0   0   0   0   0   0   0
```

Output Generated

For the query **“rock”**, following output result is obtained using the input data set mentioned in Table 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.7071	Yes
D2	0.7071	No
D3	0.7071	Yes
D4	0	Not Retrieved
D5	0.5	No

D6	0	Not Retrieved
D7	0	Not Retrieved
D8	0	Not Retrieved
D9	0	Not Retrieved

Table 4.46: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.46 and non-ranked documents are not presented.

D1	The rock marble.
D2	The rock potato.
D3	The rock music.
D5	The rock music, marble and potato.

Table 4.47: Ranked Result of a Query where Non-ranked Documents are Not Displayed

Applying LSI Model

$$\mathbf{A_{t\times d} = U_{t\times n}S_{n\times n}(V_{n\times d})^T}$$

Singular Value Decomposition (SVD)

Input Matrix A =

Terms	D1	D2	D3	D4	D5	D6	D7	D8	D9
rock	1	1	1	0	1	0	0	0	0
marble	1	0	0	0	1	0	0	0	0
Potato	0	1	0	0	1	0	0	0	0
music	0	0	1	3	1	0	0	0	0
bank	0	0	0	0	0	1	0	1	0
river	0	0	0	0	0	1	2	0	1
institute	0	0	0	0	0	0	1	1	1
finance	0	0	0	0	0	0	0	1	1

Matrix U =

```
0.3129   0.0000  -0.6930   0.0000   0.0000   0.0000  -0.6495   0.0000
0.1706   0.0000  -0.4267   0.0000   0.0000  -0.7071   0.5375   0.0000
0.1706   0.0000  -0.4267   0.0000   0.0000   0.7071   0.5375   0.0000
0.9186   0.0000   0.3945   0.0000   0.0000   0.0000   0.0216   0.0000
0.0000   0.2330   0.0000   0.4451   0.8556   0.0000   0.0000  -0.1248
0.0000   0.7557   0.0000  -0.5851   0.1366   0.0000   0.0000   0.2607
0.0000   0.5329   0.0000   0.2884  -0.3958   0.0000   0.0000  -0.6901
0.0000   0.3011   0.0000   0.6135  -0.3044   0.0000   0.0000   0.6635
```

Matrix S =

```
3.4717 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 2.9702 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 2.3074 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 1.6468 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 1.1581 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.7895 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.3531 0.0000
```

Matrix V^T =

```
  0.1393   0.1393   0.3547   0.7938   0.4530   0.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.3329   0.6883   0.3592   0.5352
 -0.4853  -0.4853  -0.1294   0.5130  -0.4992   0.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000  -0.0850  -0.5354   0.8180   0.1924
   0.0000   0.0000   0.0000   0.0000   0.0000   0.8567  -0.1059   0.1342  -0.4867
 -0.7071   0.7071   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
 -0.1419  -0.1419  -0.7953   0.0820   0.5662   0.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.3848  -0.4779  -0.4288   0.6631
   0.0000   0.0000  -0.0161   0.0441  -0.0044   0.0122   0.0859  -0.1780   0.1780
```

1. A_2 Approximation (i.e. 2D-Representation)

Query term: "rock"

Query Vector:

```
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
                    0.0901 0.0000
```

Output Generated

For the query **"rock"**, following result is obtained using the data set mentioned in Table 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	1	Yes
D2	1	No
D3	1	Yes
D4	1	Yes
D5	1	No
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.48: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.48 and non-ranked documents are not presented.

D1	The rock marble.
D2	The rock potato.
D3	The rock music.
D4	Type of loud modern music with music guitars and music drums.
D5	The rock music, marble and potato.

Table 4.49: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

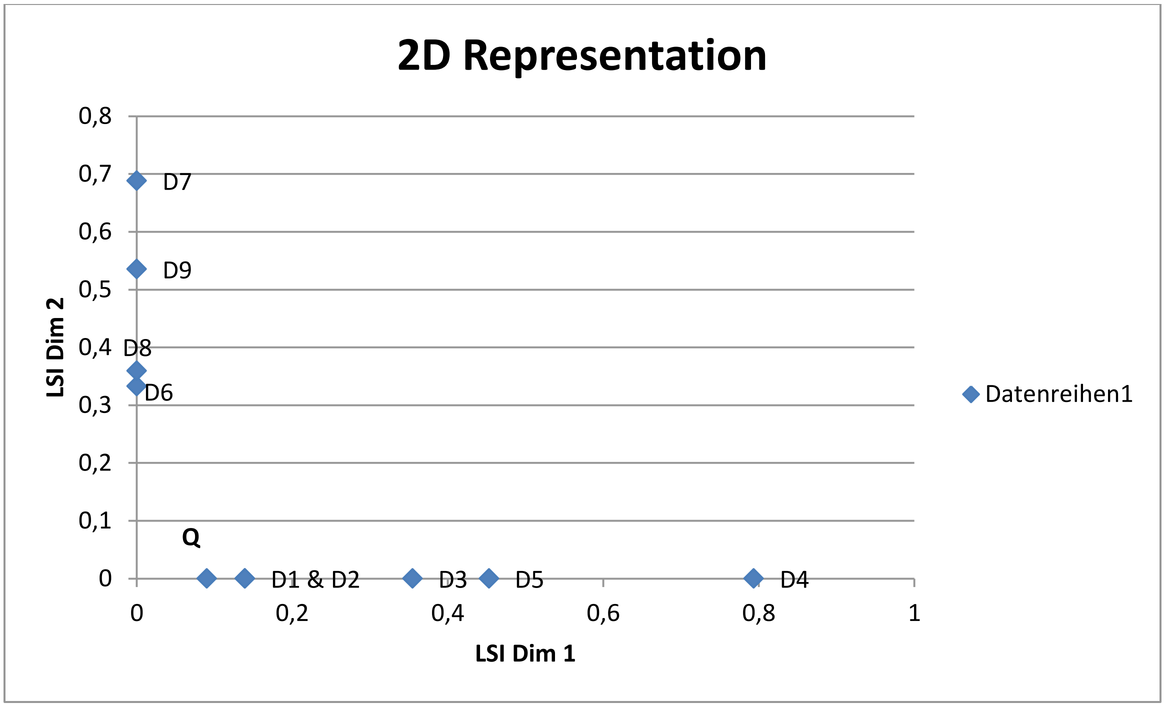


Figure 4.5: Documents Projected into 2D Semantic Space

Analysis of the Experiment Result

Query term: "rock"

From Table 4.47, When VSM has been applied; it has returned documents D1, D2, D3 and D5 since they share **"rock"** term with the query. It has missed document D4 which is also relevant to the query in polysemous meaning.

As depicted in table 4.49, LSI has been applied using 2 factors, i.e. A_2 is used to approximate the original 8×9 term-document matrix **A.** When only positive values are considered (so as to compare with VSM), then 5 documents have been returned from D1 to D5. But out of these, only D1, D3 and D4 are relevant which shows true polysemy meaning to the query. But documents D2 and D5 are irrelevant to the query in polysemy sense though posses the well cosine scores. Hence, it can be concluded that LSI approach does not work well for the polysemy or partial works for polysemy.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various test on different number of k factors for the same query (i.e. **"rock"**). The choosing of different number of k factors and their result are presented below.

2. A_4 Approximation (i.e. 4D-Representation)

Query term: "rock"

Query Vector:

```
        1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
                   0.0901  0.0000 -0.3003  0.0000
```

Output Generated

For the query **"rock"**, following result is obtained using the data set mentioned in Table 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.9999	Yes
D2	0.9999	No
D3	0.5982	Yes
D4	-0.279	Yes but not taken
D5	0.9024	No
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.50: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.50 and non-ranked documents are not presented.

D1	The rock marble.
D2	The rock potato.
D5	The rock music, marble and potato.
D3	The rock music.

Table 4.51: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.51, LSI has been applied using 4 factors, i.e. A_4 is used to approximate the original 8×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 4 documents D1, D2, D5, and D3 have been returned. Out of these, only D1 and D3 are relevant which shows true polysemy meaning to the query. But documents D2 and D5 are irrelevant to the query in polysemy sense though posses the well cosine scores. Also it has missed document D4 which shows polysemy meaning. Hence, it can be concluded that LSI approach does not work well for the polysemy or partial works for polysemy.

.3. A_6 Approximation (i.e. 6D-Representation)

<u>Query term</u>: "rock"

<u>Query Vector</u>

```
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

<u>Query Vector Coordinates:</u>

```
0.0901   0.0000 -0.3003   0.0000   0.0000   0.0000
```

<u>Output Generated</u>

For the query ***"rock"***, following result is obtained using the data set mentioned in 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0.5811	Yes
D2	0.5811	No
D3	0.5982	Yes
D4	-0.279	Yes but not taken
D5	0.9024	No
D6	0	No
D7	0	No
D8	0	No
D9	0	No

Table 4.52: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.52 and non-ranked documents are not presented.

