5-2009

# A study of relevance feedback in vector space model

Deepthi Katta

# A STUDY OF RELEVANCE FEEDBACK IN VECTOR SPACE MODEL

by

Deepthi Katta

Bachelor of Technology in Computer Science and Engineering
Vellore Institute of Technology, India
May 2005

A thesis submitted in partial fulfillment
of the requirements for the

**Master of Science Degree in Computer Science**
**School of Computer Science**
**Howard R. Hughes College of Engineering**

**Graduate College**
**University of Nevada, Las Vegas**
**May 2009**

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

**Thesis Approval**
The Graduate College
University of Nevada, Las Vegas

APRIL 24TH, 2009

The Thesis prepared by

DEEPTHI KATTA

**Entitled**

A STUDY OF RELEVANCE FEEDBACK IN VECTOR SPACE MODEL

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

[redacted]
*Examination Committee Chair*

[redacted]
*Dean of the Graduate College*

[redacted]
*Examination Committee Member*

[redacted]
*Examination Committee Member*

[redacted]
*Graduate College Faculty Representative*

1017-53

ABSTRACT

**A Study of Relevance Feedback in Vector Space Model**

by
Deepthi Katta

Dr. Kazem Taghva, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Information Retrieval is the science of searching for information or documents based on information need from a huge set of documents. It has been an active field of research since early 19th century and different models of retrieval came in to existence to cater the information need.

This thesis starts with understanding some of the basic information retrieval models, followed by implementation of one of the most popular statistical retrieval model known as Vector Space Model. This model ranks the documents in the collection based on the similarity measure calculated between the query and the respective document. The user specifies the "information need" which is more commonly known as a "query" using the visual interface provided. The given query is then processed and the results are displayed to the user in a ranked order.

We then focus on the Relevance feedback, a technique that modifies the user query based on the characteristics of the document collection to improve the results. In this thesis, we explore different types and models of relevance feedback that can be applied to Vector Space model and how they affect the performance of the model.

TABLE OF CONTENTS

LIST OF TABLES

## LIST OF FIGURES

## ACKNOWLEDGEMENTS

I take this opportunity to sincerely thank my advisor, Dr. Kazem Taghva who gave me support and guidance throughout this thesis work. I am greatly indebted to my graduate coordinator Dr. Ajoy K Datta for his help and invaluable support during my masters program. I also would like to thank other members of my committee, Dr. Laxmi P. Gewali and Dr. Venkatesan Muthukumar.

I would like to thank the Department of Computer Science for their financial support through a Teaching Assistantship.

I owe my loving thanks to my husband, Sandeep Sindol and my family for standing by me with all their love and encouragement at times I needed most.

Last but not least, I thank everyone for their support in the successful completion of this work.

# CHAPTER 1

## INTRODUCTION

Information retrieval (IR) is defined as 'finding material of an unstructured nature that satisfies an information need from within large collections' [1]. In other words, it is the science of searching for documents which contain the information required. The emergence of computers had made the task of storing large amounts of information easy. In 1950, the field of information retrieval (IR) was born, since finding the information that is useful and required from such collections had become essential [2].

Data retrieval is a closely related area of Information Retrieval and it is quite often misinterpreted of both being same. The main difference between both of them is that, in data retrieval we usually search for an exact match, that is, we check to see presence or absence of an item in a file. In information retrieval, the main interest would be to find those items that match the request partially or not completely and then filter them to find the best matched items [3].

The most important development in the field of information retrieval was the creation of SMART system at Cornell University by Gerald Salton and his team in 1960. This system was later used by the researchers to come up with new methods and models to increase the search quality. By early 1980's, many information retrieval models were developed and evaluated based on the previous research. In 1990, the worldwide TREC (Text Retrieval Conference) project started which was aimed at the evaluation of methods for querying databases of realistic size and scope. Prior to the establishment of TREC, there were no large test datasets, and information retrieval research was dominated by measured performance on some small databases for which sample queries and relevance judgments were available [4].

A typical information retrieval system would look like in the figure below [5]. Retrieval is initiated by the user entering the query wanting to find documents that match his criteria. Before the retrieval process is initiated, a text model is developed from the document collection by performing text operations such as removing stop words and stemming. The text model is then used to build an index. An index is a critical data structure because it allows faster searching over large volumes of data. Inverted Index is the most popular form of index used in different retrieval models.

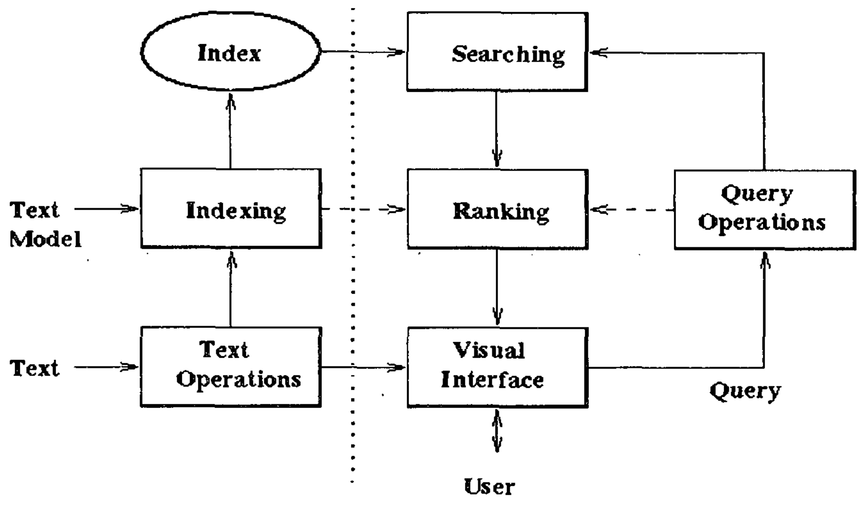


Figure1.1. Retrieval Process

One of the main objectives of this thesis is to understand several information retrieval models that were introduced from the time the concept of retrieval came in to existence to constantly improve the effectiveness of the retrieval and to serve different needs and requirements by the user. We start with one of the earliest models of retrieval called Boolean Retrieval and finish with the latest technique of retrieval popularly known as Language Model. We then implement one of the retrieval models known as Vector Space Model and also try to improve the performance of the same using some query modification techniques.

# CHAPTER 2

# MODELS OF RETREIVAL

## 2.1 Boolean Retrieval

In this type of retrieval, the query is formed using operators such as AND, OR and NOT between the keywords [3]. The documents in this model are viewed as set of keywords. The query is processed using inverted index file which is built for the collection in advance. For each term in the query, the index is searched and the corresponding posting for the term is retrieved. Posting contains the list of documents in which the respective term occurs [1]. Once all the postings for the terms in the query are retrieved, they are merged based on the operator given in the query. Final outcome in this case the list of the documents is displayed to the user. In Boolean retrieval, we deal with the exact match, so, it is often considered as data retrieval model.

**Simple Example of Boolean query processing**

Consider a small document collection of four documents as follows [1]:

| Document ID | Text |
|---|---|
| Doc 1 | new home sales top forecasts |
| Doc 2 | home sales rise in July |
| Doc 3 | increase in home sales in July |
| Doc 4 | July new home sales rise |

Table 2.1.1 Document collection of four documents

The inverted index for the collection is shown in the figure below, sort-based indexing is used for building the index, a common technique in which the terms are sorted and grouped to build the index. The document frequency of each term is also stored on the index. This information is used to minimize the amount of temporary memory space during query processing. In the figure, the left side shows all the terms which is also called as dictionary and the right hand side shows the postings.

Let us consider the following Boolean query and see how the result will be displayed to the end user.

**Example User Boolean Query**: Forecasts AND New

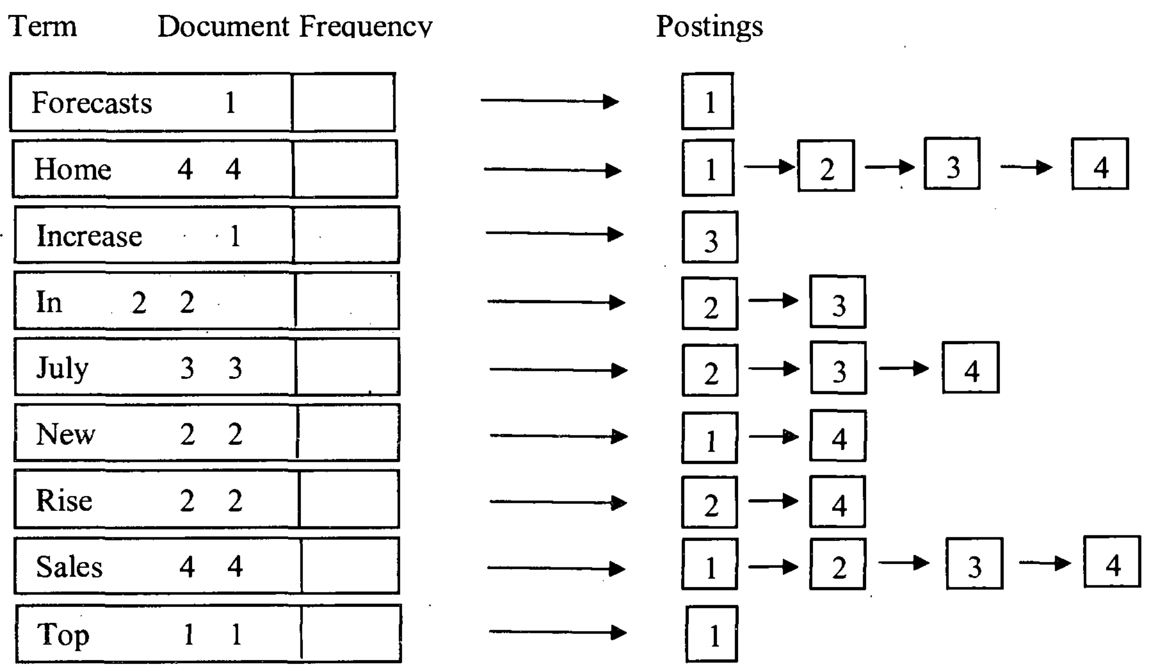


Figure2.1. Inverted Index of collection

First, we need to sort the terms in the query by increasing frequency. In this case, the first term hence would be Forecasts, so, the corresponding posting for the term will be loaded in to the memory. The postings of the remaining terms are compared against the posting in the memory. Since, it is a conjunctive query, the final result must be the list of documents which has all the terms in the query. In this case, the final result would be document 1 since it contains both the terms.

Extended Boolean retrieval models can be built by adding additional operators other than AND, OR and NOT, such as proximity operators which gives how close two terms specified in the query can occur in the document.

Boolean retrieval is preferred by users who need greater control over the retrieved results. Many users use them as it is easy to understand especially for simple queries. But, the model fails to provide the user with some of the additional details or features which will help the user cut down time and effort to find the piece of information of interest. For example, it does not use or maintain the information on term frequency which will play an important role in deciding which documents are more relevant to the query. Also, it just retrieves set of matching documents, but the results are not ranked, that is they are in no particular order and user need to browse through all of them to find which one will suit his requirement [6].

