ENHANCEMENT OF INFORMATION RETRIEVAL RANKING USING FUZZY LOGIC

BY

Lubna Nassar Abd Al Aziz.

SUPERVISED BY

Dr. Khaled Shaalan

A dissertation submitted to the IT Department

In fulfilment of the requirements

for the master degree in IT

The British University in Dubai

Dubai, UAE

June 2011

Acknowledgements

First of all I want to thank god that he granted me the opportunity to meet my dedicated sincere supervisors, Dr. Khaled Shaalan and Dr. Farahad Oroumchian. I feel grateful for all our motivating inspiring discussions, providing me with helpful comments on my thesis and their patience and motivation. I have learned a lot from them. I thank Dr. Khaled for the opportunities he provided me, his insightful comments and directions and his motivating spirit and support that kept me going despite of all faced obstacles. I enjoyed working with Dr. Farahad who never refused a meeting despite of his tight schedule, I am grateful for his best ideas, strong algorithmic knowledge, suggestions to orientate my research towards fuzzy Information Retrieval and for teaching me how to apply the discussed ideas using fuzzy logic. We had fruitful collaboration that allowed me to understand and learn most of the needed background knowledge about fuzzy information retrieval.

TABLE OF CONTENT:

LIST OF FIGURES

LIST OF FUNCTIONS

LIST OF TABLES

LIST OF ALGORITHMS

Abstract:

Having high precision is essential for improving the quality of information retrieval (IR). In this dissertation an IR model is presented to improve the IR ranking function by trying to utilize fuzzy logic in the process of document ranking. Fuzzy logic is chosen since it is a formal, flexible, high performance model for information retrieval that tolerates the uncertainty inherent in natural language [14]. This work extends earlier fuzzy IR models by adding more fuzzy linguistic values, fuzzy variables, using different membership functions, using hedges[1] and rules that consider the document structure. The document is quantified by describing it using features/linguistic variables that contribute most to its relevance to the query like the term frequency ratio, the document frequeny ratio, the title frequency and the document query term overlap. Linguistic values are assigned to each of these variables that associate them with fuzzy membership degrees of high, low, and medium using fuzzy membership functions. Different membership functions were investigated for each of these linguistic variables and the best function was chosen for each variable which contributed the most to the IR precision. An inference engine was built using fuzzy rules that handle these variables to measure the degree of document relevance to the query. Moreover, adding hedges to the query terms was investigated. Hedges served as modifiers of the relevance values by allowing users to emphasize the importance of the query terms to their information needs. It was found that using the standard hedges functions improved the performance slightly by 1.6% over the baseline results, achieved by Lucene[2] at the 10th document cutoff. However their meaning contradicted the IR assumptions which necessitated a modification to the hedges functions to match the purpose behind using them in information retrieval. After modifying the hedges function the performance was raised up to 4.7%; which is more than ***three times better*** than the standard hedges functions. Furthermore, after comparing my Improved Ranking Fuzzy Inference System (IR-FIS) results to a comparable fuzzy logic system, Ruben's R-FIS, it was found that my model's improvement to Lucene's precision is ***five times better*** than Rubens' improvement at the 10th document cutoff.

Hedges[1]: ***Hedges are adjectives or adverbs that precede atomic terms like very, slightly and indeed.*** *In any language there are atomic terms like heavy, old, sick, etc. These terms are referred to as linguistic variables in fuzzy logic. These variables are used to describe uncertain concepts. The membership function for these terms can be modified using linguistic hedges[29].*

Lucene [2]: ***Lucene is a state of the art vector based search engine written in Java*** *[19]. It is a free open source IR software library with full features. It allows indexing and text search with high performance [16].*

Chapter 1

Introduction

In the proposed IR model, fuzzy logic was successfully used as an attempt to improve the performance of the information retrieval. To achieve this, four fuzzy variables or parameters that can quantify the main features of text documents were used; the term frequency ratio (tfr), document frequency ratio (dfr), title frequency and the ratio of the number of query terms that appear in one document to the length of the query (overlap). Three fuzzy values were associated with each variable, which are high, medium and low, to represent their linguistic values. A variety of membership functions were tested for these values to explore their effect on performance; like the bell shaped, S-shaped, L-shaped, trapezoidal and triangular functions. Afterwards, all these variables were considered in the construction of the rule base of the fuzzy inference system. Using hedges, which are unary operators on fuzzy sets [30], was investigated and it did improve the precision especially after changing the function to fit the IR field. IR-FIS Search engine results were compared to other two IR models, one model was the Lucene which is based on the vector space model and another model was a fuzzy logic model called R-FIS by Rubens. IR-FIS outperformed both at the 10th document cutoff (P10). In this dissertation, particularly in chapter 2, an overview is given for the main models used in information retrieval like the vector space, binary, probabilistic models, and the fuzzy logic model. Detailed explanation of the proposed fuzzy system is given in chapter 3. Chapter 4 describes experiments setup and result discussion. Finally, chapter 5 sums up the achievements and has also the needed future enhancements.