D5	The rock music, marble and potato.
D3	The rock music.
D1	The rock marble.
D2	The rock potato.

Table 4.53: Ranked Result of a Query where Non-ranked Documents are Not Displayed

As depicted in table 4.53, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 8×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 4 documents D1, D2, D3 and D5 have been returned. Out of these, only D1 and D3 are relevant which shows true polysemy meaning to the query. But other documents D2 and D5 are irrelevant to the query in polysemy sense though posses the well cosine scores. Also it has missed document D4 which shows polysemy meaning. Hence, it can be concluded that LSI approach does not work well for the polysemy or partial works for polysemy.

Comparison Table

Table 4.50, lists the LSI-ranked documents with different numbers of factors (k). The documents represented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors.		
K = 2	K = 4	K = 6
D1	D1	D5
D2	D2	D3
D3	D5	D1
D4	D3	D2
D5		

Table 4.54: Returned Documents Based on Different Numbers of LSI Factors.

From the above table, it can be concluded that non of the above approximation is best but while considering, k = 2 is quite better approximation for the given query term **"rock"**. Therefore, LSI model only partially supports for the polysemy can be concluded from observing the above table.

1. A$_2$ Approximation (i.e. 2D-Representation)

Applying LSI

Query term: **"bank"**

Query Vector:

```
          0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
                            0.0000 0.0785
```

Output Generated

For the query **"bank"**, following result is obtained using the data set mentioned in 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	1	Yes
D7	1	No
D8	1	Yes
D9	1	Yes

Table 4.55: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.55 and non-ranked documents are not presented.

D6	The river bank.
D7	Short river or long river and tuition institute.
D8	The bank finance institute.
D9	Finance institute or sides of a river.

Table 4.56: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

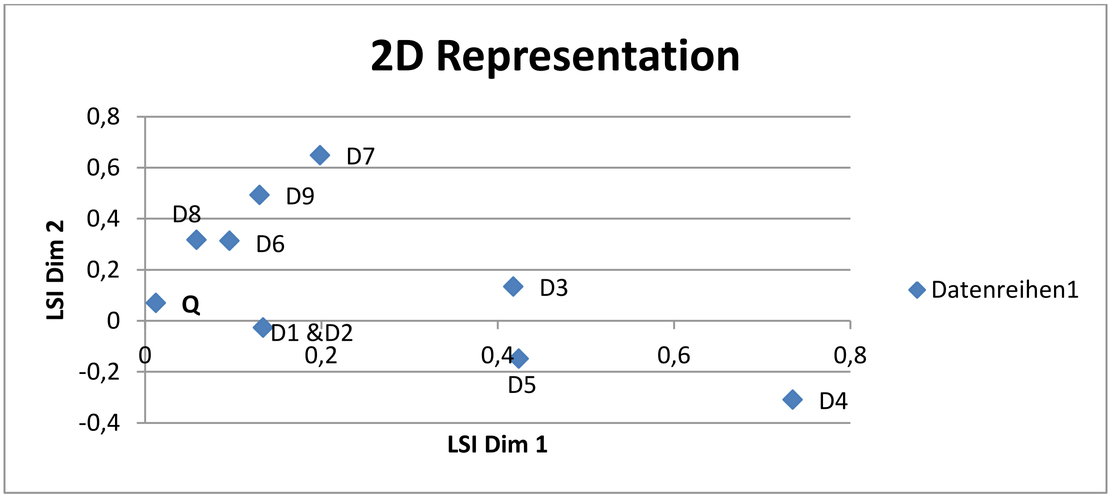


Figure 4.6: Documents Projected into 2D Semantic Space

Analysis of the Experiment Result

Query term: “bank”

From **experiment 1** and **experiment 2**, it is clear that VSM would return documents only when documents share one or more terms with the query given. So, it is obvious that when applying VSM for this query. It would have returned documents D6 and D8 because they share the term **“bank”** with the query and missed D7 and D9 which are also relevant in meaning (i.e polysemy)

As depicted in table 4.56, LSI has been applied using 2 factors, i.e. A_2 is used to approximate the original 8×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 4 documents from D6 to D9 have been returned. But out of these, only D6, D8 and D9 are relevant which shows true polysemy meaning to the query. But other document D7 is irrelevant to the query in polysemy sense though posses the well cosine scores. Hence, it can be concluded that LSI approach does not work well for the polysemy or partial works for polysemy.

As discussed earlier in **section 2.3 (3)** that choosing the number of dimensions (k) for A_k is an interesting problem and affects the performance of LSI, the present study has made various test on different number of k factors for the same query (i.e. ***“bank”***). The choosing of different number of k factors and their result are presented below.

2. A_4 Approximation (i.e. 4D-Representation)

Query term: “bank”

Query Vector:

```
          0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
                        0.0000 0.0785 0.0000 0.2703
```

Output Generated

For the query **"bank"**, following result is obtained using the data set mentioned in 4.42 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	0.0326	Yes
D7	-0.369	No
D8	0.9914	Yes
D9	0.5873	Yes

Table 4.57: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.57 and non-ranked documents are not presented.

D8	The bank finance institute.
D9	Finance institute or sides of a river.
D6	The river bank.

Table 4.58: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.58, LSI has been applied using 4 factors, i.e. A_4 is used to approximate the original 8×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 3 documents D6, D8 and D9 have been returned. All of them are relevant which shows true polysemy meaning to the query. So, A_4 is the best approximation for this query.

3. A_6 Approximation (i.e. 6D-Representation)

Query term: **"bank"**

Query Vector:

```
          0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000
```

Query Vector Coordinates:

```
                 0.0000 0.0785 0.0000 0.2703 0.7388 0.0000
```

Output Generated

For the query **"bank"**, following result is obtained using the data set mentioned in 4.44 is presented below.

Doc. Id	Cosine Scores	Relevancy
D1	0	No
D2	0	No
D3	0	No
D4	0	No
D5	0	No
D6	0.8717	Yes
D7	-0.243	No
D8	0.4879	Yes
D9	-0.449	Yes but not taken

Table 4.59: Computing Cosine Scores and Marking Document Relevancy

Final retrieved documents with sorted are presented below according to the cosine scores ordered in descending order from Table 4.59 and non-ranked documents are not presented

D6	The river bank.
D8	The bank finance institute.

Table 4.60: Ranked Result of a Query where Non-Ranked Documents are Not Displayed

As depicted in table 4.60, LSI has been applied using 6 factors, i.e. A_6 is used to approximate the original 8×9 term-document matrix **A**. When only positive cosine values are considered (so as to compare with VSM), 2 documents D6 and D8 have been returned which are relevant and shows true polysemy meaning to the query. Also it has missed document D9 which is also relevant in polysemous meaning.

Comparison Table

Table 4.61 lists the LSI-ranked documents with different numbers of factors (k). The documents represented below satisfy a cosine threshold of taking all those values which are positive only. The table below clearly demonstrates that its value associated with returned documents can significantly vary with changes in the number of factors k.

Number of Factors.		
K = 2	K = 4	K = 6
D6	D8	D6
D7	D9	D8
D8	D6	
D9		

Table 4.61: Returned Documents Based on Different Numbers of LSI Factors.

From the above table, it can be concluded that k=4 is best approximation for the given query term ***"*****bank*****"***.

4.3 Precision and Recall Calculation & Analysis

This study has calculated precision/recall and analyzed for both VSM and LSI model for different queries.

Since the result from the above analysis (i.e. **section 4.2**) shows that A_2 is the best approximation for most of the queries for LSI model, precision and recall is calculated for it.