## 2.2 Co-ordinate Matching

In this model, documents that contain more number of terms in the query are given more importance than documents which contain few or none of them. In other words, we are calculating the inner product of query and each document both represented in form of n-dimensional vectors, where n is the number of terms in the index and then taking the result as the similarity measure. This introduces the concept of ranking and also flexibility to simple Boolean retrieval. The similarity measure between the query and document in this type of retrieval model is represented as follows [4]

$$M (Q, Dd) = Q.Dd$$

For example, if we consider the same document collection given in Table 2.1.1 and a query "new top". The vector representation of documents and sample query are given in the table below.

| **Doc ID** | **Forecasts** | **Home** | **Increase** | **July** | **New** | **Sales** | **Rise** | **Top** |
|---|---|---|---|---|---|---|---|---|
| Doc 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| Doc2 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| Doc3 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| Doc4 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| Query | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

Table 2.1.2 Vector representation of document collection and sample query

For convenience, I have assumed that stop words have been removed from the document collection. Stop words are the most common words in a text like are, in, and etc.

Now, we can calculate the inner product of query and each document as follows:

M (new top, Doc1) = (0, 0, 0, 0, 1, 0, 0, 1) . (1, 1, 0, 0, 1, 1, 0, 1) = 2

M (new top, Doc4) = (0, 0, 0, 0, 1, 0, 0, 1) . (0, 1, 0, 1, 1, 1, 1, 0) = 1

Similarly, we can calculate for the rest of the documents in the collection. For this example query, the coordinate matching ranking is Doc1 > Doc4 > Doc2 = Doc3 = 0.

The best feature of co-ordinate matching retrieval model is that it is very simple and straight forward as all the required information is in the inverted index. Also, in simplest way possible it introduces ranking, which means that it gives the result to the user's query in form of list of documents, the document with most of the query terms at the top. But, it has three notable drawbacks which are listed below [4]

1. Term frequency is not taken in to consideration, that is, in vector representation we just note if the term is "present" or "not present" using binary notation.

2. Term scarcity defines how important the term might be in describing the document, which is also not taken in to consideration.

3. Long documents might always top the retrieval list since they are likely to have more of most of the query terms when compared to small documents.

To overcome first drawback, we can include the with-in document frequency ($f_{d,t}$) in the vector representation of documents. This will change the inner product similarity formulation as given below. [4]

$$M(Q, Dd) = Q.Dd = \sum_{t \in Q} w_{q,t} . w_{d,t}$$

Where $w_{d,t}$ is the document-term weight for term t in document d. Similarly, $w_{q,t}$ is the weight for query vector.

To tackle the second problem, the weight of the term ($w_{d,t}$) has to be reduced if it appears in many documents. This can be done by incorporating "Inverse document frequency" in to the term weight, which gives more importance or weight to the terms which occur less frequently in the documents and vice versa. Now, weight of the term, $w_t$ can be calculated as

$$w_t = \frac{1}{f_t}$$

Where $f_t$ is the number of documents in which term t occurs. Now, $w_{d,t}$ can be calculated as [4]

$$w_{d,t} = f_{d,t} \times w_t$$

This type of assigning document-term weights is called TF×IDF rule. There are many variant methods available in the literature for calculating document-term weights with different interpretations for relative term frequency and inverse document frequency. One can choose which one to use based on a particular situation.

The last problem can be removed by taking the length of the document, which is count of the terms it contains in to consideration.

## 2.3 Vector Space Model

Vector space model is considered to be a statistical based retrieval model since, it uses statistical information to determine the relevance between the document and the query. In this model, the document is represented as a vector of keywords from the respective document. The corresponding weights for each keyword determine its importance in the document and also in the collection [7]. Similarly, the query is also a vector representation of keywords in the query and also has corresponding weights denoting the importance of the respective keywords in the query.

Figure 2.2 below [8], shows a typical three dimensional index space representation of three documents with three distinct terms. Generally, the index terms are not limited and can be of any magnitude. So, a document in a collection would be a t-dimensional vector where t is the number of distinct terms in the document.

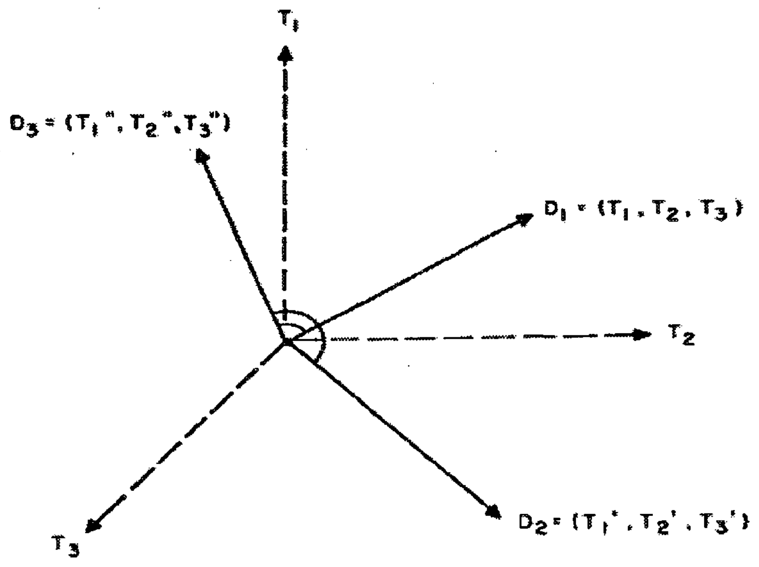


Figure2.2 Vector representation of document space

In a collection, the similarity between the document vector and query vector is measured and the documents are ranked based on the measure. One of the most popular and common way to measure the similarity is known as cosine rule. The logic behind the cosine rule of ranking is that, if we assume a query vector to be starting from the origin in the space in some particular direction, the highest rank should be given to the documents that are closer to the query in angular sense [4]. When two vectors are identical then the angle between them would be zero, then $\cos\Theta = 1$ since $\Theta = 0$. This means that similar documents with the query vector will have higher scores.

The cosine rule for ranking the documents is given below [4].

$$\text{Cosine }(Q, D_d) = \frac{1}{W_q W_d} \sum_{t=1}^{n} w_{q,t} . w_{d,t}$$

Where,

$$W_q = \sqrt{\sum_{t=1}^{n} w^2_{q,t}} \quad \text{and} \quad W_d = \sqrt{\sum_{t=1}^{n} w^2_{d,t}}$$

In the above equations, $w_{q,t}$ and $w_{d,t}$ denote the weights of the terms in the query and the document respectively. There are many different algorithms to weigh these terms and which one to choose depends on the characteristics of the collection [9]. Once the inverted index similar to as shown in the Figure 2.1 is built and the weights of the terms in each document are pre calculated, query weights and cosine measure can then be calculated once the user initiates the query. The results are displayed to the user in descending order of document's cosine measure values.

Vector space model is most admired and widely used because of its simplicity and yet the capability of producing good results. It introduces ranking to the results and also provides partial matching. Even with many advantages of the model over others, it is far from being perfect. One of the main flaws that are observed in vector space model is that it considers all terms to be independent. In other words, the model assumes that the terms do not have any relation between them. This eliminates the two properties, *polysemy* and *Synonymity* in which the terms are related [10].

## 2.4 Probabilistic Model

The 'probabilistic ranking principle' which states that the documents need to be ranked or ordered based on their estimated probability of relevance with respect to the query or the information need is the most fundamental part of probabilistic model of retrieval [11,1]. Many probability retrieval techniques proposed over years have different ways of probability of relevance estimation [2].

**Formal Model**

Two events can be associated for a document query pair. If we name the event as R when document D is relevant to Query Q, then the other event would be a complement of the first, $\sim R$ when document D is not relevant to Q. So, P(R/D) gives the value of probability of relevance of document D. Using Bayes' theorem P(R/D) can be expressed in terms of P(D/R) as follows [12].

$$P(R/D) = \frac{P(D/R)P(R)}{P(D)}$$

To avoid considering the expansion of P(D), we take the log odds instead of odds as given below.

$$\log \frac{P(R/D)}{P(\sim R/D)} = \log \frac{P(D/R)P(R)}{P(D/\sim R)P(\sim R)}$$

Since P(R) and P(R) are just scaling factors they can be ignored in the above equation. Independence assumption is made between the terms in the simplest version of the model, so P(D/R) can be written as a product of each term's probabilities: [2]

$$P\ (D/R) = \prod_{ti \in Q,D} P(ti/R) . \prod_{tj \in Q,D} (1 - P(tj/R))$$

The above equation uses two probabilities; one is the probability of presence of term $t_i$ in relevant documents set. The other is the probability of absence of term $t_j$ in relevant documents set. Here, we consider all the terms which are common to the query and the document.

Substituting the value of P(D/R) in the log of odds equation and also removing constant values for a given query, we get the following ranking function. For further simplification we denote $P(t_i/R)$ as $p_i$ and $P(t_i/\sim R)$ as $q_i$ [2].

$$\log \prod_{ti \in Q,D} \frac{pi.(1-qi)}{qi.(1-pi)}$$

The individual fraction value in the above equation is nothing but the weight of the term $t_i$ in document D.

## 2.5 Language Model

Statistical language models were being used and researched from a very long time. It is the mechanism of generating text and for many years was extensively used in the field of speech recognition. But, language modeling approach to information retrieval was first proposed in 1998. Ponte and Croft were the first ones to propose an idea that language models can be used for an effective retrieval [14].

Language modeling approach to information retrieval is based on the idea that an efficient query can be formulated to get the required results by imagining or guessing which words the relevant documents would contain and then using a set of those words in the query. In probabilistic retrieval model described in the section 2.4, we have seen that the model estimates the probability of relevance of the document with respect to the query and then ranks the documents based on the score. In this model, instead of estimating the probability of relevance, we develop a probabilistic language model called $M_d$ for each document in the collection and the documents are ranked based on the probability of model generating the query [1].

The probability of generating the query Q given the language model $M_d$ , is represented using $P(Q/ M_d)$. The maximum likelihood estimate (MLE) of term t, given the model is given by [14]:

$$P\hat{}_{ml}(t/\ M_d) = \frac{termfrequency\ in\ document(tft,d)}{total\ number\ of\ tokens(dld)}$$

The ranking formula for each document which is $P(Q/\ M_d)$ can be calculated using the following [1]:

$$P\hat{}\ (Q/\ M_d) = \prod_{t \in Q} P\hat{}ml(t/\ Md)$$

The symbol (^) suggests that the model is estimated. One of the important questions here is that what do we do for the terms that have not occurred in the document at all? We definitely do not want to assign $P\hat{}_{ml}(t/\ M_d) = 0$, since if the term did not occur it does not mean that it is not possible, so some weight should be assigned. The answer to this is smoothing of weights [14]. Usually a minimal value is assigned that means that it might still be possible for the term to occur. In other words, if tf $_{(t,d)} = 0$, then we assign

$$P\hat{}_{ml}(t/\ M_d) = \frac{cft}{cs}$$

Where $cf_t$ is term count in the collection and cs is the total number of tokens in the collection. There are a variety of smoothing techniques available for overcoming this practical problem of assigning zero weights [1].

Based on the smoothing method, the probability estimate of generating the query is calculated for each document and they ranked based on that. Ponte and Croft in their experiments have compared their language model with traditional tf idf model on two different query sets and collections. Their experiments showed that the language model outperformed the other in both the cases [14].

# CHAPTER 3

## IMPLEMENTATION OF VECTOR SPACE MODEL

A formal introduction to vector space retrieval model is given in the section 2.3. To get a deeper understanding on how the model works, we consider a collection with small number of documents, a sample query and calculate the weights and corresponding cosine similarity measure to rank the documents. Let us consider the document collection given in Table 3.1.1, which is of four documents, the number of times a term occurs in a respective document, is shown in the brackets for convenience [15]. The document vectors can be constructed in a similar way we constructed vector representation in Table 2.2, but in this case the presence and absence of terms in the documents is replaced by individual term weights. As mentioned earlier, there are many ways to calculate the term weights. Let us suppose, we have chosen the following from the literature to calculate the same.