1.1 Research Questions

The dissertation is addressing the following questions.

- How different ***membership*** *functions* would affect the precision in a fuzzy information retrieval?
- How the ***structure*** *of a document* can be utilized in a fuzzy IR and what would be its effect?
- How using more *fuzzy* ***values*** such as high, medium, and low would affect precision?
- Whether incorporating *fuzzy* ***hedges*** in queries will improve precision or not?

1.2 Scope

The main aim is to study using fuzzy logic in the field of information retrieval for retrieving English documents. The dissertation concentrated on improving the precision of the IR system which is having high relevant fraction of documents in the returned list. This was approached by utilizing the structure of the document (title,body), ratio of match between the query and the document (overlap), using more fuzzy values, enhancing the fuzzy membership functions, adding more inference rules and investigating the effectiveness of using hedges.

The speed of the retrieval process was not considered as a main search objective in this study since it is immaterial for the purpose of this thesis. Using languages other than English, enhancing the efficiency of the IR process, or considering other retrieval models other than the fuzzy logic were all considered as out of scope topics.

1.3 Contribution summary

I was able to build a fuzzy IR model (IR-FIS) that has outperformed Lucene [16] at document cut off P10 by ***4.7%*** which is ***5 times better*** improvement than the one reached by Rubens R-FIS [19]. I experimented with my IR system to find answers for my research questions.

- First, after studying the different membership functions I discovered that the S-shape function and the bell function were more effective than the triangle and L-shape functions.
- Second, considering the document structure as (title, body) in the retrieval process didn’t help the performance which made me recommend investigating the importance of considering more important contextual factors like having the term in the first line of the paragraph or beginning of the document.
- Third, using more fuzzy values, like medium and low, plus considering them in the fuzzy rules led to better results than those achieved by the systems that ignored them like Rubens R-FIS in [19].
- It was found that using hedges did improve precision only when it was used in a way that matched the information retrieval field. After experimenting with using hedges in the same way others are using it in other fields, I discovered that their way was counterintuitive when used in the IR field. Therefore, I deduced that the hedges function should be adjusted to match the goal behind using it in IR. Changing the function resulted in improving the performance by nearly three times compared to the performance achieved when the old commonly used hedges function 3.6 was used.

  More elaboration on my contribution will be given in sections 3.1 and 4.5.

Chapter 2

Background and Literature review

This chapter starts by defining the basic meaning of information retrieval and how it differs from other data retrieval systems. It also elaborates the two stages of information retrieval; indexing and scoring. This is followed by a description of the most commonly used retrieval models. Four scoring models are discussed, the vector space, probabilistic, binary and fuzzy models. The results of using each of these models are usually evaluated using a standard IR evaluation system to be able to compare their performance. Section 2.1.3 gives a brief description of the commonly used IR system evaluation techniques. Finally, the chapter ends by discussing applications of fuzzy logic in the IR field as well as other fields and how this affected the IR-FIS construction.

2.1 Overview of Information retrieval

The formal definition for information retrieval is as follows:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) "[4].

Nowadays information retrieval is replacing old sorts of database search or data retrieval systems where the value of a key field is used to search a table for the required record. In relational databases the data searched is usually well structured while with information retrieval the documents can be unstructured or semi-structured, where the documents have some sort of basic structure like the title and body. In information retrieval, each document is perceived as a set of words and the collection of documents over which we perform information retrieval is called a corpus. The main task of any information retrieval system is to provide a list of documents which are relevant to the user *information need* that is conveyed to the computer through the query provided by the user. The user usually needs relevant documents even if the exact terms s/he used in the provided query were not present in these documents. Therefore, a major difference between information retrieval (IR) systems and other kinds of information systems is the uncertainty nature of IR. Table 2.2 elaborates on the difference between the IR and data retrieval systems [15].