For VSM:

Query terms: ***"human computer tree graph"***

Doc. Id.	Cosine Values	Relevancy	Precision	Recall
1	0.6593	Yes	1/1	1/7
7	0.5898	Yes	2/2	2/7
8	0.4238	Yes	3/3	3/7
6	0.4171	Yes	4/4	4/7
4	0.2808	Yes	5/5	5/7
2	0.2537	Yes	6/6	6/7
9	0.1914	Yes	7/7	7/7
3	0	No		
5	0	No		

Table 4.3.1: Precision Recall

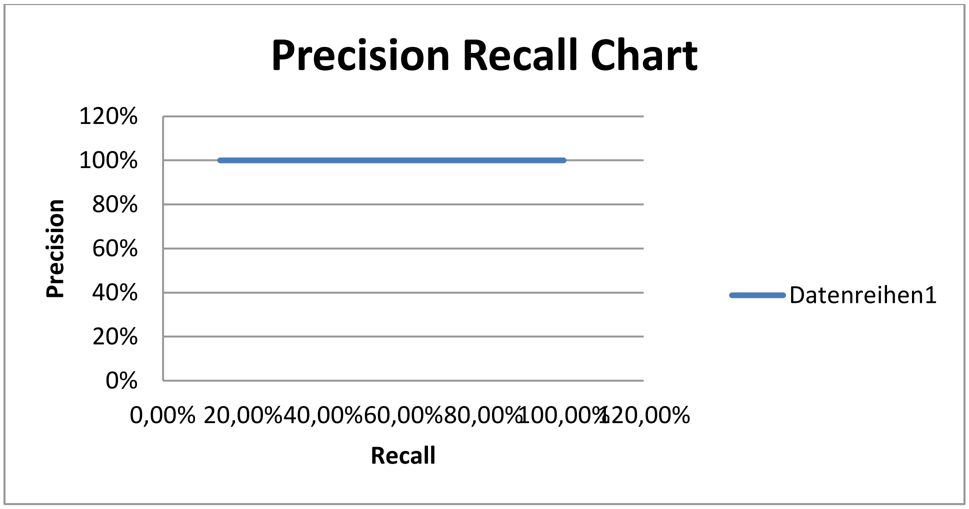


Figure 4.3.1: Graph for Precision Recall

<u>For LSI Model:</u>

<u>Query terms</u> : *"human computer tree graph"* (at k =2)

Doc. Id	Cosine Scores	Relevancy	Precision	Recall
D1	0.8116	Yes	1/1	1/9
D2	0.7892	Yes	2/2	2.9
D3	0.7804	Yes	3/3	3/9
D4	0.6686	Yes	4/4	4/9
D5	0.6155	Yes	5/5	5/9
D9	0.2965	Yes	6/6	6/9
D8	0.0888	Yes	7/7	7/9
D7	0.0675	Yes	8/8	8/9
D6	0.0167	Yes	9/9	9/9

Table 4.3.2: Precision Recall

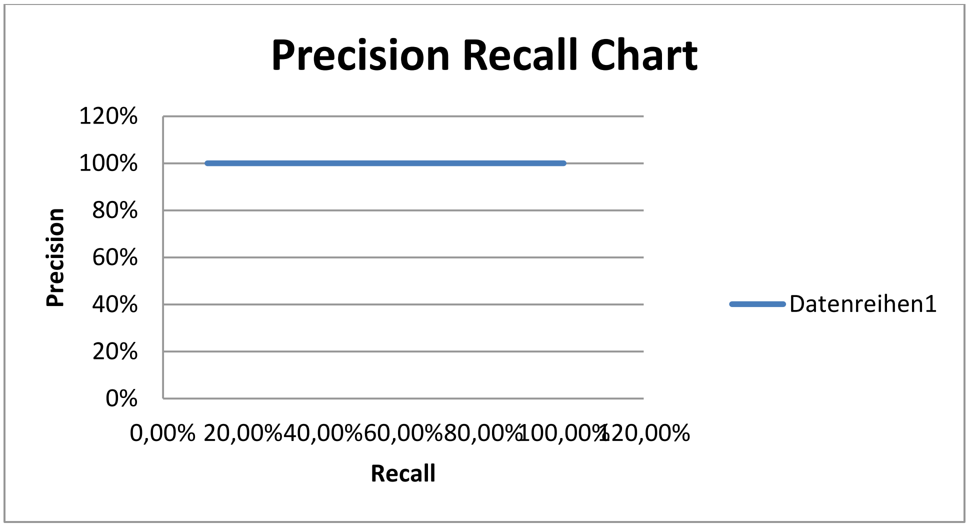


Figure 4.3.2: Graph for Precision Recall

The Figures 4.3.1 and 4.3.2 depict the precision/recall for the query ***"human computer tree graph"*** for VSM and LSI respectively. The graphs show that the precision is 100% at each level of recall which implies that there are no non-relevant documents. At the end, at 100% recall it has 100% precision which is the best state of precision/recall but it happens only in few cases. From these graphs both models seem well but carefully observing the graphs of these two models, LSI has the slightly higher recall than VSM.

For VSM:

Query terms: *"human* computer interaction"

Doc. Id	Cosine Scores	Relevancy	Precision	Recall
D1	0.8165	Yes	1/1	1/3
D4	0.3478	Yes	2/2	2/3
D2	0.3141	Yes	3/3	3/3
D3	0	No		
D5	0	No		
D6	0	No		
D7	0	No		
D8	0	No		
D9	0	No		

Table 4.3.3: Precision Recall

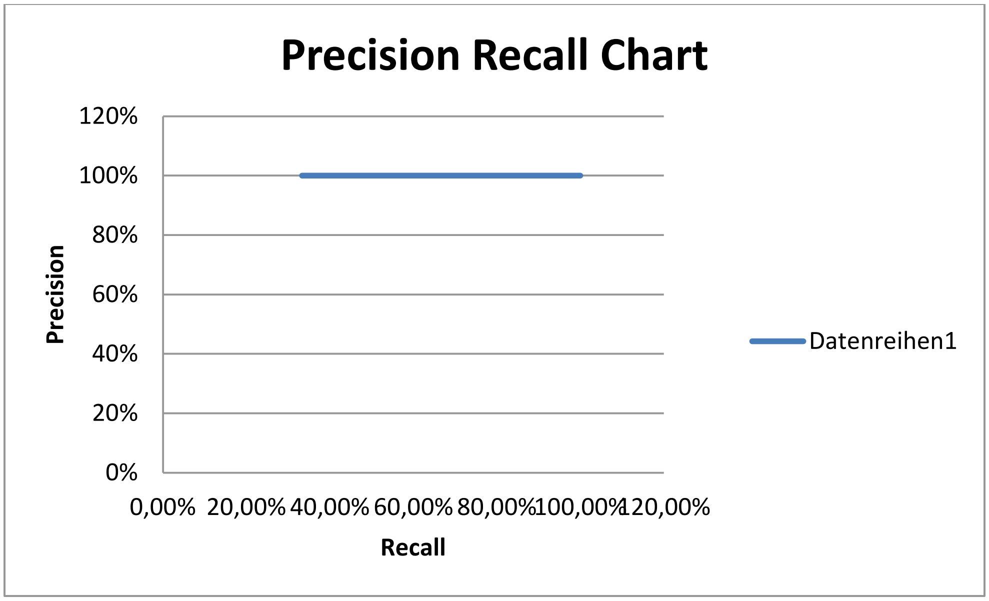


Figure 4.3.3: Graph for Precision Recall

<u>For LSI Model:</u>

<u>Query terms</u>: "human computer interaction" (at k=2)

Doc. Id	Cosine Scores	Relevancy	Precision	Recall
D1	0.986	Yes	1/1	1/5
D3	0.976	Yes	2/2	2/5
D4	0.9278	Yes	3/3	3/5
D2	0.438	Yes	4/4	4/5
D5	0.2054	Yes	5/5	5/5
D9	-0.154	Not taken		
D8	-0.359	Not taken		
D7	-0.379	Not taken		
D6	-0.425	Not taken		