$w_t = \log_e (1 + N/ft)$ → IDF( Inverse Document Frequency)

$r_{d,t} = 1 + \log_e f_{d,t}$ → Within-document frequency

$r_{q,t} = 1$ → Query term frequency

$w_{d,t} = r_{d,t}$ → Weight of document term t

$w_{q,t} = r_{q,t} \cdot w_t$ → Query term weight

Where,

N – Total number of documents in the collection.

$f_t$ – Number of documents that contain term t.

Now, the document vectors will look like shown in the Table 3.1.2. $W_d$ values in the last column of the table are calculated using the individual $w_{d,t}$ weights of the terms in the document.

| **Document ID** | **Text** |
|---|---|
| Doc 1 | apple(3) balloon(2) elephant(1) |
| Doc 2 | apple(1) balloon(2) chocolate(3) duck(1) |
| Doc 3 | balloon(5) elephant(1) |
| Doc 4 | balloon(1) Chocolate(1) elephant(1) |
| Doc 5 | apple(1) balloon(2) Chocolate(1) |
| Doc 6 | Chocolate(1) elephant(4) |

Table 3.1.1 Document Collection

| **Doc ID** | **Apple** | **Balloon** | **Chocolate** | **Duck** | **Elephant** | **Wd** |
|---|---|---|---|---|---|---|
| Doc1 | 2.0 | 1.69 | 0.0 | 0.0 | 1.0 | 2.80 |
| Doc2 | 1.0 | 1.69 | 2.0 | 1.0 | 0.0 | 2.97 |
| Doc3 | 0.0 | 2.60 | 0.0 | 0.0 | 1.0 | 2.78 |
| Doc4 | 0.0 | 1.0 | 1.0 | 0.0 | 1.0 | 1.73 |
| Doc5 | 1.0 | 1.69 | 1.0 | 0.0 | 0.0 | 2.20 |
| Doc6 | 0.0 | 0.0 | 1.0 | 0.0 | 2.38 | 2.58 |
| $f_t$ | 3 | 5 | 4 | 1 | 4 | |
| $w_t$ | 1.0 | 0.78 | 0.91 | 1.9 | 0.91 | |

Table 3.1.2 Document Vectors

The $W_d$, $w_t$ and $w_{d,t}$ values in the table 3.1.2 are pre-calculated using the information in inverted index , that is before the user is allowed to enter the query, all the information in the table must be ready for access. Table 3.1.3 shows the cosine similarity measure for two sample queries {Duck} and {Duck, Chocolate} on the document collection.

| **Doc ID** | **Duck** $W_q$**=1.9** | **Chocolate, Duck** $W_q$**=2.1** |
|---|---|---|
| Doc1 | 0.0 | 0.0 |
| Doc2 | 0.33 | 0.59 |
| Doc3 | 0.0 | 0.0 |
| Doc4 | 0.0 | 0.25 |
| Doc5 | 0.0 | 0.19 |
| Doc6 | 0.0 | 0.16 |

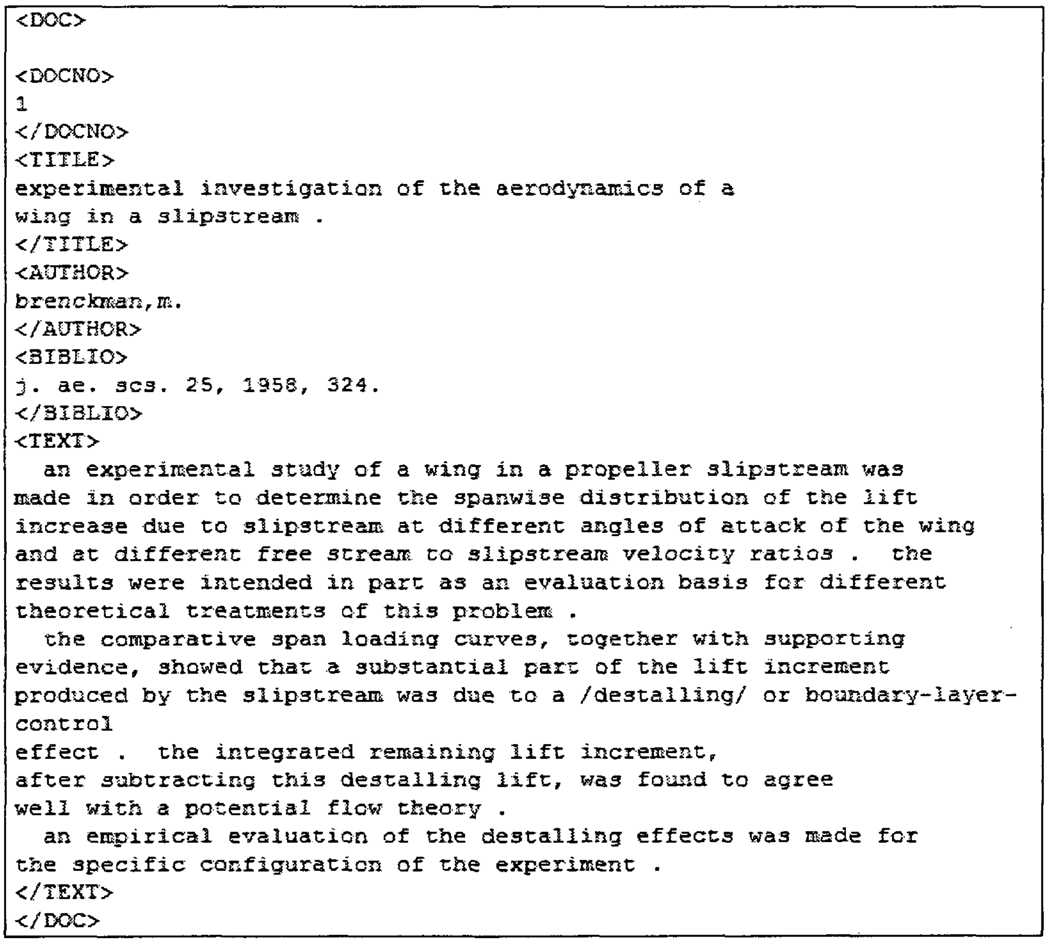
Table 3.1.3 Cosine Similarity Measure

Based on the cosine values calculated in Table 3.1.3, for sample query 1, the top ranked document would be document 2 when sorted in descending order on the measure. Similarly for query 2, the ordering would be Doc2, Doc4, Doc5, and Doc6.

## 3.1 Document Pre-processing and Term Weight Calculation

To implement and test a vector space retrieval system, a subset of 400 documents is taken from a document collection known as cranfield test collection. This collection is available for download on the web [16]. The xml version of the same is used. The collection is also provided with a set of sample queries and relevance judgments file, this contains the set of documents that are considered relevant for a query. These judgments are needed to evaluate the system's performance on retrieval.

A snapshot of an individual document from the collection is shown in the figure 3.1.



```
<DOC>

<DOCNO>
1
</DOCNO>
<TITLE>
experimental investigation of the aerodynamics of a
wing in a slipstream .
</TITLE>
<AUTHOR>
brenckman,m.
</AUTHOR>
<BIBLIO>
j. ae. scs. 25, 1958, 324.
</BIBLIO>
<TEXT>
  an experimental study of a wing in a propeller slipstream was
made in order to determine the spanwise distribution of the lift
increase due to slipstream at different angles of attack of the wing
and at different free stream to slipstream velocity ratios .  the
results were intended in part as an evaluation basis for different
theoretical treatments of this problem .
  the comparative span loading curves, together with supporting
evidence, showed that a substantial part of the lift increment
produced by the slipstream was due to a /destalling/ or boundary-layer-
control
effect .  the integrated remaining lift increment,
after subtracting this destalling lift, was found to agree
well with a potential flow theory .
  an empirical evaluation of the destalling effects was made for
the specific configuration of the experiment .
</TEXT>
</DOC>
```

Figure 3.1 Individual document snapshot

Before we actually build the inverted index for a collection, there are some preprocessing steps that need to be performed to reduce the overhead, increase e the speed and also the size of the index. These steps include parsing the xml document to get just the part in the <Text> tag [Figure3.2], tokenization, stop word removal and stemming.

While parsing, to separate the content of each document or to mark the end of document, a key word 'Reuter end' is used at the end of each document. This idea of marking the document's end is taken from another familiar test collection known as Reuters, it is specifically used for text categorization purposes. The code for this task is available on the web for download [17].

In the next step tokenization, we chop the character sequence in to what are known as individual tokens [18]. At the same time we also remove certain unwanted characters like the punctuation marks [1]. For example, if we consider a character stream from the figure 3.1, "the integrated remaining lift increment, after subtracting" after tokenization process the list of tokens produced is given in the figure 3.1.2.

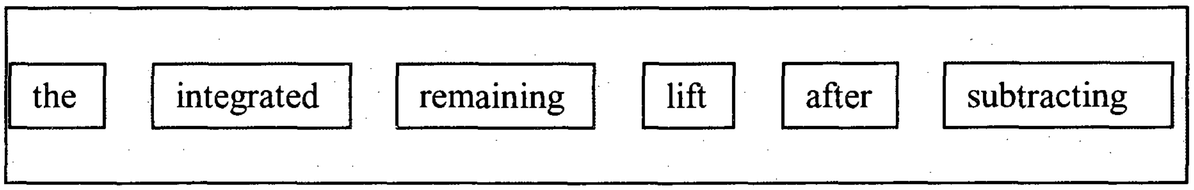


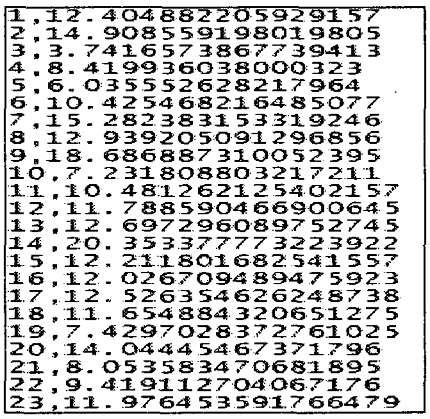
Figure 3.1.2 Output of Tokenization

The next step is removing those words from the list of tokens that are extremely common such as 'and', 'has', 'be' etc. and play no role in selecting the relevant documents to the user query. Since these words are of no use they can be removed which will reduce the index and total number of terms by a significant number. A simple java program would do the task, by storing the list of stop words and then comparing them to the tokens of the collection to remove them.

The last step in the pre-processing is stemming. In this process, the terms are reduced to their root form. For example, "fishing", "fished", "fisher" will be reduced to the root word which is "fish". The most common and empirically effective algorithm for English language is Porter's algorithm [1]. It is available in several programming languages on the web [19]. Stemming ends the pre-processing to be performed on the collection.

After stemming, an inverted index can be built similar to the one in the figure 2.1. We will only consider unique terms in the collection, these terms or tokens are also called as index terms. In this case, we will need some more information in addition to document frequency and term postings for the calculation of term weights in the document and the query. We will also need the within document frequency, $f_{d,t}$ for document term weight calculation. In java, hash map is the data structure that can map key value pairs. After building the inverted index, postings, document and term frequency information can be loaded on to individual hash maps for easy access and fast scanning.