The data retrieval systems were first used in libraries and then the need was evident for journals and doctors record keeping where information used to have a *well-defined structure*; records with predefined fields. But then there was also a high need for information retrieval in governments and other organizations where access is available only to *unstructured information* which is the focus of this study.

This type of information retrieval is also different from the web IR systems that aim at finding information over the web where billions of web pages are available and *fast search* engines are required [4]. Table 2.2 compares the web IR and the classical IR systems [15].

Table 2.1: Classical IR V.s. web IR [15]

	Information retrieval	Data Retrieval
Data	Free text, unstructured	Database tables, structured
Queries	Keywords, Natural language	SQL, Relational algebras
Results	Approximate matches	Exact matches
Results	Ordered by relevance	Unordered
Accessibility	Non-expert humans	Knowledgeable users or automatic processes

Table 2.2: Information Retrieval V.s. Data Retrieval [15]

	Classical IR	Web IR
Volume	Large	Huge
Data quality	Clean, no dups	Noisy, dups
Data change rate	Infrequent	In flux
Data accessibility	Accessible	Partially accessible
Format diversity	Homogeneous	Widely diverse
Documents	Text	HTML
# of matches	Small	Large
IR techniques	Content-based	Link-based

2.1.1 Indexing:

To simplify and speed up the process of information retrieval, preprocessing of the documents available should first take place. One of the main processes that should be completed before starting the retrieval process is indexing. The input to this process is all the documents in the corpus and the output is an index that has all the main terms available across all these documents, a term here is any non-trivial word reduced to its word stem, along with its associated list of documents where the term occurred at least once. This index should be built in advance in order to gain the benefits of indexing at retrieval time. The indexing process includes four main sub-processes which are document tokenization, stop words removal, tokens normalization, and stemming; see algorithms 2.1 and 2.2.

After collecting the documents to be indexed, i.e. the corpus, the preprocessing algorithm 2.1, which is considered as a sub-module for the indexing process, is applied not only to each document in the document corpus but to each query as well. The sub-module preprocessing accepts the list of raw documents or queries and preprocesses it to prepare it for indexing. The indexing module calls the preprocessing one at the beginning and then creates the index using the main terms in all the corpus documents. These terms form the content of what is sometimes called the IR dictionary.

Here are detailed steps of the preprocessing and indexing modules. After collecting the documents and the queries the first step is to tokenize each document or query and this means turning each document or query into a list of tokens. As a result, any sentence in the text will be split at spaces and any punctuation characters will be removed. The second step is dropping stop words or frequent terms which occur with high frequency across the corpus to the extent that it loses its significance. For example, common words like “the”, “and”, or “what” are irrelevant to the document or query content. These words should be found and deleted before starting the retrieval process to reduce the amount of processing and speed up the retrieval. The following list has twenty five most commonly used stop words; a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, or with [3].

The third step is the normalization of tokens so term matching can be achieved regardless of differences in the token character sequence [4]. The index should only contain terms from the IR dictionary not tokens and the difference is that a term is a normalized type which is a class of all tokens containing the same character sequence. This can be achieved for example by changing everything into lower case so that “Hard” is the same as “hard”. Also by removing characters like hyphens which allows two tokens like “pre-process” and “preprocess” to be mapped into the same term “preprocess”.

The fourth and last step before indexing is stemming which should take place to bring the words to their root. The performance of an IR system will be improved if term groups like (direct directive directed directing direction directions), are mapped into a single term like _direct which is good since it will lead to finding the word regardless of its format. This may be done by removing the various suffixes (-ED, -ING, -ION, IONS) to leave the single term. So the process refers to getting rid of the ends of the words and the removal of derivational affixes like chopping the “ness” from the end of the word so that two words like “clever” and “cleverness” are mapped to the same term _clever[4]. This process will reduce the total number of terms in the IR system, and hence reduce the size and complexity of the data in the system, which is always advantageous.

Term	Document list
brutus	d1,d3,d6,d7
caesar	d1, d2, d4, d8, d9
julius	d10
killed	d8
noble	d5
with	d1, d2, d3, d5

Figure 2.1: Example of an index [4]

All mentioned steps should be applied in the same way to the queries before starting the retrieval process. Now that the documents are preprocessed we can consider the group of terms available in the documents as the IR dictionary. Finally, the index is created according to algorithms 2.1 and 2.2. Figure 2.1 shows the output as a sample index which has the term and the list of documents where it occurred.

```
Preprocessing(x)
{ for each document (d) in x
  {
    d = Tokenize(d)
    d=Stop-word-removal(d)
    d= Normalization(d)
    d=stemming(d)
    add d to the preprocessed-x
  }
  Return (preprocessed-x) }
```