Table 4.3.4: Precision Recall

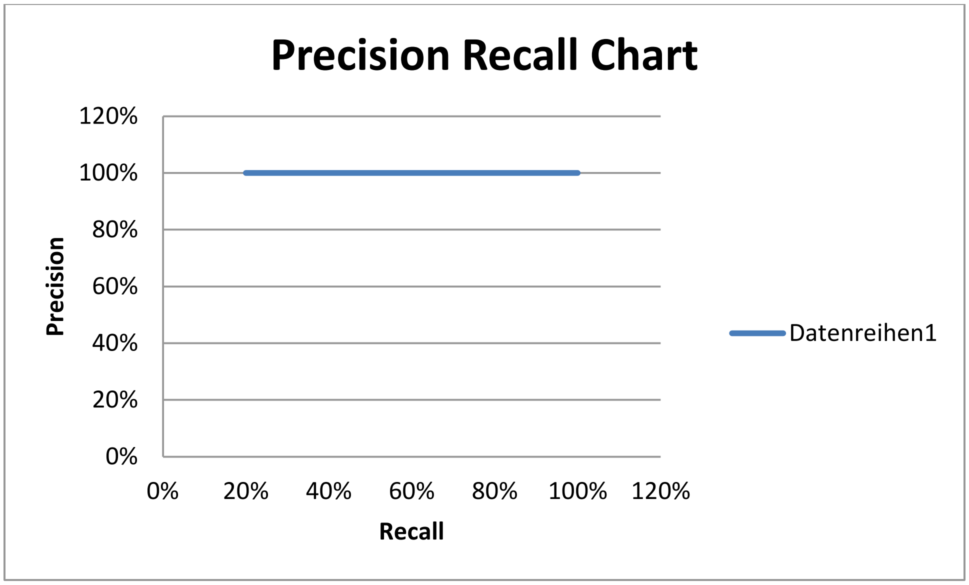


Figure 4.3.4: Graph for Precision Recall

The Figures 4.3.3 and 4.3.4 depict the precision/recall for the query ***"human* computer interaction*"*** for VSM and LSI respectively. The graphs show that the precision is 100% at each level of recall which implies that there are no non-relevant documents. At the end, at 100% recall it has 100% precision which is the best state of precision/recall but it happens only in few cases. From these graphs both models seem well but carefully observing the graphs of these two models, LSI has the slightly higher recall than VSM.

For VSM:

Query term: "humor"

Doc. Id	Cosine Values	Relevancy	Precision	Recall
D5	1	Yes	1/1	1/2
D1	0.8944	Yes	2/2	2/2
D2	0	No		
D3	0	No		
D4	0	No		
D6	0	No		
D7	0	No		
D8	0	No		
D9	0	No		

Table 4.3.5: Precision Recall

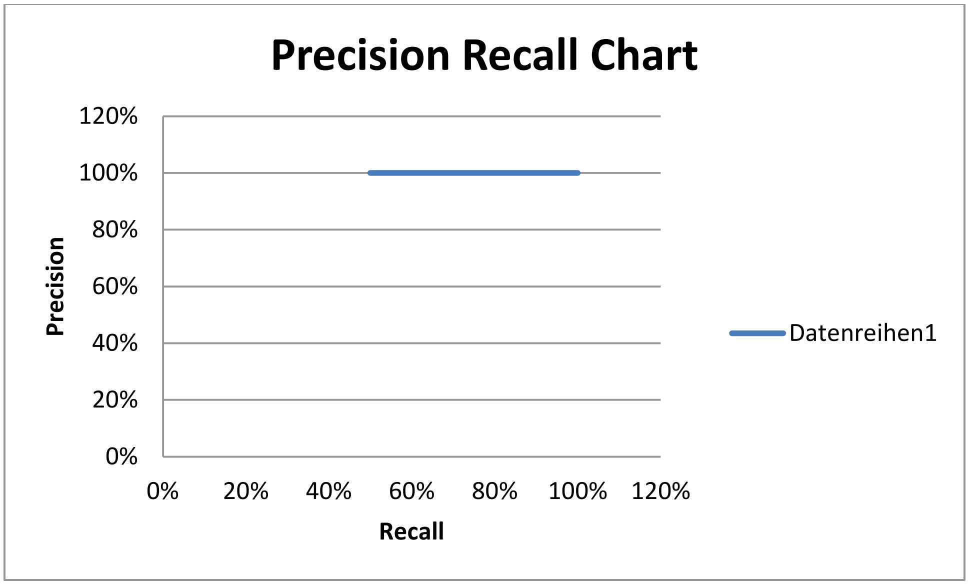


Figure 4.3.5: Graph for Precision Recall

<u>For LSI Model:</u>

<u>Query term</u>: "humor" (at k=2)

Doc. Id	Cosine Scores	Relevancy	Precision	Recall
D1	1	Yes	1/1	1/5
D2	1	Yes	2/2	2/5
D3	1	Yes	3/3	3/5
D4	1	Yes	4/4	4/5
D5	1	Yes	5/5	5/5
D6	0	No		
D7	0	No		
D8	0	No		
D9	0	No		

Table 4.3.6: Precision Recall

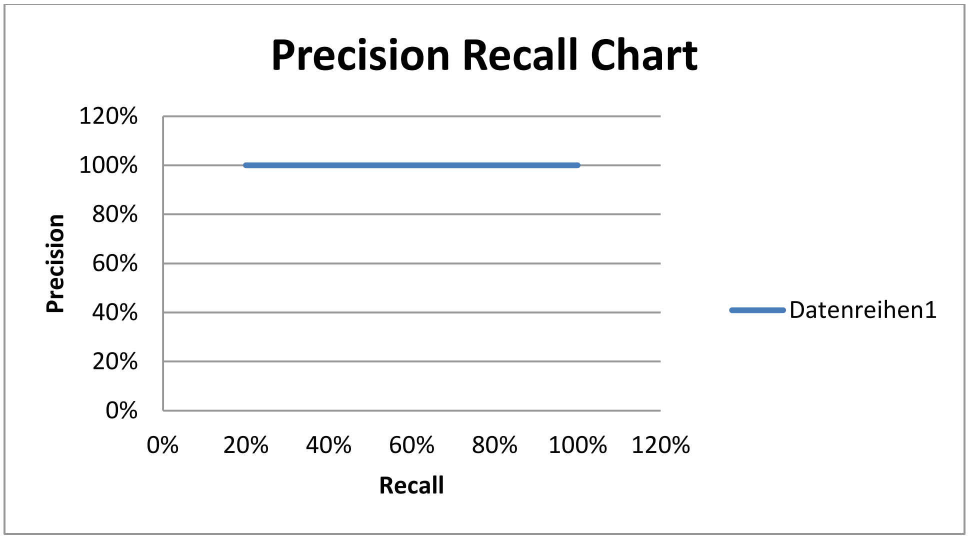


Figure 4.3.6: Graph for Precision Recall

The Figures 4.3.5 and 4.3.6 depict the precision/recall for the query ***"humor"*** for VSM and LSI respectively. The graphs show that the precision is 100% at each level of recall which implies that there are no non-relevant documents. At the end, at 100% recall it has 100% precision which is the best state of precision/recall but it happens only in few cases. From these graphs both models seem well but carefully observing the two graphs it can easily be concluded that the recall is reduced for VSM than LSI for synonyms.

Similarly, precision/recall graph would be obtained for the query term: ***"car"*** for both models.

For VSM:

Query term: "rock"

Doc. Id	Cosine Values	Relevancy	Precision	Recall
D1	0.7071	Yes	1/1	1/2
D2	0.7071	No	1/2	1/2
D3	0.7071	Yes	2/3	2/2
D5	0.5	No	2/4	2/2
D4	0	No		
D6	0	No		
D7	0	No		
D8	0	No		
D9	0	No		

Table 4.3.7: Precision Recall

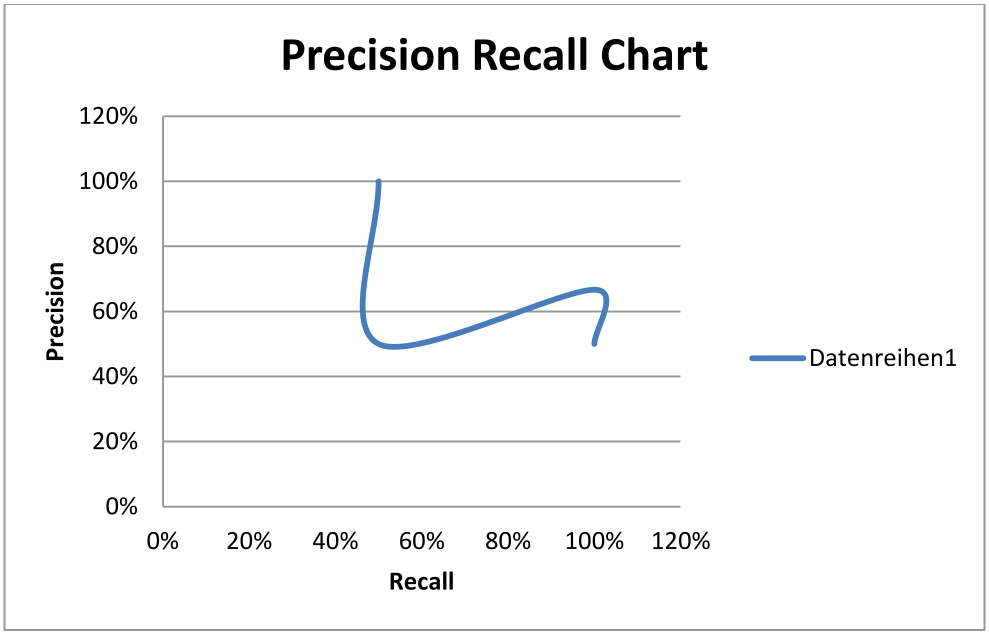


Figure 4.3.7: Graph for Precision Recall

<u>For LSI Model:</u>

<u>Query term</u>: "rock" (at k = 2)