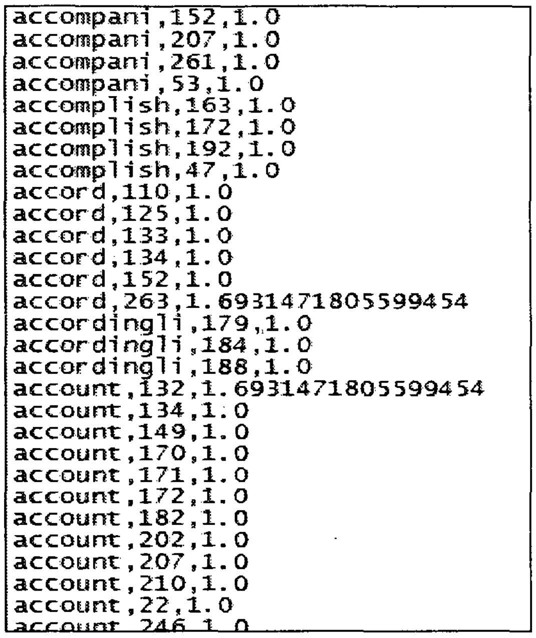
The weights that must be pre calculated before the query processing are $W_d$, $w_{d,t}$ and $w_t$. These values are independent of the query terms or the information need, so can be calculated in advance. The formulae used for the calculation are given in chapter 3 introduction. The snapshot of the calculated values is given in the figure 3.1.3, 3.1.4 and 3.1.5 respectively for $W_d$, $w_{d,t}$ and $w_t$.



```
1,12.404882205929157
2,14.908559198019805
3,3.7416573867739413
4,8.419936038000323
5,6.035552628217964
6,10.425468216485077
7,15.282383153319246
8,12.939205091296856
9,18.686887310052395
10,7.231808803217211
11,10.481262125402157
12,11.788590466900645
13,12.697296089752745
14,20.353377773223922
15,12.211801682541557
16,12.026709489475923
17,12.526354626248738
18,11.654884320651275
19,7.429702837276l025
20,14.044454673717796
21,8.053583470681895
22,9.419112704067176
23,11.976453591766479
```

Figure 3.1.2 $W_d$ values

In figure 3.1.3, the $W_d$ values for each document in the collection is given. The document number and the values are separated by a comma.



```
accompani,152,1.0
accompani,207,1.0
accompani,261,1.0
accompani,53,1.0
accomplish,163,1.0
accomplish,172,1.0
accomplish,192,1.0
accomplish,47,1.0
accord,110,1.0
accord,125,1.0
accord,133,1.0
accord,134,1.0
accord,152,1.0
accord,263,1.6931471805599454
accordingli,179,1.0
accordingli,184,1.0
accordingli,188,1.0
account,132,1.6931471805599454
account,134,1.0
account,149,1.0
account,170,1.0
account,171,1.0
account,172,1.0
account,182,1.0
account,202,1.0
account,207,1.0
account,210,1.0
account,22,1.0
account,246,1.0
```

Figure 3.1.4 $w_{d,t}$ values

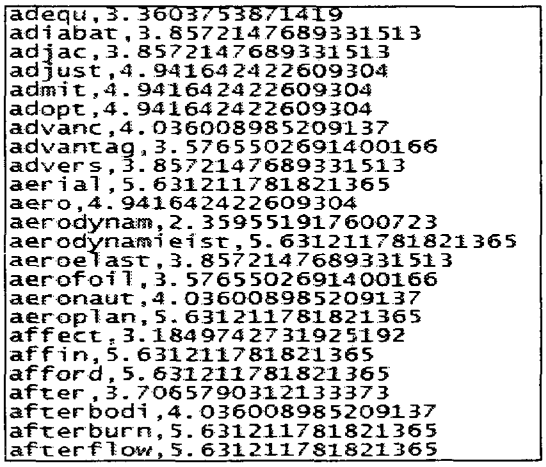
In figure 3.1.4, $w_{d,t}$ values of each term in a document is given. Term, document number and values are separated by a comma. Similarly in figure 3.1.5, we have wt values separated by a comma with the index term.

## 3.2 Algorithm and Pseudo Code

After the pre-calculation is done, the user can now enter a query to the system to find relevant documents. Given below are the steps that are performed before the results are given to the user.

1. Query Input and processing.

   Just like the test collection, the user's query also needs some processing before the weight calculation. Since the user enters the query in natural language we do not need the parsing here described in section 3.1.



```
adequ,3.3603753871419
adiabat,3.8572147689331513
adjac,3.8572147689331513
adjust,4.941642422609304
admit,4.941642422609304
adopt,4.941642422609304
advanc,4.036008985209137
advantag,3.5765502691400166
advers,3.8572147689331513
aerial,5.631211781821365
aero,4.941642422609304
aerodynam,2.359551917600723
aerodynamieist,5.631211781821365
aeroelast,3.8572147689331513
aerofoil,3.5765502691400166
aeronaut,4.036008985209137
aeroplan,5.631211781821365
affect,3.184974273l925192
affin,5.631211781821365
afford,5.631211781821365
after,3.706579031213373
afterbodi,4.036008985209137
afterburn,5.631211781821365
afterflow,5.631211781821365
```

Figure 3.1.5 $w_t$ values

We perform tokenization, removal of stop words and stemming on the query. For example, if the user enters the query as ‘have flow fields been calculated for blunt-nosed bodies and compared with experiment for a wide range of free stream conditions and body shapes’. After processing steps it becomes ‘flow field calcul blunt nose bodi compare experi wide rang free stream condit bodi shape’.

**Pseudo code**

```
If (txt.Querystring is not null)
{
// Get the query in to a string, trim and call string tokenizer
StringTokenizer st = new StringTokenizer(Query);
// call objects of stemmer and removestopwords class
Removewords rmstopqy_obj= new Removewords();
Stemmer stemqy_obj=new Stemmer();

Try
{
// Open a new buffered writer for a file inputqrystring.txt
// while string tokenizer has more tokens
while (st.hasMoreTokens())
{
//Write the token to the file inputqrystring.txt
}
// close the writer
Writer.close();
// call removestopwords class
rmstopqy_obj.main(null);
// call stemmer class passing required parameters
} //end of try
Catch()
{
//Catch the exception of buffered writer
} // end of catch
}
```

2. Query weights calculation.

After processing the query, for the remaining terms or tokens, Wq value should be calculated which remains constant for a query. As mentioned in section 2.3, Wq can be calculated using the following formula.

$$W_q = \sqrt{\sum_{t=1}^{n} w^2_{q,t}}$$

Where $w_{q,t}$ = $r_{q,t}$ . $w_t$ as per the literature for weight calculation. Since, $r_{q,t}$ is 1, we can ignore it and concentrate on getting the wt values for the stemmed query terms. If the query terms exist in the collection, the wt values of which are pre calculated and are loaded on to hash maps, we can get those values by accessing the data structure, otherwise we can consider the $w_t$ value to be zero. Now, the $w_t$ values can be used to calculate $W_q$. Given below the pseudo code for the function to calculate $W_q$ value once the $w_{q,t}$ values are ready.

**Pseudo code**

```
public double CalculateWq(ArrayList<Double> wqtlist_terms){
// declare a double variable to hold the summation value
Double sum_val =0.0;
for(iterate through the passed arraylist wqtlist_terms){
// get the current value from the list
// calculate the square value of w_{q,t} value
double sq_wqtvalue= Math.pow(wqt_val,2);
 sum_val = sum_val +sq_wqtvalue;
}// end of for
// declare a double to hold the final value of Wq
double Wq_value=Math.sqrt(prior_Wq);
// return the Wq value to the function
 return Wq_value;
}// end of the function
```

3. Cosine measure calculation of each document in the collection.

Assuming that all the required hash maps for $W_d$, $w_{d,t}$ are loaded and $W_q$ value for the query is calculated, we can now begin the calculation of cosine similarity score for each document. We declare a hash map for holding the scores, first as we do not know the scores, the keys would be the document id's (1... n), n being the total number of documents in the collection. The values for all the keys are initialized to be zero in the beginning. We do this to avoid looping through all the documents which will save the computation time. After the values in the structure are updated with the summation of query terms that are common to the query and the document, we normalize the values by the product of $W_q$ and $W_d$.

Given below is the pseudo code for the function CalculateCosineMeasure which takes the calculated Wq value and an array list of stemmed query words.

**Pseudo code**

```
public  void CalculateCosineVal(double Wq_val,
ArrayList<String> query_stemmed_words )
{
    // for each query term
    For (int i=0; i<query_stemmed_words.size(); i++)
     {
           // Get the documents list that contain the current
   query term
   terms_in_docs_List= Get_Docs_of_Term(stemmed_word);
   for ( int j=0; j< terms_in_docs_List.size(); j++)
   {
              // Get the wd,t and wq,t values of the current
           stemmed word
              Double wd,t = get the value from hash map
              Double wq,t = get the the value from hashmap
              //Extract the value in the data structure for the
           key as DOC
              ID and update by adding the product of $w_{d,t}$ and
           $w_{q,t}$
              if (hashmap.contains(j))
               {
            Cosineval =  hashmap.get(j)+ wd,t * wq,t;
```

```
                    }
                } // End of inner for loop
            } // end of for
        } // end of the function
        //Normalize the values in the data structure
```

4. Sort and display of results.

   In this final step, we access the cosine measures hash map, sort the values based on cosine score using the java inbuilt function 'collections.sort', we also print the top 20 values to a file which can be displayed to the user as relevant results.

### 3.3 Interface and Results snapshots

The interface and classes were implemented in Java using Net Beans IDE. The screen shot of the screen presented to the user for entering the query and also to analyze the results is shown in the figure 3.3.1.

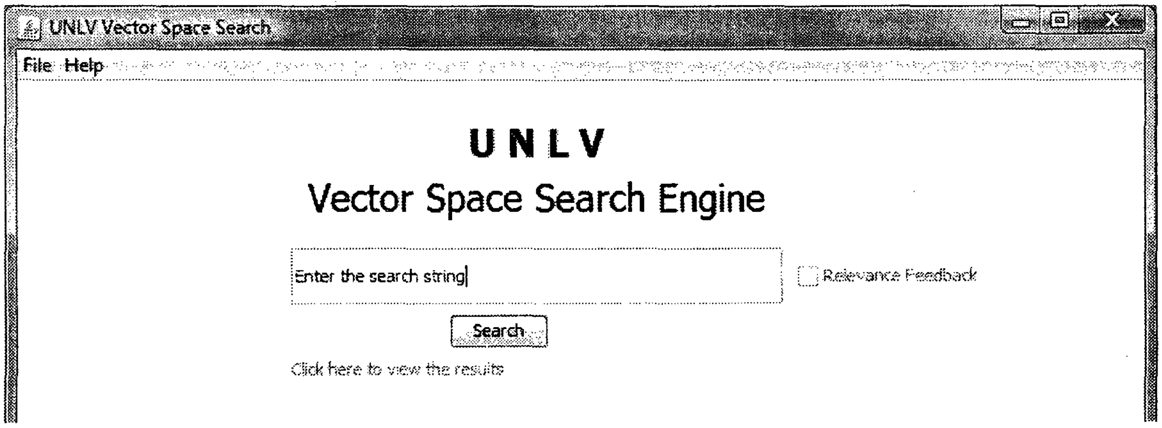


Figure 3.3.1 Initial screen

In the screenshot given in the figure 3.3.2, the user has entered a query and pressed the search button to initiate the retrieval process.