Algorithm 2.1: Preprocessing sub-module

```
Indexing (corpus)
{ preprocessed-corpus = Preprocessing(corpus)
  index=an empty file
  for each document(d) in the preprocessed-corpus
    for each term (t) in d
    {
      if (t in index) then append d to the document list of t
      else add a new index entry to index that has t and d
    }
  return(index)}}
```

Algorithm 2.2 : Indexing module

An index can also have the list of terms in the IR dictionary and associated with each term, not only the list of document ids where the term is found, but also the frequency of its occurrence in each of these documents as shown in Figure 2.2. So instead of keeping (termID, docIDs) pairs in the index, the 3-tuples (termID, docIDs, termfrequency) are kept. Here the frequency can be utilized in the ranking process later on.

Doc 1	**Doc 2**
I did enact Julius Caesar: I was killed i' the capitol; Brutus killed me.	So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

Term	**docID**		**term**	**docID**		**term**	**Doc. Freq.**	**→**	**Postings lists**	
I	1		ambitious	2		ambitious	1	→	2	
did	1		be	2		be	1	→	2	
enact	1		brutus	1		brutus	2	→	1→	2
julius	1		brutus	2		capitol	1	→	1	
caesar	1		capitol	1		caesar	2	→	1→	2
I	1		caesar	1		did	1	→	1	
was	1		caesar	2		enact	1	→	1	
killed	1		caesar	2		hath	1	→	2	
i'	1		did	1		I	1	→	1	
the	1		enact	1		i'	1	→	1	
capitol	1		hath	1		it	1	→	2	
brutus	1		I	1		julius	1	→	1	
killed	1		I	1		killed	1	→	1	
me	1		i'	1		let	1	→	2	
so	2		It	2		me	1	→	1	
let	2	➔	julius	1	➔	noble	1	→	2	
it	2		killed	1		so	1	→	2	
be	2		killed	1		the	2	→	1→	2
with	2		let	2		told	1	→	2	
caesar	2		me	1		you	1	→	2	
the	2		noble	2		was	2	→	1→	2
noble	2		so	2		with	3	→	2	
brutus	2		the	1						
hath	2		the	2						
told	2		told	2						
you	2		you	2						
caesar	2		was	1						
was	2		was	2						
ambitious	2		with	2						

Figure 2.2: Instead of keeping (termID, docIDs) pairs in the index the 3-tuple (termID, docfrequency, docIDs) is kept [4]

2.1.2 Scoring models:

To be able to give a score for each document according to its relevance to the query, there should be a method by which documents can get a weight representing their degree of relevance which is the scoring model. There is more than one model for scoring and they differ in their ranking process. One model may base the document weight on the query term frequency of occurrence in the document regardless of where it appears. Other models can give different weights according to the position of the term in the document or the importance of the term in the query. For example, the vector space scoring model [11], uses the free text queries where the order of the query terms has no significance. The query and the document are considered as an unordered set of terms with no difference in term importance resulting from the order of term appearance in the query or in the document. Ranked Boolean retrieval [27], on the other hand, is another way of scoring that relies on zone scoring which means different weights are given to the terms according to the place or the zone they appear in within the document. So the weight given to the terms occurring in the title is different from that given to the terms occurring in the body [4].

These models of information retrieval also differ in their document as well as query representation in addition to their methods of matching them and ranking the results [14]. Several models of information retrieval were used before fuzzy logic, like the vector space model, the binary model and the probabilistic IR model as will be discussed in the coming sections.

2.1.2.1 Vector space:

Vector space retrieval can be used to find documents with high term frequency. The idea behind the vector space is that each document is represented by the weights of its terms (t_1 to t_n) as a vector of weights. Therefore an n-dimensional vector space has n unique terms and each unique term in the document or the query corresponds to a dimension in the space.