Doc. Id	Cosine Scores	Relevancy	Precision	Recall
D1	1	Yes	1/1	1/3
D2	1	No	1/2	1/3
D3	1	Yes	2/3	2/3
D4	1	Yes	3/4	3/3
D5	1	No	3/5	3/3
D6	0	No		
D7	0	No		
D8	0	No		
D9	0	No		

Table 4.3.8: Precision Recall

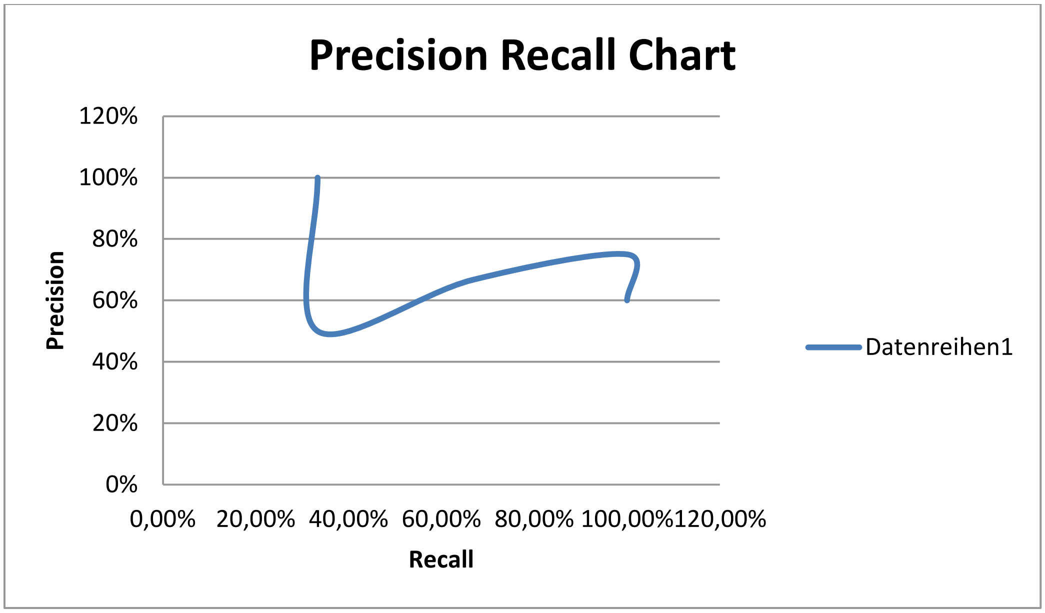


Figure 4.3.8: Graph for Precision Recall

The Figures 4.3.7 and 4.3.8 depict the precision/recall for the query **"rock"** for VSM and LSI respectively. The graphs show that at 50% recall, there is a 100% precision then precision decrease but recall remains constant. At 100% recall, there is 66.67% precision but precision % begins to fall and reaches to 50% while recall remains constant in VSM. In LSI, similar case can be seen but at last precision falls to 60% while recall remains constant at 100%. But carefully observing the two graphs it can easily be concluded that the precision is reduced for VSM than LSI for polysemy.

Similarly, precision/recall graph would be obtained for the query term: ***"bank"*** for both models.

CHAPTER-V

SUMMARY, CONCLUSIONS & FUTURE RECOMMENDATION

This chapter focuses on summarizing the study held with the researcher’s conclusion. The next attempt in this chapter will be made for the future recommendation on the basis of the whole study. For this whole purpose, the chapter is sub divided into Summary, Conclusions and Future Recommendation as following.

5.1 Summary

Beginning with defining of IR, IR is the process of finding information materials of an unstructured nature that satisfies an information need from the large collection of documents. Supporting with this definition, IR is not only the process of retrieving information but at the same time the relevancy of the information that satisfies the user’s need is equally concerned and also the main problem of IR.

So, for the purpose of relevant information retrieval that satisfies the user’s need, several models have been developed. Some are based on set theory concept, some are based on linear algebra concept and some are based on probability concept. All models have their own pros and cons.

The present study has studied the comparative study of two information retrieval models viz. VSM and LSI Model. These models are based on linear algebra concept and have their own advantages and limitations. Those advantages and limitations have been analyzed by implementing these two models and experimenting with several examples. At last, precision/recall have been calculated and analyzed to measure the effectiveness of the two models.

5.2 Conclusions

From the above study, it can be concluded that VSM would return documents only when documents share one or more terms with the query term/s. But LSI creates concepts between query term/s and documents and returned the documents even there is no sharing of term/s with the query term/s. LSI supports well for the synonyms but partially supports for the polysemy but VSM supports only when documents share one or more terms with the query term/s.

When effectiveness of these two models is considered, it can be concluded that recall reduces for synonyms and precision reduces for polysemy when applying the VSM which are the benefits of LSI.

Therefore, from comparative study of these two models, it can be concluded that these two models have their own importance in the information retrieval field but LSI overcomes the problems of VSM and it is better than VSM when observing in overall performance.

5.3 Future Recommendation

This study has concluded that though these two models under study have very vital role in information retrieval field. But LSI seems substantial performance gains over VSM. It well supports for synonyms but only partially supports for the polysemy. This study further recommends for Probabilistic Latent Semantic Indexing (PLSI) Model which is evolved from LSI and adding a sounder probabilistic model.

PLSI [21] is a novel approach to automated document indexing and has a solid statistical foundation, since it is based on the likelihood principle and defines a proper generative model of the data and directly minimizes word perplexity. It is a better model for capturing polysemy and synonymy than Latent Semantic Indexing (LSI). It has substantial performance gains over direct term matching methods as well as over LSI. The benefits of PLSI which achieves significant gains in precision over both, standard term matching and LSI.

References

[1] C., D., Manning et al., Introduction to Information Retrieval. Cambridge University Press Edition, 2008.

[2] C., R., Douglass et al., Scatter/gather: A cluster-based approach to browsing large document collections. In Proceedings of ACM/SIGIR Conference, 1992.

[3] Goker, A. and Davies, J., Information Retrieval: Searching in the 21st Century. John Wiley and Sons Ltd., November 2009.

[4] http://en.wikipedia.org/wiki/Information_retrieval

[5] J., Han and M., Kamber , Data Mining: Concepts and Techniques. 2nd Edition, Morgan Kaufmann Publishers, 2006.

[6] N., P., Gopalan and B., Sivaselvan, Data Mining: Techniques and Trends. Prentice-Hall of India Pvt.Ltd., New Delhi, 2009.

[7] Dr. E., Garcia, The Classic Vector Space Model: Description, Advantages and Limitations of the Classic Vector Space Model. 2006.
http://www.miislita.com/term-vector/term-vector-3.html

[8] B., Rosario, Latent Semantic Indexing: An overview. INFOSYS 240 Spring 2000 Final Paper.

[9] M., W., Berry et al., Using Linear Algebra for Intelligent Information Retrieval. SIAM Review, 37(4), 1995, 573-595.

[10] T., K., Landauer et al, Introduction to Latent Semantic Analysis. Discourse Processes 25(2-3): pages 259–284, 1998.

[11] K., Aberer, Information Retrieval. EPFL-IC, Laboratoire de systèmes d'informations répartis, 2004/5.

[12] S., Gerard and M., J., Michael, Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[13] S., Dipesh, Text Mining with Lucene and Hadoop: Document Clustering with Feature Extraction. Thesis, Wakhok University, 2009.

[14] C., Erica and K., G., Tamara, New Term Weighting Formulas For The Vector Space Method In Information Retrieval. Computer Science and Mathematics Division, Oak Ridge National Laboratory, US 1999.