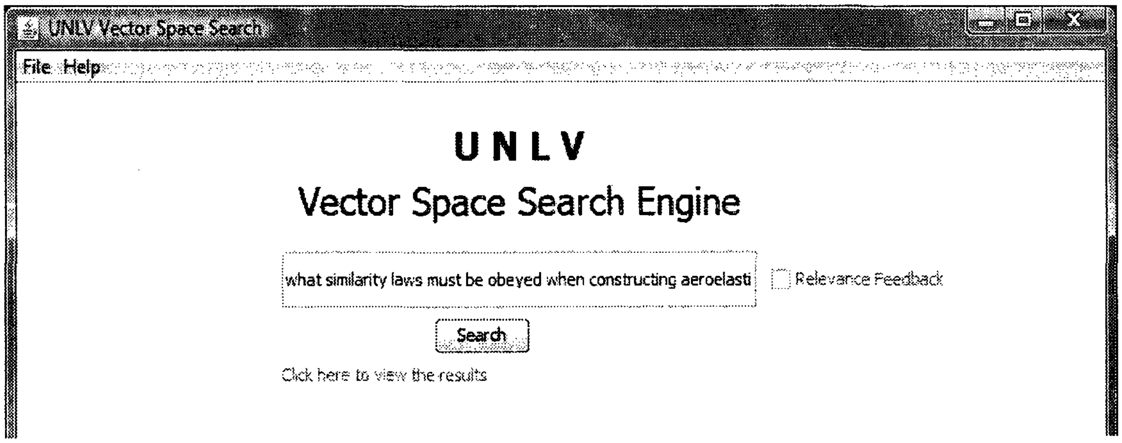

UNLV Vector Space Search
File Help
UNLV
Vector Space Search Engine
what similarity laws must be obeyed when constructing aeroelasti
Relevance Feedback
Search
Click here to view the results

Figure 3.3.2 User enters the query and clicks on Search button

Once the search is complete, the label below the text box provided to enter the query is enabled and the user can click on the label to view the results. The screenshot of the same is provided in the figure 3.3.3.

The results are displayed to the user in a separate window that pops up when the user clicks on the label on the screen 'Click here to view the results'. The screenshot of the results window is shown in the figure 3.3.4.

In figure 3.3.4, the cosine measures and their respective document numbers are displayed as results separated by a comma.

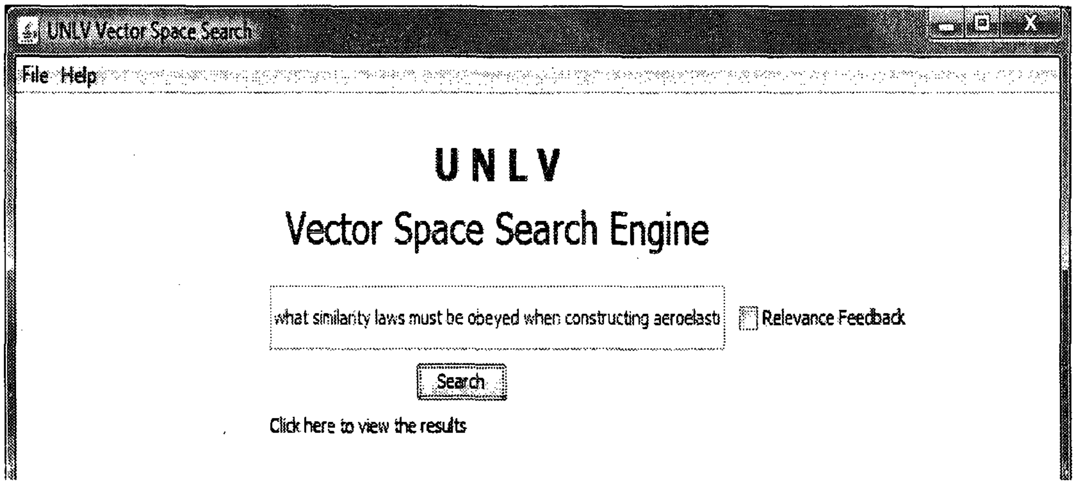
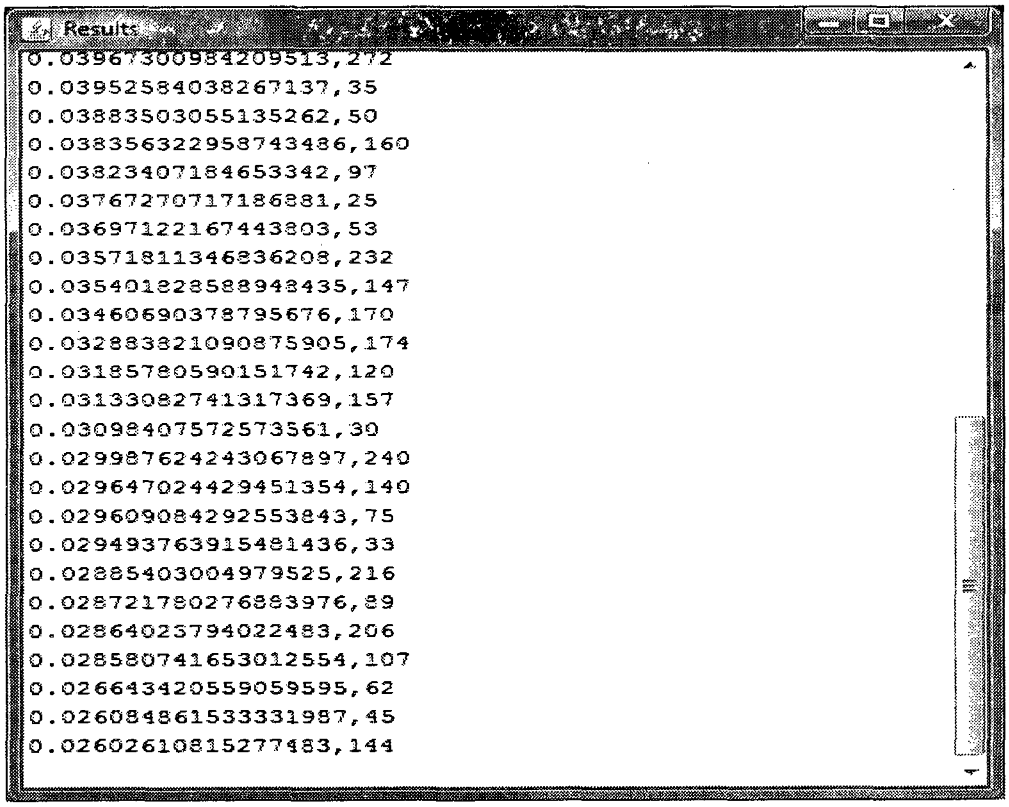

UNLV Vector Space Search
File Help
UNLV
Vector Space Search Engine
what similarity laws must be obeyed when constructing aeroelastı
Relevance Feedback
Search
Click here to view the results

Figure 3.3.3 User can now click on the label to view the results



```
0.03967300984209513,272
0.03952584038267137,35
0.03883503055135262,50
0.038356322958743486,160
0.03823407184653342,97
0.03767270717186881,25
0.03697122167443803,53
0.035718113468362O8,232
0.035401828588948435,147
0.034606903787956 76,170
0.032883821090875905,174
0.03185780590151742,120
0.03133082741317369,157
0.03098407572573561,30
0.029987624243067897,240
0.029647024429451354,140
0.029609084292553843,75
0.029493763915481436,33
0.028854030049795 25,216
0.028721780276883976,89
0.02864023794022483,206
0.028580741653012554,107
0.026643420559059595,62
0.026084861533331987,45
0.02602610815277483,144
```

Figure 3.3.4 Results

# CHAPTER 4

# RELEVANCE FEEDBACK

Users usually feel that effective retrieval query formulation is a tedious process, especially if they have don't have detailed knowledge of the document collection. So, to improve the effectiveness initial user query must be reformulated such that it can provide user with more relevant documents based on the initially retrieved relevant documents. One such technique for automatic and controlled query reformulation was introduced in mid 1960s is relevance feedback. This alteration to the query actually moves it nearer to the direction of relevant documents [20].

## 4.1 Types of Relevance Feedback

Relevance feedback techniques are usually differentiated based on the type of feedback or involvement of the user.

### 4.1.1 Implicit Feedback

This type of feedback requires the least amount of effort from the user to improve the retrieval performance using relevance feedback. Data required is collected without the user interference by monitoring his behavior while performing the search. Some of the commonly used behaviors include reading time, scrolling and interaction. For relevant documents the time spent and reading done will be definitely more than non relevant ones [21].

### 4.1.2 Explicit Feedback

In explicit feedback technique, the user's opinion is taken in to consideration to decide if a document is relevant or not. For example, a checkbox may be provided for each document retrieved initially, to mark the relevancy or even options could be given from which the user can choose one option which gives indication on the relevancy of the document.

### 4.1.3 Pseudo Feedback

This type of feedback is also known as blind relevance feedback since this completely eliminates the user interaction and makes an assumption that the top k documents in the initial retrieval are relevant. This technique is automatic and works most of the time. The only drawback with this comes with the assumption made, when the top k documents retrieved initially are not actually relevant to the query, then the relevance feedback applied may drift the results in to a totally different direction. [1]

## 4.1 Relevance Feedback Models in Vector Space

In section 4.1, we have seen how the feedback techniques are differentiated based on user interference. But, the techniques are also different when applied to different information retrieval models. In this section, we discuss the feedback models that can be applied to a vector space model. Rocchio and Ide are the two most frequently used feedback models in vector model. A version of Ide known as Ide dec-hi and Rocchio are implemented in SMART retrieval system [9].

### 4.2.3 Rocchio Model

Figure 4.2.1 shows how rocchio relevance feedback works. [1] It modifies the initial query in a such a way that the revised query is nearer to the set of relevant documents.

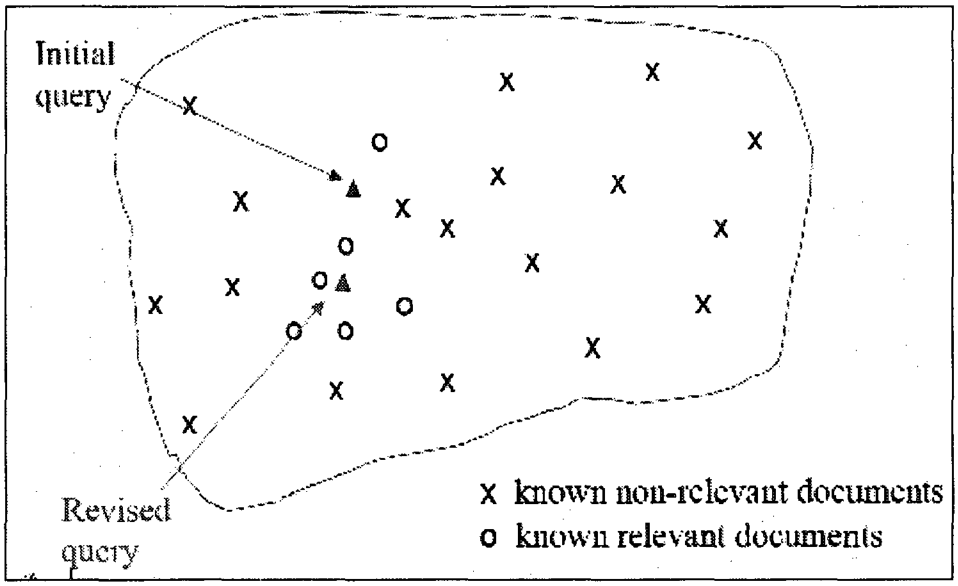


Figure 4.2.1 Rocchio model illustration

If the initial query is marked by $Q_0$ and the modified query is denoted by $Q_1$, then, as per the rocchio algorithm the revised query can be obtained from $Q_0$ using the equation given below [22].

$$\text{Rocchio } Q_1 = Q_0 + \beta \sum_{k=1}^{n1} \frac{Rk}{n1} - \gamma \sum_{k=1}^{n2} \frac{Sk}{n2}$$

Where $R_k$ and $S_k$ are the vectors of relevant and non relevant documents respectively. n1 and n2 are the number of relevant and non relevant documents considered respectively. $\beta$ and $\gamma$ are the parameters that control the contribution of relevant and non relevant documents.