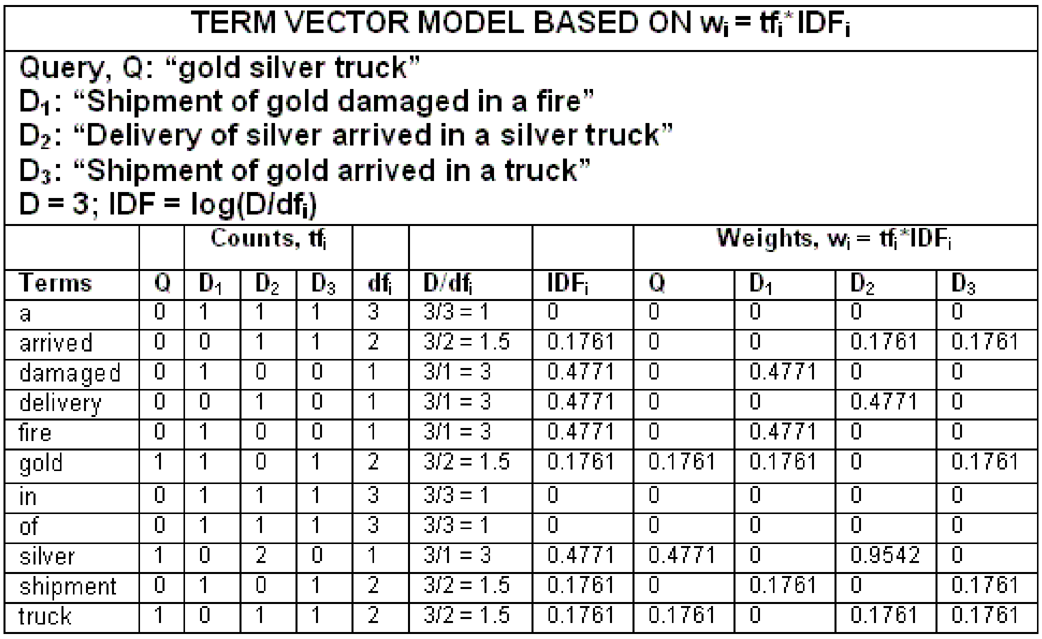
$$V(d) = (w(t_1, d), w(t_2, d), \ldots, w(t_n, d)) \qquad \text{as in [11] [2.1]}$$

These weights are usually calculated by the tf – idf measures as shown in Table 2.3. The use of term-weighting based on the distribution of a term within a collection, like using the tf measure, always improves the performance or at least does not have negative effect on performance. The vector space also gives higher weights to infrequent terms across the documents even if they appear frequently in the document itself; like using the IDF measure [2].

Each query is represented as a vector too which is V(q) and the relevance of the document to the query is represented by a score that is calculated by finding the inner product of the query vector and the document vector, (score(q, d) = v (q).v (d)). This similarity value can be normalized afterward by dividing the score by the document length, scores[d] <- scores[d] divided by length[d][4].

Ranking according to this score should take place to limit the results returned since large document sets are often retrieved so only the top K scores should be selected [11]. Ranking models can be divided into two types: those that rank the query against each document and those that rank the query against the group of related documents [5]. The first type of ranking model is the one used in the vector space model as well as the probabilistic model. The following table shows an example of how weight calculation is done based on the tf and IDF measures. It assumes having three documents and one query and ends up with three document vectors for D1,D2 and D3

Table 2.3: This table shows how weights are calculated in the vector space model [5]



TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$

Query, Q: "gold silver truck"
D_1: "Shipment of gold damaged in a fire"
D_2: "Delivery of silver arrived in a silver truck"
D_3: "Shipment of gold arrived in a truck"
D = 3; IDF = $\log(D/df_i)$

		Counts, tf_i						Weights, $w_i = tf_i * IDF_i$			
Terms	Q	D_1	D_2	D_3	df_i	D/df_i	IDF_i	Q	D_1	D_2	D_3
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Like any IR model, the vector space model has advantages as well as draw backs. The first benefit of using vector space is having ranked retrieval. The second is the fact that weighing the terms is done according to their importance. The third is being able to achieve partial matching [31]. On the other hand, the model has its disadvantages as well. One of the vector space weaknesses is that the weights are calculated intuitively since they are not based on a formal theory [31]. Another main disadvantage of using vector space is that the model assumes that there are no relations between the terms and this assumption is not realistic and can negatively affect precision. This is due to the fact that it can lead to the retrieval of irrelevant documents because of "Polysemy"; which refers to having the same term referring to two different meanings depending on the context. Recall can also decline by failing to retrieve documents that are relevant and this is due to "Synonymity"; where two different terms can refer to the same thing [6].