[15] S., Gerard, Automatic Text Processing. Addison-Wesley Longman Publishing Co., Inc., Boston, 1988.

[16] Dr. E. Garcia, Cosine Similarity and Term Weight. Mi Islita, 2006. http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html

[17] G., Golub, and W., Kahan, Calculating the Singular Values and Pseudo-Inverse of a Matrix. J.SIAM, Numer. Anal. Ser. B., Pages: 205-224, Vol 2, Issue No. 2, 1965.

[18] Dr. E., Garcia, **SVD and LSI Tutorial 1: Understanding SVD and LSI**. Mi lslita, 2005. http://www.miislita.com/information-retrieval-tutorial/svd-lsi-tutorial-1-understanding.html

[19] Dr. E., Garcia, **Document Indexing Tutorial: Document Indexing Tutorial for Information Retrieval Students and Search Engine Marketers**. Mi lslita, 2005. http://www.miislita.com/information-retrieval-tutorial/indexing.html

[20] Manandhar, R., Implementation of Document Ranking in Vector Space Model for Unstructured Documents. Unpublished Master's Thesis , T.U., Kirtipur, 2011.

[21] T., Hofmann, Probabilistic Latent Semantic Indexing. Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval, 1999.

Coding for VSM

```
long int nout=100;       /* print informations every nout tokens processed */
long int nsave=5000;     /* save every nsave tokens processed */
int nocase=1;
 /* ignore case */
int precision=2;
 /* no. of digits for printing matrix */
#include <stdio.h>
#include <string.h>
#include <math.h>
#define BUFLEN 4096
#define WLEN 30
double totall=0.;
struct st_node {
  char wrd[WLEN];
long int  lbl;
  int tf; /* term freq (n occurrences in a doc) */
int df; /* document freq (n docs with occurrence) */
 int cf; /* collection freq (total occurrences in coll) */

 struct st_node *l, *r;
int isfirst;
};
long int ucnt, rdcnt, wcnt;
long int ix;
```

```
double entr;

FILE *ftp;

char fln[BUFLEN];

char fdn[]="dict.a";

FILE *fdp;

char fmn[]="tdm.dat";

FILE *fmp;

char w[WLEN];

char **dcv;

long int dc=1;

long int dcn=0;

double *v;

struct st_node * newnode(void);

struct st_node * deltree(struct st_node *);

void printtree(FILE *,struct st_node *);

void addcode(const char *, struct st_node **);

long int encode(const char *, struct st_node **);

double entropy(struct st_node*);

long int words(struct st_node*);


void docstart(struct st_node*);

char** getdocnames(char*fln);

void dict2vect(struct st_node *,double *);

double *vect(int);

char *string(int);
```

```
main(int argc, char *argv[])
{
  struct st_node *dicttree=NULL;
long int code;
/* start command line */
if (argc>2)
    fprintf(stderr,"WARNING - extra strings on command line\n");
 else if (argc==1)
    fprintf(stderr,"ERROR - no filename given on command line\n"),exit(-3);
 strcpy(fln,argv[1]);
/* end command line */
/* start initializing algorithm */
 rdcnt=0;
  ucnt=0;
  dcv=getdocnames(fln);
/* end initializing algorithm */
 for(dc=1;dc<=dcn;dc++) {
/* start read file into binary tree */
    docstart(dicttree);


   if ((ftp=fopen(dcv[dc],"r"))==NULL)
      fprintf(stderr,"ERROR #1 - system reports:\n"),perror(dcv[dc]), exit(-4);
    while (getword(&w,ftp)) {
      rdcnt++;
      addcode(w,&dicttree);
  if (rdcnt%nout==0)
 {
      fprintf(stderr,"\r        %12li read - %12li unique      MEM %.1lf KB ",
        rdcnt,ucnt,totall);
/*DEBUG*/
fprintf(stderr,"");
   }
    if (rdcnt%nsave==0) {
      fprintf(stderr,"\r(saving...)");
    if ((fdp=fopen(fdn,"w"))==NULL)
        fprintf(stderr,"ERROR #2 - system reports:\n"),perror(fdn), exit(-4);
  printtree(fdp,dicttree);
      fclose(fdp);


    }


}


 fclose(ftp);




 fprintf(stderr,"\n");
/* end read file into binary tree */
}
/* start write tree on file */
```

```
if (rdcnt<=0)
    fprintf(stderr,"ERROR - no words found\n"), exit(-5);
 else
    fprintf(stderr,"\rTotal %li tokens read - %li unique words\n",rdcnt,ucnt);
 fprintf(stderr,"Total memory allocated:  %.0lf kilobytes\n",totall);
 if ((fdp=fopen(fdn,"w"))==NULL)
    fprintf(stderr,"ERROR #3 - system reports:\n"),perror(fdn), exit(-4);
 wcnt=words(dicttree);
 entr=entropy(dicttree);
  fprintf(fdp,"STORED   %10li\n",wcnt);
 fprintf(fdp,"UNIQUE   %10li\n",ucnt);
  fprintf(fdp,"ENTROPY  %.2lf BITS \n",entr);
  fprintf(fdp,"Term-------------------------- -----Label ---df ---cf\n");
printtree(fdp,dicttree);
 fclose(fdp);
/* end write tree on file */
/* start initialize term-document matrix file */
  v=vect(ucnt);
 fprintf(stderr,"\nAllocated document vector - Total memory now %.0lf KB\n\n",
    totall);
  if ((fmp=fopen(fmn,"w"))==NULL)
    fprintf(stderr,"ERROR #5 - system reports:\n"), perror(fmn), exit(-4);
  fprintf(stderr,"\nPrinting term-document matrix...");
  fprintf(fmp,"%li\n", dcn);
  fprintf(fmp,"%li\n", ucnt);
```

```
/* end initialize term-document matrix file */
 for(dc=1;dc<=dcn;dc++) {
/* start convert file into document vector */
    docstart(dicttree);
   fprintf(stderr,"\rReading document %6i...",dc);
  if ((ftp=fopen(dcv[dc],"r"))==NULL)
      fprintf(stderr,"ERROR #4 - system reports:\n"),perror(dcv[dc]), exit(-4);
    wcnt=0;
    while (getword(&w,ftp))
      encode(w,&dicttree),wcnt++;
   fclose(ftp);
    fprintf(stderr,"converting to vector...");
    ix=1;
    dict2vect(dicttree,v);
  fprintf(stderr,"done\n");
/* end convert file into document vector */
/* start print vector into term-document matrix */
  for(ix=1;ix<=ucnt;ix++) {
      fprintf(fmp,"%.*lg%c",precision,v[ix],ix==ucnt?'\n':' ');

   }
    fflush(fmp);
/* end print vector into term-document matrix */
 }
  fclose(fmp);
  fprintf(stderr,"...done\n");
  deltree(dicttree);
  fprintf(stderr,"END OF RUN\n");
}

void docstart(struct st_node *t)
{
  if (t==NULL) return;
 docstart(t->l);
  t->tf=0;
  t->isfirst=1;
  docstart(t->r);
}

void dict2vect(struct st_node *t, double *v)
{
  if (t==NULL) return;
 dict2vect(t->l,v);
 v[ix++]=
    0?0.:(double)t->tf*log((double)dcn/(double)t->df)/(double)wcnt;
  dict2vect(t->r,v);

}

struct st_node * newnode(void)
```

```
{
  struct st_node *tmp;
 const int memsize=1;//100;
  static int imem=0;
  static struct st_node *mem=NULL;
  if (imem==0) {
    mem=(struct st_node *)malloc(memsize*sizeof(struct st_node));
  imem=memsize;
    if (mem==NULL)
      fprintf(stderr,"ERROR - memory allocation\n"),exit(-1);
   totall+=memsize*(double)sizeof(struct st_node)/1000.;
 }
  tmp=mem+(--imem);
  tmp->l=tmp->r=NULL;
  ucnt++;
  tmp->lbl=ucnt;
  return tmp;

}


struct st_node *deltree(struct st_node *t)
{
  if (t->l!=NULL) deltree(t->l);
  if (t->r!=NULL) deltree(t->r);
free(t);
```

```
  ucnt--;
 return NULL;
}

void addcode(const char *s, struct st_node **t)
{
  int cmp;
  if ((*t)==NULL) {
    *t=newnode();
    strcpy((*t)->wrd,s);
    (*t)->cf=1;
    (*t)->df=1;
    return;
  }
  cmp=strcmp(s,(*t)->wrd);
 if (cmp==0) {
    (*t)->cf++;
    if((*t)->isfirst){
      (*t)->df++;
      (*t)->isfirst=0;
    }

  return;

 }
```

```
  if (cmp>0)
    return addcode(s,&((*t)->r));

 if (cmp<0)
    return addcode(s,&((*t)->l));

}

long int encode(const char *s, struct st_node **t)
{
  int cmp;



 if ((*t)==NULL) {
    fprintf(stderr,"ERROR: Trying to encode but dict tree is empty\n");

  exit(-10);
  }
  cmp=strcmp(s,(*t)->wrd);
  if (cmp==0) {
    (*t)->tf++;
    return (*t)->lbl;

  }
  if (cmp>0)
```

```
    return encode(s,&((*t)->r));

 if (cmp<0)
    return encode(s,&((*t)->l));
}

void printtree(FILE *fp, struct st_node *t)

{
  if (t==NULL)
   return;

 else {
    printtree(fdp,t->l);
   fprintf(fp,"%-30s %10li %5i %5i\n",t->wrd, t->lbl, t->df, t->cf);
   printtree(fdp,t->r);

 }
}

int getword(char *wbuf,FILE *fp)
{
  char c;
  int i;
/* start leggi token */
```

```
START:
  i=0;
  do {
    c=fgetc(fp);
  } while(!isalpha(c) && !feof(fp));

if (feof(fp))
    return 0;

 do {
    wbuf[i++]=c;
  if (i>BUFLEN)
      fprintf(stderr,"ERROR: buffer overflow\n"),exit(-2);
   c=fgetc(fp);

 } while(isalpha(c) && !feof(fp));
  wbuf[i]='\0';
/* end leggi token */
  if(nocase)strlwr(wbuf);
/* start stoplisting */

 if (i<=2)
    goto START;
/* end stoplisting */
/* start stemming */
```

```
  stripsuffix("ing",w);
  stripsuffix("ities",w);
 stripsuffix("ity",w);
  stripsuffix("ly",w);
  stripsuffix("ed",w);
  stripsuffix("s",w);
 stripsuffix("e",w);
/* end stemming */
 return 1;
}
int stripsuffix(const char *s, char *w)
{
  int i,j;



  i=strlen(w);
  j=strlen(s);
  while (i>0 && j>0 && w[i]==s[j]) {
    i--, j--;


 }
  if (i>0 && j==0 && w[i]==s[j])
    w[i]='\0';
  return 1;
}
```

```
long int words(struct st_node *t)
{
  long int val;
  if (t==NULL)
    return 0.;
  val = t->cf;
  val=val+words(t->l);
  val=val+words(t->r);
  return val;
}

double entropy(struct st_node *t)
{
  double val=0;

  if (t==NULL)
    return 0.;
val=(double)(t->cf)/(double)wcnt;
  val=-val*log2(val);
  val=val+entropy(t->l);
  val=val+entropy(t->r);
  return val;
}
char** getdocnames(char*fln)
{
  char **dcv;
  char nbuf[BUFLEN];
  FILE *flp;
 if ((flp=fopen(fln,"r"))==NULL)
    fprintf(stderr,"ERROR #6 - system reports:\n"), perror(fln), exit(-4);
  if(fgets(nbuf,BUFLEN,flp)==NULL)
 {
    fprintf(stderr,"ERROR - While scanning the list of document names:\n");
   fprintf(stderr,"Document name list ended prematurely\n");
  exit(-9);
 }
  if(sscanf(nbuf,"%li",&dcn)==0)
{
    fprintf(stderr,"ERROR - While scanning the list of document names:\n");
   fprintf(stderr,"The list should start with the number of documents\n");
 exit(-7);
 }
  if(dcn==0) {
    fprintf(stderr,"ERROR - While scanning the list of document names:\n");
  fprintf(stderr,"The number of documents should be > 0\n");
 exit(-8);
  }
  dcv=(char**)malloc((1+dcn)*sizeof(char*));
 totall+=(double)((1+dcn)*sizeof(char*))/1000.;
  for(dc=1;dc<=dcn;dc++)
```

```
{
    if(fgets(nbuf,BUFLEN,flp)==NULL)
  {
      fprintf(stderr,"ERROR - While scanning the list of document names:\n");


     fprintf(stderr,"Document name list ended prematurely\n");
      exit(-9);
   }
    dcv[dc]=string(BUFLEN);
   totall+=(strlen(nbuf)*sizeof(char))/1000.;


    if(dcv[dc]==NULL)
      fprintf(stderr,"ERROR - Out of memory for document name list\n"),exit(-1);
   strcpy(dcv[dc],nbuf);
    dcv[dc][strlen(nbuf)-1]='\0';
  }
  return dcv;
}
char *string(int len)
{
  char *s;
  s = (char *)
      malloc((size_t)(len * sizeof(char)));
 if(!s)
    fprintf(stderr,"allocazione non riuscita in string()"),exit(-1);
```

```
else
    totall+=(double)(len * sizeof(char))/1000.;
  return s;
}
double *vect(int len)
{
  double *v;
  v = (double *)
      malloc((size_t) ((len + 1) * sizeof(double)));
  if(!v)
    fprintf(stderr,"allocazione non riuscita in vect()"),exit(-1);
  else
    totall+=(double)((len + 1) * sizeof(double))/1000.;
 return v;
}
```