### 4.2.2 Ide Model

In 1971, Ide extended Rocchio's work and proposed two different feedback models. They are very close to the Rocchio's model of feedback, in this model the terms found from the previously retrieved relevant documents are added or subtracted to the original query without the normalization to obtain the new query. Given below are the two versions of Ide, one is known as "Ide Regular" and the other is "Ide dec-hi" [20].

$$\text{Ide Regular } Q_1 = Q_0 + \sum_{k=1}^{n1} Rk - \sum_{k=1}^{n2} Sk$$

$$\text{Ide dec-hi } Q_1 = Q_0 + \sum_{k=1}^{n1} Rk - S_k$$

Where $Q_0$, $Q_1$, $R_k$, $S_k$, n1 and n2 denote the same as specified in the section 4.2.1. In the feedback method 'Ide Regular', we consider all the non relevant documents, but, in the method 'Ide dec-hi', we only consider one non relevant item, usually the one that is retrieved earliest in the search.

## 4.3 Pseudo Rocchio Relevance Feedback in Vector Space Model

The implementation and interface for a vector space retrieval model system is described in the chapter 3. To improve the efficiency and number of relevant documents retrieved for a given user query, one of the relevance feedback techniques mentioned in section 4.2 is incorporated in to the vector space model implemented.

For its simplicity and known efficiency, a pseudo rochhio model of feedback is implemented. The description on pseudo and rocchio types of feedback models is given in section 4.1.3 and section 4.2.3 respectively.

### 4.3.1 Query Expansion

Since it is a pseudo or blind feedback, we assume that the top 10 documents retrieved initially are relevant and use the terms from the same for query expansion. A new query, $Q_1$ is constructed from the initial user query $Q_0$ using rocchio's algorithm where $n_1$ in this case would be 10. Given below is the equation of rocchio's model for new query generation.

$$\text{Rocchio } Q_1 = Q_0 + \beta \sum_{k=1}^{n1} \frac{Rk}{n1} - \gamma \sum_{k=1}^{n2} \frac{Sk}{n2}$$

Where,

n1= number of relevant documents = 10

$\beta = 1$ and $\gamma = 0$(non relevant documents not considered)

$R_k$ = Document vector of relevant document k

Now coming to choosing the terms from the assumed relevant documents, experiments have shown that selecting all the terms from the selected documents is not a good option since it might add not so important terms to the query and also makes the query really huge since each document may contain hundreds of terms. Study has shown that using smaller and good set of terms from the relevant documents often helps in providing the user more number of relevant documents. Also, there are some term selection techniques available for choosing the terms based on the document frequency, term frequency or inverse document frequency information [23].

Number of terms and term selection technique chosen from the relevant documents usually depends on the document collection, since different collections seem to perform differently on the criteria chosen.

### 4.3.2 Implementation

The feedback comes in to picture once the user enters the initial query and clicks on the search button as shown in the figure 3.3.3. The following steps are performed for query expansion and reweighing.

1. Get the terms from the top 10 relevant documents retrieved.

   In this step, we first get the top 10 document numbers or Id's of the initial result. We can then pass this id to a function that actually gets the terms in the corresponding document id passed. Pseudo code of the function getting the terms given the document id's as input is shown below.

**Pseudo code**

```
public ArrayList<String> GetTermsinRelDoc(Integer docnum)
{
// function that takes input as one of the top 10 document id
   and returns an array of document terms
// Get the records (term, document id, $w_{d,t}$) where document id
   is same as the document id passed
Integer DOCNO=Integer.parseInt(Docnum);
if(DOCNO.compareTo(docnum)== 0)
   {   String termandwdt=Wdtval+","+term;
            Listterms_reldoc.add(termandwdt);
   }
// sort based on wd,t value in descending order
// only select the top 5 terms from the array
Listterms_reldoc.subList(5,sizeofarray).clear();
// return the arraylist
return Listterms_reldoc;
} // end of function
```

2. Modify or add the new query term weights after adding the document terms. A hash map stores the query terms and their corresponding weights. But these need to be modified, if the term exists, the weight need to be modified or a new term must be added if otherwise. Given below is the pseudo code for the same task that is updating the hash map for each term selected from the top ten relevant documents.

**Pseudo code**

```
    public void UpdateHashMapqueryterm_wt()
    {
    // for each document in the relevant documents list
    for(int i=0;i<RelDocnum_List.size();i++)
    { // Get the document id and pass it  to function that gets the
       terms from the document
       TermsinRelDoc=GetTermsinRelDoc(DNO);
       // for each term selected from the document
              for(int j=0;j<TermsinRelDoc.size();j++)
       { // Get the term and get the value from hash map       if
    entry exists or add one if otherwise.
              If (queryterm_wqt_mapping.containsKey(termonly))
       { // update the current value by adding the fraction
   Newwqtval = queryterm_wqt_mapping.get(termonly) +
(wdtdoubval/10);
              // update the current value for the term in the hash
map with calculated value
               queryterm_wqt_mapping.put(termonly, newwqtval);
              }
              Else
       { // add new entry to the map
       queryterm_wqt_mapping.put(termonly,      (wdtdoubval/10
    ));
        }
    }// close of inner for loop
    // clearing the temporary array for next loop values
    TermsinRelDoc.clear();
    } //close of outer for loop
    }
```

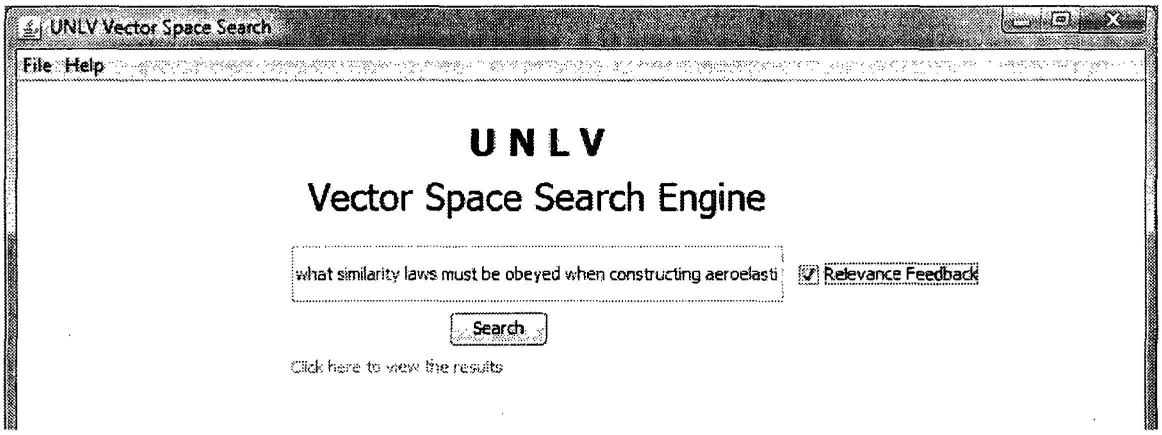
3. Use the modified query to calculate new cosine measures. Since the modified query and the weights are now ready. From here on the process will be similar to what has been discussed in section

3.2. Steps 2, 3, and 4 in the algorithm for vector space model cosine similarity measure will be repeated here.

### 4.3.3 Results

As shown in the figure 3.3.3, the check box 'Relevance Feedback' will be enabled for the user selection once the initial results are ready for the review. The user can run the retrieval model again for the same query, the difference this time would be the application of pseudo relevance feedback to the model for new set of results.

Once the user checks the option for feedback, all the steps given in section 4.3.2 will be executed in order to display the results. Given below is the screen shot of user initiating the feedback.



what similarity laws must be obeyed when constructing aeroelasti

Figure 4.3.1 User initiates feedback

Once the user checks the relevance feedback option, the label 'click here to view results' will be disabled and will be enabled once the new results are ready for review as shown in the figure 4.3.2.

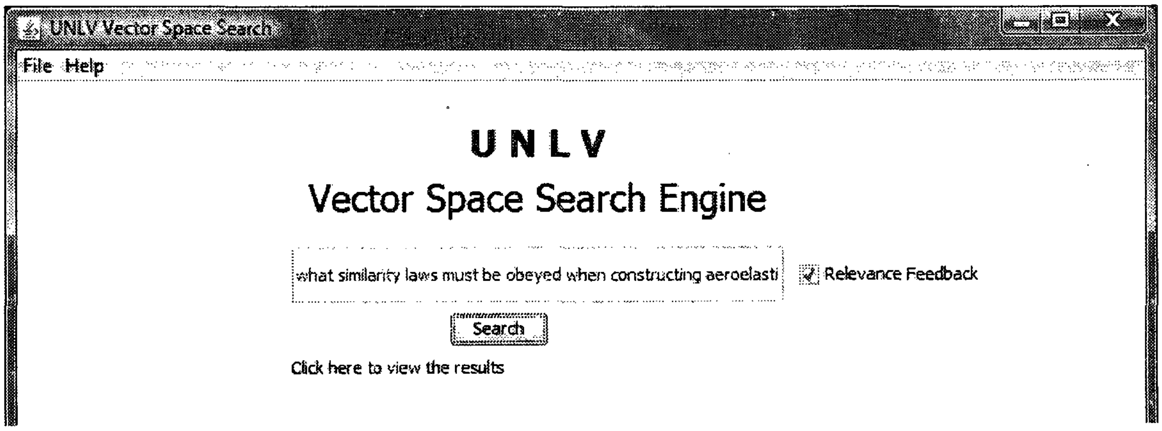
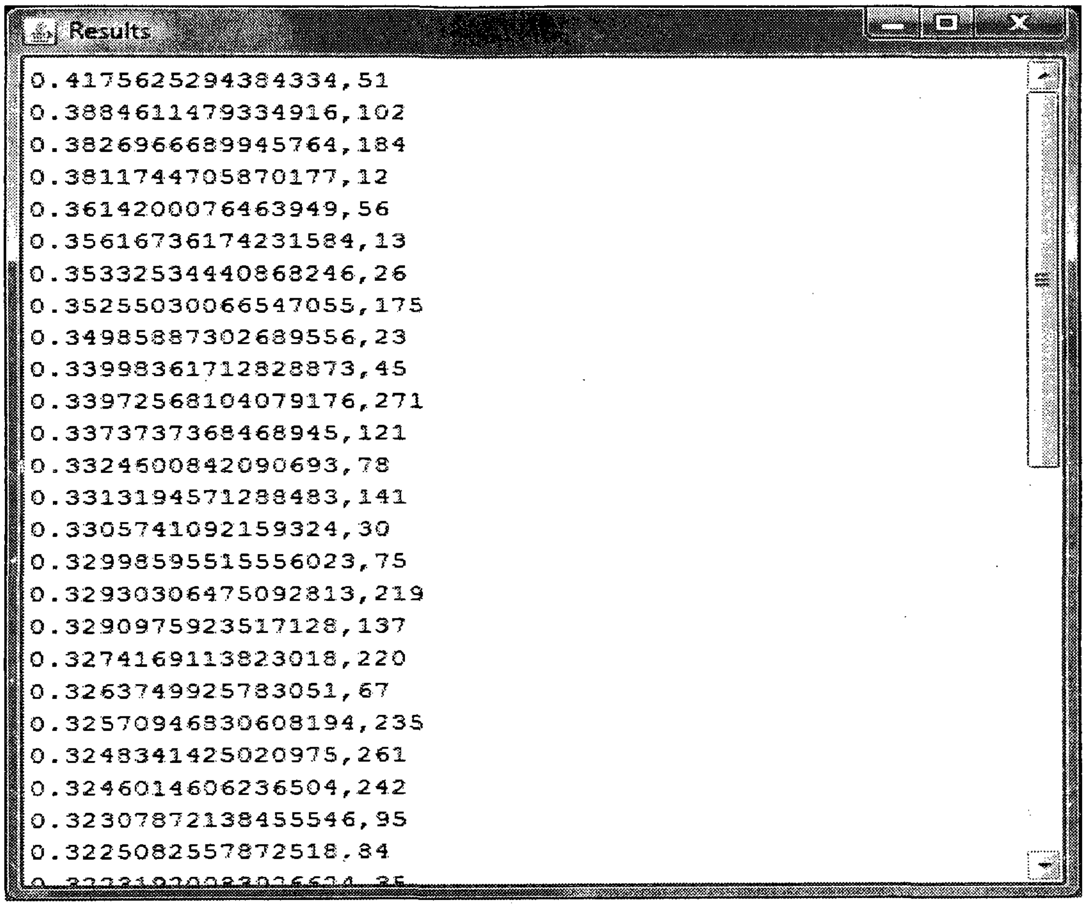

UNLV Vector Space Search
File Help
UNLV
Vector Space Search Engine
what similarity laws must be obeyed when constructing aeroelasti
Relevance Feedback
Search
Click here to view the results

Figure 4.3.2 new results ready for review

The results open in a different window that has information on the cosine measure and the corresponding document id as shown in the figure 4.3.3.



```
0.4175625294384334,51
0.3884611479334916,102
0.3826966689945764,184
0.3811744705870177,12
0.3614200076463949,56
0.3561673617423158 4,13
0.3533253444086824 6,26
0.35255030066547055,175
0.3498588730268955 6,23
0.3399836171282887 3,45
0.33972568104079176,271
0.3373737368468945,121
0.3324500842090693,78
0.3313194571288483,141
0.3305741092159324,30
0.3299859551555602 3,75
0.32930306475092813,219
0.3290975923517128,137
0.3274169113823018,220
0.3263749925783051,67
0.32570946830608194,235
0.3248341425020975,261
0.3246014606236504,242
0.3230787213845554 6,95
0.3225082557872518,84
```

Figure 4.3.3 Results window

# CHAPTER 5

# RESULTS AND EVALUATION

To evaluate any information retrieval system, we need a test collection, sample set of queries which is the information need and a set of relevance judgments for the sample queries which has information on relevant documents for a given query from the collection.

As specified in section 3.1, a subset of cranfield collection is used as a test collection which is provided with a set of 225 queries along with their relevance judgments.

## 5.1 Evaluation of Vector Space Model

There are many ways in which a ranked retrieval system can be evaluated. Choosing one among them highly depends on the requirements of the system on the results. Some of very common evaluation methods of a ranked retrieval are 11-point interpolated precision or more commonly a recall-precision graph, Mean average recision (MAP), precision at k and R-precision [1].

For all the evaluation methods we will need two measures in common which are recall and precision. The precision $P_r$ is defined as a fraction of relevant documents retrieved in top r ranked documents [4].

$$P_r = \frac{Number\ of\ relevant\ retreived}{total\ number\ retreived}$$

Recall $R_r$, on other hand is the fraction of relevant documents retrieved to the total number of relevant documents for a information need.

$$R_r = \frac{Number\ of\ relevant\ retreived}{total\ number\ relevant}$$

Now, for evaluation we consider a set of queries from the sample queries of the test collection and calculate recall and precision at each document retrieved. We then average the precision measure across the measure. The table below gives details on k-precision, k being 20 as we consider the top retrieved results and also R- precision average value for set of queries.

| **Average value of K-Precision** | **Average value of R-precision** | **Average recall after 20 documents retrieved** |
|---|---|---|
| 35% | 54.7% | 66.5% |

Table 5.1.1 Precision and Recall average values

Precision at k is nothing but the exact value of precision at some value of k, where k is the number of top retrieved documents considered. For example, if we consider a query $q_1$, and suppose it have a total of eight relevant documents as per the relevance judgments of the collection. At the 20th document retrieved, the precision value is say 30%, similarly each query will have a different precision value at the same level. The first column in the table 5.1 shows the average of exact precision value of all the queries considered.

The second column in the table gives the average of R-precision values at a particular level of each query considered. R-precision usually gives better estimate than K- precision since it takes the number of total relevant documents for a query in to consideration. Suppose a query has R number of total relevant documents, then we examine top R retrieved results and say r out of them are relevant which means the value of precision at that point would be r/R and so will be recall [1]. For example, if a query has 19 relevant documents, we take the precision value at 19 based on the number of relevant documents retrieved at that point.

The last column shows the average of all the exact recall values after the twentieth document is retrieved.

Table 5.2 shows the calculation of precision and recall after each document is retrieved for a sample query. The process will be repeated for each of the query considered to get the average values.

Query: *"what similarity laws must be obeyed when constructing aero elastic models of heated high speed aircraft?"*

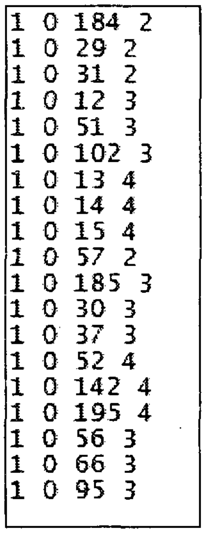
| **Documents** | **Relevant** | **Recall** | **Precision** |
|---|---|---|---|
| 1 | R | 5% | 100% |
| 2 | R | 10% | 100% |
| 3 | R | 15% | 100% |
| 4 | R | 21% | 100% |
| 5 | R | 26% | 100% |
| 6 | R | 31% | 100% |
| 7 | R | 36% | 100% |
| 8 | - | 36% | 87% |
| 9 | R | 42% | 88% |
| 10 | R | 47% | 90% |
| 11 | - | 47% | 81% |
| 12 | - | 47% | 75% |
| 13 | R | 52% | 76% |
| 14 | - | 52% | 71% |
| 15 | - | 52% | 66% |
| 16 | - | 52% | 62% |
| 17 | - | 52% | 58% |
| 18 | - | 52% | 55% |
| 19 | - | 52% | 52% |
| 20 | - | 52% | 50% |

Table 5.1.2 Detailed Precision and Recall values for sample query

## 5.2 Effect on results with Relevance Feedback

Before we examine the statistics of the retrieval method after applying relevance feedback to the initial results, we focus on how the results vary by taking a sample query and comparing the initial results which is the outcome of vector space model with the feedback results. Let us consider the same query used in section 5.1 for table 5.2.

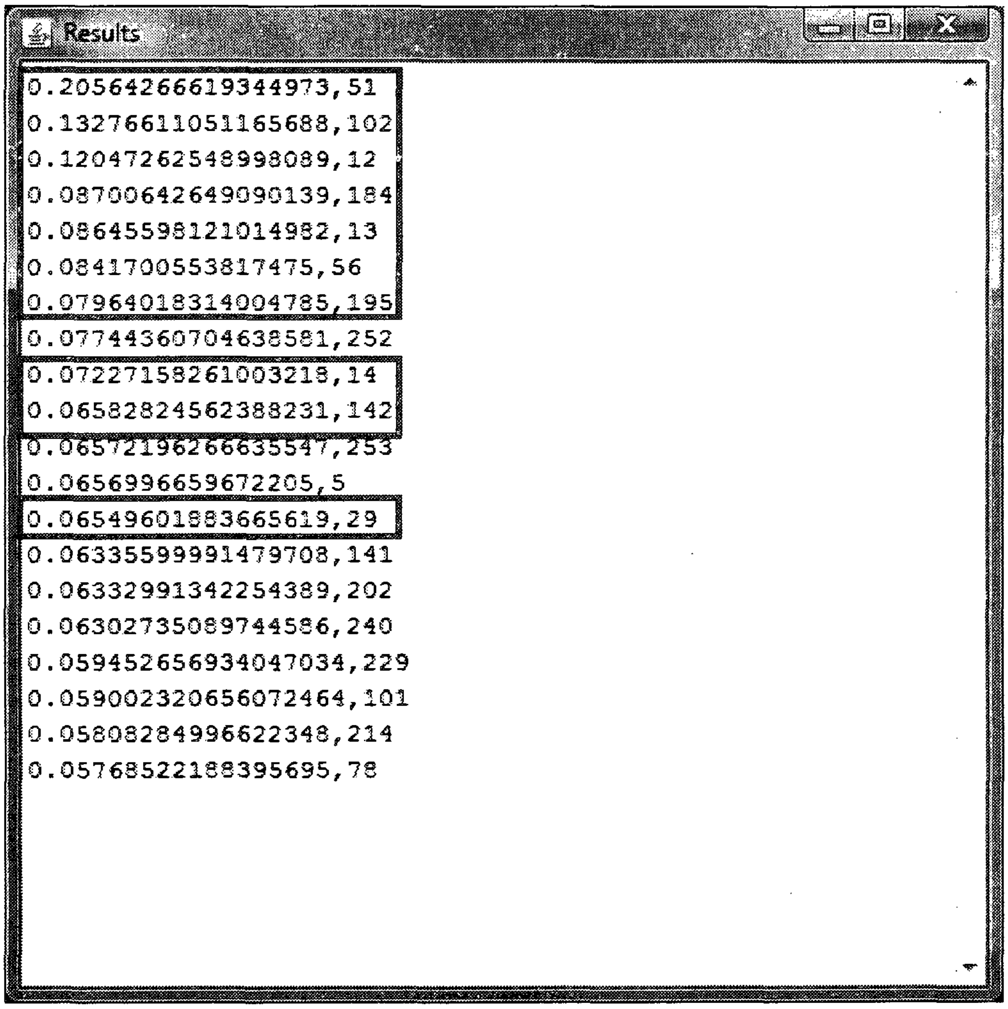
The relevant documents list for the query from the relevance judgments of the test collection is given in figure 5.2.1 and the initial results for the query are given in the figure 5.2.2. The relevant documents that are retrieved initially are marked with a red rectangle.



```
1 0 184 2
1 0 29 2
1 0 31 2
1 0 12 3
1 0 51 3
1 0 102 3
1 0 13 4
1 0 14 4
1 0 15 4
1 0 57 2
1 0 185 3
1 0 30 3
1 0 37 3
1 0 52 4
1 0 142 4
1 0 195 4
1 0 56 3
1 0 66 3
1 0 95 3
```