Coding for LSI

```
/*
 * svdcomp - SVD decomposition routine.
 * Takes an mxn matrix a and decomposes it into udv, where u,v are
 * left and right orthogonal transformation matrices, and d is a
 * diagonal matrix of singular values.
 *

 * Input to dsvd is as follows:
 *   a = mxn matrix to be decomposed, gets overwritten with u
 *   m = row dimension of a
 *   n = column dimension of a
 *   w = returns the vector of singular values of a
 *   v = returns the right orthogonal transformation matrix
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "defs_and_types.h"

static double PYTHAG(double a, double b)
{
   double at = fabs(a), bt = fabs(b), ct, result;

   if (at > bt)       { ct = bt / at; result = at * sqrt(1.0 + ct * ct); }
   else if (bt > 0.0) { ct = at / bt; result = bt * sqrt(1.0 + ct * ct); }
   else result = 0.0;
   return(result);
}


int dsvd(float **a, int m, int n, float *w, float **v)
{
   int flag, i, its, j, jj, k, l, nm;
   double c, f, h, s, x, y, z;
   double anorm = 0.0, g = 0.0, scale = 0.0;
   double *rv1;

   if (m < n)
   {
      fprintf(stderr, "#rows must be > #cols \n");
      return(0);
   }
```

```
rv1 = (double *)malloc((unsigned int) n*sizeof(double));

for (i = 0; i < n; i++)
{
   /* left-hand reduction */
   l = i + 1;
   rv1[i] = scale * g;
   g = s = scale = 0.0;
   if (i < m)
   {
      for (k = i; k < m; k++)
         scale += fabs((double)a[k][i]);
      if (scale)
      {
         for (k = i; k < m; k++)
         {
            a[k][i] = (float)((double)a[k][i]/scale);
            s += ((double)a[k][i] * (double)a[k][i]);
         }
         f = (double)a[i][i];
         g = -SIGN(sqrt(s), f);
         h = f * g - s;
         a[i][i] = (float)(f - g);
         if (i != n - 1)
         {
            for (j = l; j < n; j++)
            {
               for (s = 0.0, k = i; k < m; k++)
                  s += ((double)a[k][i] * (double)a[k][j]);
               f = s / h;
               for (k = i; k < m; k++)
                  a[k][j] += (float)(f * (double)a[k][i]);
            }
         }
         for (k = i; k < m; k++)
            a[k][i] = (float)((double)a[k][i]*scale);
      }
   }
   w[i] = (float)(scale * g);

   /* right-hand reduction */
   g = s = scale = 0.0;
   if (i < m && i != n - 1)
   {
      for (k = l; k < n; k++)
         scale += fabs((double)a[i][k]);
      if (scale)
      {
         for (k = l; k < n; k++)
         {
            a[i][k] = (float)((double)a[i][k]/scale);
            s += ((double)a[i][k] * (double)a[i][k]);
         }
         f = (double)a[i][l];
         g = -SIGN(sqrt(s), f);
         h = f * g - s;
         a[i][l] = (float)(f - g);
         for (k = l; k < n; k++)
            rv1[k] = (double)a[i][k] / h;
         if (i != m - 1)
         {
            for (j = l; j < m; j++)
            {
               for (s = 0.0, k = l; k < n; k++)
                  s += ((double)a[j][k] * (double)a[i][k]);
               for (k = l; k < n; k++)
                  a[j][k] += (float)(s * rv1[k]);
            }
         }
         for (k = l; k < n; k++)
            a[i][k] = (float)((double)a[i][k]*scale);
      }
   }
   anorm = MAX(anorm, (fabs((double)w[i]) + fabs(rv1[i])));
}

/* accumulate the right-hand transformation */
for (i = n - 1; i >= 0; i--)
{
   if (i < n - 1)
   {
      if (g)
      {
         for (j = l; j < n; j++)
           v[j][i] = (float)(((double)a[i][j] / (double)a[i][l]) /g));
            /* double division to avoid underflow */
         for (j = l; j < n; j++)
         {
            for (s = 0.0, k = l; k < n; k++)
               s += ((double)a[i][k] * (double)v[k][j]);
            for (k = l; k < n; k++)
               v[k][j] += (float)(s * (double)v[k][i]);
            }
        }
        for (j = l; j < n; j++)
            v[i][j] = v[j][i] = 0.0;
    }
    v[i][i] = 1.0;
    g = rv1[i];
    l = i;
}

/* accumulate the left-hand transformation */
for (i = n - 1; i >= 0; i--)
{
    l = i + 1;
    g = (double)w[i];
    if (i < n - 1)
        for (j = l; j < n; j++)
            a[i][j] = 0.0;
    if (g)
    {
        g = 1.0 / g;
        if (i != n - 1)
        {
            for (j = l; j < n; j++)
            {
                for (s = 0.0, k = l; k < m; k++)
                    s += ((double)a[k][i] * (double)a[k][j]);
                f = (s / (double)a[i][i]) * g;
                for (k = i; k < m; k++)
                    a[k][j] += (float)(f * (double)a[k][i]);
            }
        }
        for (j = i; j < m; j++)
            a[j][i] = (float)((double)a[j][i]*g);
    }
    else
    {
        for (j = i; j < m; j++)
            a[j][i] = 0.0;
    }
    ++a[i][i];
}

for (k = n - 1; k >= 0; k--)
{                              /* loop over singular values */
    for (its = 0; its < 30; its++)
```

```
{                    /* loop over allowed iterations */
   flag = 1;
   for (l = k; l >= 0; l--)
   {                 /* test for splitting */
      nm = l - 1;
      if (fabs(rv1[l]) + anorm == anorm)
      {
         flag = 0;
         break;
      }
      if (fabs((double)w[nm]) + anorm == anorm)
         break;
   }
   if (flag)
   {
      c = 0.0;
      s = 1.0;
      for (i = l; i <= k; i++)
      {
         f = s * rv1[i];
         if (fabs(f) + anorm != anorm)
         {
            g = (double)w[i];
            h = PYTHAG(f, g);
            w[i] = (float)h;
            h = 1.0 / h;
            c = g * h;
            s = (- f * h);
            for (j = 0; j < m; j++)
            {
               y = (double)a[j][nm];
               z = (double)a[j][i];
               a[j][nm] = (float)(y * c + z * s);
               a[j][i] = (float)(z * c - y * s);
            }
         }
      }
   }
   z = (double)w[k];
   if (l == k)
   {              /* convergence */
      if (z < 0.0)
      {           /* make singular value nonnegative */
         w[k] = (float)(-z);
         for (j = 0; j < n; j++)
            v[j][k] = (-v[j][k]);
   }
   break;
}
if (its >= 30) {
   free((void*) rv1);
   fprintf(stderr,"No convergence after 30,000! iterations \n");
   return(0);
}

/* shift from bottom 2 x 2 minor */
x = (double)w[l];
nm = k - 1;
y = (double)w[nm];
g = rv1[nm];
h = rv1[k];
f = ((y - z) * (y + z) + (g - h) * (g + h)) / (2.0 * h * y);
g = PYTHAG(f, 1.0);
f = ((x - z) * (x + z) + h * ((y / (f + SIGN(g, f))) - h)) / x;

/* next QR transformation */
c = s = 1.0;
for (j = l; j <= nm; j++)
{
   i = j + 1;
   g = rv1[i];
   y = (double)w[i];
   h = s * g;
   g = c * g;
   z = PYTHAG(f, h);
   rv1[j] = z;
   c = f / z;
   s = h / z;
   f = x * c + g * s;
   g = g * c - x * s;
   h = y * s;
   y = y * c;
   for (jj = 0; jj < n; jj++)
   {
      x = (double)v[jj][j];
      z = (double)v[jj][i];
      v[jj][j] = (float)(x * c + z * s);
      v[jj][i] = (float)(z * c - x * s);
   }
   z = PYTHAG(f, h);
   w[j] = (float)z;
   if (z)
```

```
                {
                    z = 1.0 / z;
                    c = f * z;
                    s = h * z;
                }
                f = (c * g) + (s * y);
                x = (c * y) - (s * g);
                for (jj = 0; jj < m; jj++)
                {
                    y = (double)a[jj][j];
                    z = (double)a[jj][i];
                    a[jj][j] = (float)(y * c + z * s);
                    a[jj][i] = (float)(z * c - y * s);
                }
            }
            rv1[l] = 0.0;
            rv1[k] = f;
            w[k] = (float)x;
        }
    }
    free((void*) rv1);
    return(1);
}
```