Figure 5.2.1 Relevant documents list of the query

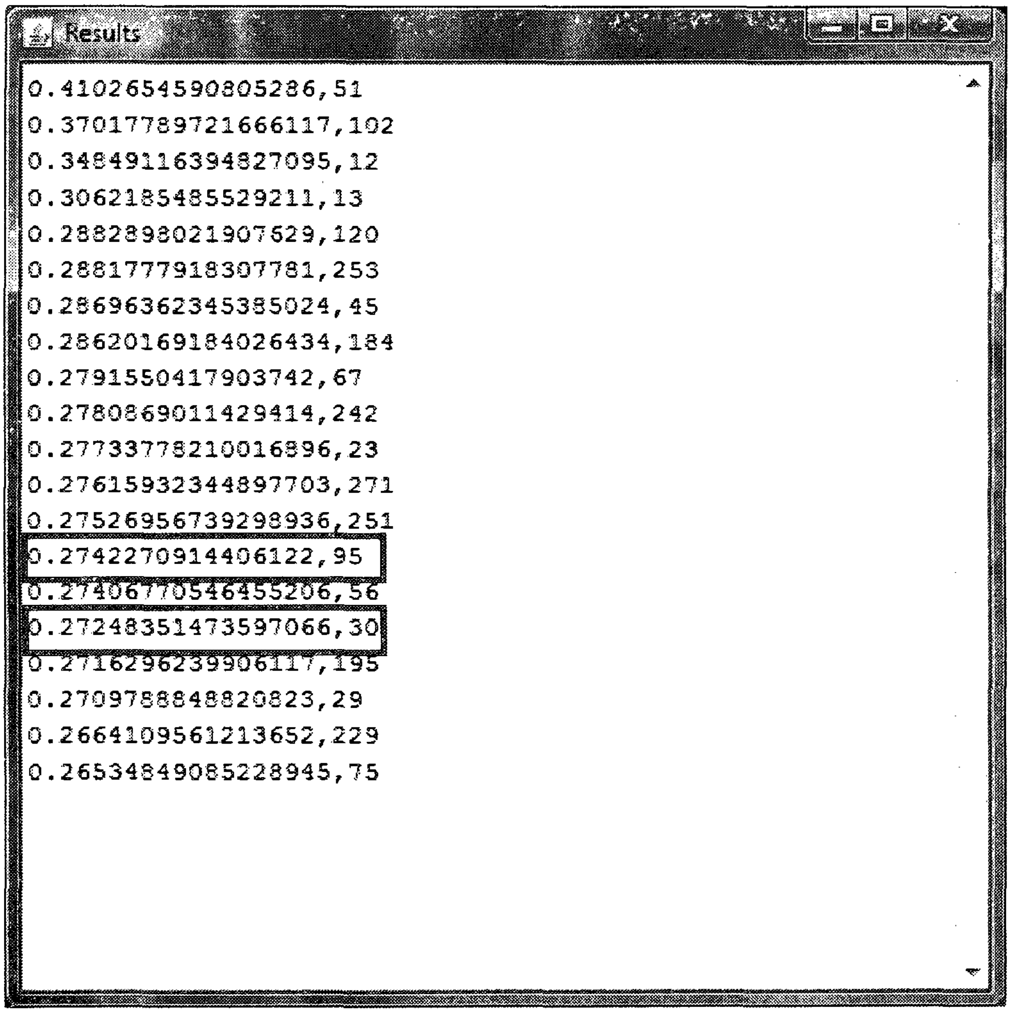
Initially the vector space retrieval model identifies ten of the relevant documents of the query given in the figure 5.2.1. We then apply relevance feedback and the results after the feedback is given in figure 5.2.3. We can see that the results have changed quite a bit even though some of the documents retrieved are same in both the cases. The feedback gives new direction to the results, a simple feedback that assumes the top k number of retrieved documents as relevant uncovers two new relevant documents than the vector space. The new relevant documents are marked in red rectangles.



```
0.20564266619344973,51
0.13276611051165688,102
0.12047262548998089,12
0.08700642649090139,184
0.08645598121014982,13
0.0841700553817475,56
0.07964018314004785,195
0.07744360704638581,252
0.07227158261003218,14
0.06582824562388231,142
0.06572196266635547,253
0.0656996659672205,5
0.06549601883665619,29
0.06335599991479708,141
0.06332991342254389,202
0.06302735089744586,240
0.059452656934047034,229
0.059002320656072464,101
0.05808284996622348,214
0.05768522188395695,78
```

Figure 5.2.2 Initial Results of query showing relevant documents

Feedback results can be varied and examined for performance changes by modifying simple parameters such as number of top documents considered for feedback or even the number of terms from each assumed relevant document.



```
0.41026545908805286,51
0.37017789721666117,102
0.34849116394827095,12
0.3062185485529211,13
0.2882898021907629,120
0.2881777918307781,253
0.28696362345385024,45
0.28620169184026434,184
0.2791550417903742,67
0.2780869011429414,242
0.27733778210016896,23
0.27615932344897703,271
0.27526956739298936,251
0.2742270914406122,95
0.27406770546455206,56
0.27248351473597066,30
0.27162962399906117,195
0.2709788848820823,29
0.2664109561213652,229
0.26534849085228945,75
```

Figure 5.2.3 Results of feedback showing new relevant documents

The most obvious way to measure performance of a system that uses relevance feedback is to calculate recall and precision again for the new results and compare the value with the initial ones. The performance will definitely be high in the second case specifically because the vector space has already uncovered ample amount of relevant documents in the set of top retrieved documents that will be ranked higher in the second set of results.

To overcome this problem, we can use the documents in residual collection that is by removing the documents which are already marked relevant for evaluation of new results after feedback, but, doing this would give the projected performance a lower value almost all the time than the original query. It is very difficult to compare the performance of the system with or without relevance feedback. Usually the best possible way is to do a survey with different users on how many relevant documents they were able to find using feedback [1].

The relative performance of two different versions or variants of a feedback method can be compared in a valid way. Table 5.2.1 shows the performance measures of the two variants of feedback. The variants differ in the number of documents they consider from the initial results for feedback. The first one assumes the first five documents being relevant and the second assumes ten.

The performance measures include precision averaged over a set of queries. Both the variants perform almost same with a minimal difference between average precision across queries.

| Feedback Variant | Number of top ranked documents | Average Precision | Percent Change |
|---|---|---|---|
| 1 | 5 | 63.25% | - |
| 2 | 10 | 61.25% | -3.0% |

Table 5.1.3 varying the number of top ranked documents

Similarly, other parameters can be varied of the feedback method to compare and the performance and choose the one that is most appropriate.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

The main objective of this thesis is to implement and examine a retrieval model and its behavior when relevance feedback is used. Vector space retrieval model was implemented among the different models discussed in chapter 2. Based on the results and evaluation performed on the model, we can conclude that vector space works really well all by itself in extracting most of the relevant documents for given information need. But, with application of one of the simplest forms of feedback strategy it tends to extract even more documents that are relevant.

This thesis concentrates on Vector space model for retrieval. Other models can be implemented and the performance between the models can be compared over a larger collection of data. Also, different feedback strategies discussed in chapter 4 can be applied to different retrieval models to analyze which one outperforms the others. It can further be extended by varying several variants in a feedback method based on term and document selection.

## BIBLIOGRAPHY

1. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, 'Introduction to Information Retrieval', Chapters 1, 6, 8, 9, 11 & 12, Cambridge University Press, 2008. http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html.

2. Amit Singhal, 'Modern Information Retrieval: A Brief Overview', IEEE Data Engineering Bulletin, Volume 24, pages 35-43, 2001.

3. C. J. Van Rijsbergen, 'Information Retrieval', Second Edition, Chapters 1, 6 & 7, Information Retrieval Group, University of Glasgow, London: Butterworths, 1979.

4. Ian H. Witten, Alistair Moffat, and Timothy C. Bell, 'Managing Gigabytes', Second Edition, Chapter 4, Morgan Kaufmann Publishers, Inc, San Francisco, May 1999.

5. Ricardo Baeza Yates, Berthier Riberio Neto, 'Modern Information Retrieval', Chapter 1, Addison Wesley, Addison Wesley Longman, 1999. http://people.ischool.berkeley.edu/~hearst/irbook/1/node2.html.

6. Anselm Spoerri, 'Info Crystal: A Visual Tool for Information Retrieval', Chapter 2, Massachusetts Institute of Technology, 1995.

7. Dik L. Lee, Huei Chuang, Kent Seamons, 'Document Ranking and the Vector- Space Model', IEEE, Volume 14, Issue 2, Page(s): 67 - 75, March/April 1997.

8. G. Salton, A. Wong and C. S. Yang, 'A Vector Space Model for Automatic Indexing', Cornell University, Communications of ACM, Volume 18, Number 11, Page(s): 613 - 620, November 1975.

9. Kazem Taghva, Julie Borsack and Allen Condit, 'Effects of OCR Errors on Ranking and Feedback Using the Vector Space Model', Information Science Research Institute, UNLV, Inf. Proc. and Management, 32(3): 317-327, 1996.

10. Dr. E. Garcia, 'The Classic Vector Space Model: Description, Advantages and Limitations of Vector Space Model', Article 3 of series Term Vector Theory and Keyword Weights.
http://www.miislita.com/term-vector/term-vector-3.html

11. S. E. Robertson, C. J. van Rijsbergen and M. F. Porter, 'Probabilistic models of Indexing and Searching', Proceedings of the 3rd annual ACM conference on Research and development in information retrieval, Cambridge, England, Page(s): 35 - 56, June 1980.

12. K. Sparck Jones, S. Walker and S. E. Robertson, 'A Probabilistic model of Information Retrieval: Development and Comparative Experiments', Part 1, Information Processing and Management: an International Journal, Pergamon Press, Inc, Volume 36, Issue 6, January 2000.

13. LEMUR Toolkit for Language Modeling and Information Retrieval, background and tutorial
http://www.lemurproject.org/

14. Jay M. Ponte and W. Bruce Croft, 'A Language Modeling Approach to Information Retrieval', Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, Melbourne, Australia, Page(s): 275 - 281, 1998.

15. Kazem Taghva, 'Data Mining Concepts', Lecture Notes, Spring 2008, University of Nevada, Las Vegas

16. Information retrieval group by Keith van Rijsbergen, Test Collections.
http://ir.dcs.gla.ac.uk/

17. Kiran Pai, 'A simple way to read an XML file in Java', 2002.
http://www.developerfusion.com/code/2064/a-simple-way-to-read-an-xml-file-in-java/

18. HappyCoders,'TokenizingJavasourcecode'(n.d.)
http://www.java.happycodings.com/Core_Java/code84.html

19. Martin Porter, 'The Porter Stemming Algorithm', Jan 2006.
http://tartarus.org/~martin/PorterStemmer/.

20. Gerald Salton and Chris Buckley, 'Improving Retrieval Performance by Relevance Feedback', Readings in information retrieval, Page(s):

355-364, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 1997.

21. Diane Kelly and Nicholas J. Belkin, 'Exploring Implicit Sources of User Preferences for Relevance Feedback During Interactive Information Retrieval', School of Communication, Information and Library Studies, Rutgers, The State University of New Jersey.

22. Donna Harman, 'Relevance Feedback Revisited', Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval, Copenhagen, Denmark, Page(s) :1 - 10, 1992.

23. Carol Lundquist, David A. Grossman, Ophir Frieder, 'Improving Relevance Feedback in the Vector Space Model', Proceedings of the sixth international conference on Information and knowledge management, Las Vegas, Nevada, United States, Page(s): 16 - 23, 1997.

## VITA

Graduate College
University of Nevada, Las Vegas

Deepthi Katta

Home Address:
4006 Emerald Street, Apt # 105
Torrance, CA - 90503

Degrees:
Bachelor of Technology in Computer Science and Engineering, 2005
Vellore Institute of Technology, India

Thesis Title: A Study of Relevance Feedback in Vector Space Model

Thesis Examination Committee:
Chair Person, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Ajoy K. Datta, Ph.D.
Committee Member, Dr. Laxmi P. Gewali, Ph.D
Graduate College Representative, Dr. Muthukumar Venkatesan, Ph.D.