transpose of matrix

```
#include<stdio.h>



main()

{

  int m, n, c, d, matrix[10][10], transpose[10][10];



  printf("Enter the number of rows and columns of matrix ");

  scanf("%d%d",&m,&n);

  printf("Enter the elements of matrix \n");



  for( c = 0 ; c < m ; c++ )
```

```
{
  for( d = 0 ; d < n ; d++ )
  {
    scanf("%d",&matrix[c][d]);
  }
}

for( c = 0 ; c < m ; c++ )
{
  for( d = 0 ; d < n ; d++ )
  {
    transpose[d][c] = matrix[c][d];
  }
}

printf("Transpose of entered matrix :-\n");

for( c = 0 ; c < n ; c++ )
{
  for( d = 0 ; d < m ; d++ )
  {
    printf("%d\t",transpose[c][d]);
  }
  printf("\n");
}
```

```
  return 0;

}
```

float mat mult

```
#include <stdio.h>
#include <stdlib.h>

/* Make the data structure self-contained.  Element at row i and col j
   is x[i * w + j].  More often than not, though,  you might want
   to represent a matrix some other way */
typedef struct { int h, w; double *x;} matrix_t, *matrix;

inline double dot(double *a, double *b, int len, int step)
{
        double r = 0;
        while (len--) {
                r += *a++ * *b;
                b += step;
        }
        return r;
}

matrix mat_new(int h, int w)
{
        matrix r = malloc(sizeof(matrix) + sizeof(double) * w * h);
        r->h = h, r->w = w;
        r->x = (double*)(r + 1);
        return r;
}

matrix mat_mul(matrix a, matrix b)
{
        matrix r;
        double *p, *pa;
        int i, j;
        if (a->w != b->h) return 0;

        r = mat_new(a->h, b->w);
        p = r->x;
        for (pa = a->x, i = 0; i < a->h; i++, pa += a->w)
                for (j = 0; j < b->w; j++)
```

```
                        *p++ = dot(pa, b->x + j, a->w, b->w);
        return r;
}

void mat_show(matrix a)
{
        int i, j;
        double *p = a->x;
        for (i = 0; i < a->h; i++, putchar('\n'))
                for (j = 0; j < a->w; j++)
                        printf ("\t%7.3f", *p++);
        putchar('\n');
}

int main()
{
        double da[] = {  1, 1,  1,   1,
                         2, 4,  8,  16,
                         3, 9, 27,  81,
                         4,16, 64, 256      };
        double db[] = { 4.0,   -3.0,  4.0/3,
                        -13.0/3, 19.0/4, -7.0/3,
                          3.0/2,   -2.0,  7.0/6,
                         -1.0/6,  1.0/4, -1.0/6};

        matrix_t a = { 4, 4, da }, b = { 4, 3, db };
        matrix c = mat_mul(&a, &b);

        /* mat_show(&a), mat_show(&b); */
        mat_show(c);
        /* free(c) */
        return 0;
}
```