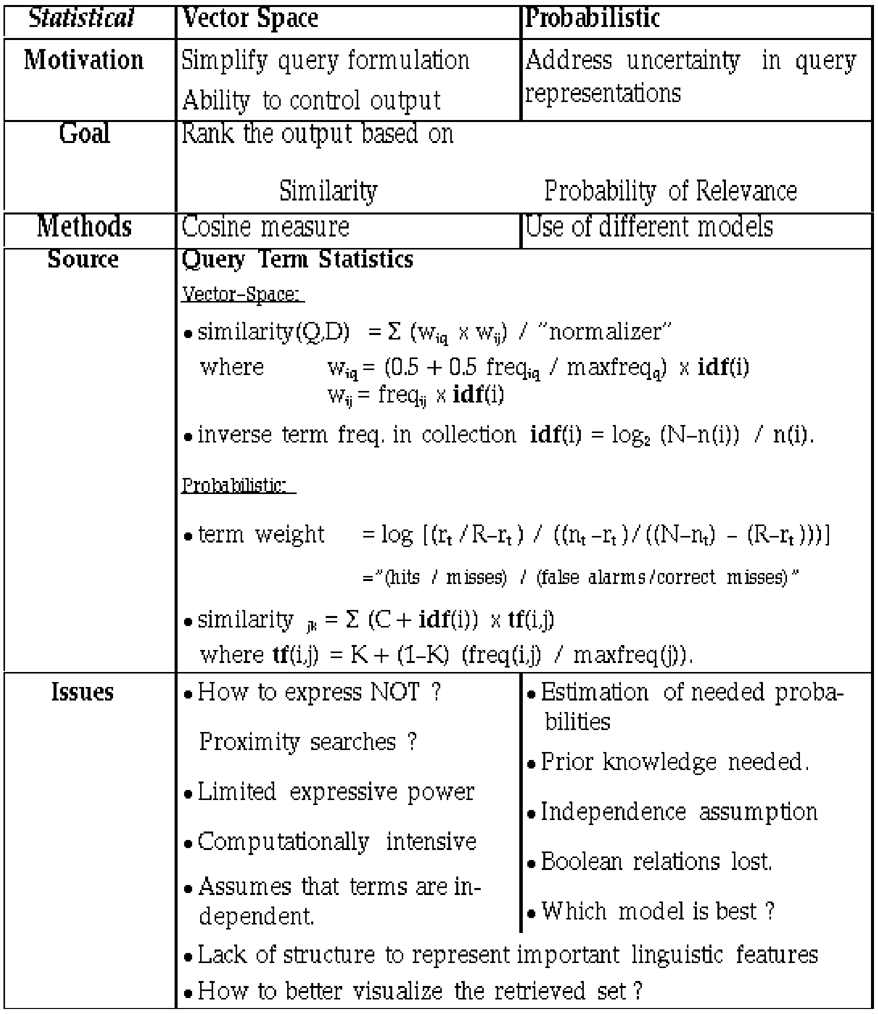
2.1.2.2 Probabilistic model:

Another main IR ranking model is the probabilistic model which is another model that provides reasoning under uncertainty. It is an information retrieval system that takes into account the uncertainty in the presentation of information needs and documents. This model ranks the documents based on their probability of relevance to the query, given all the evidences available. There can be many sources of evidence used by the probabilistic retrieval model; the most common one is the statistical distribution of the terms in both the relevant and non-relevant documents. Information about the statistical distribution of terms was used by researchers as a trial to improve the IR performance, which means the frequencies with which terms occur in documents, document corpus, or subsets of document collections such as documents considered relevant to a query [6]. The information about the term distribution is used to assign probability of relevance to each document in the set so that ranking of the retrieved documents is based on its probability of relevance [2].

One of the first and most influential information retrieval methods that uses probability is the Binary Independence Model [4], where document d is relevant iff P(R=1|d,q) > P(R=0|d,q). A document (d) and a query (q) are represented by the vector $\vec{x} = (x_1, \ldots, x_M)$ where x_t=1 if the term (t) is present in the document d or the query q and x_t=0 if t is not present in d or q [1]. The model doesn't recognize any association between terms. Documents are then ordered in descending order by the estimated probability of relevance or the *P(Info need | document),* which is the probability that an information need is satisfied by a given document.

The model usually needs prior knowledge before retrieval which is sometimes considered as a main disadvantage. For example, weight computation considers term frequency in the relevant and irrelevant documents, which necessitates that the relevant documents are known or that these frequencies can be reliably estimated. So if we knew the percentage of relevant documents in the collection, then we could use this number to estimate $P(R=1|\vec{q})$ and $P(R=0|\vec{q})$ [20]. Moreover, it requires the use of at least a few relevant documents then the user can assign relevance judgments to documents in relation to his query then the task of the IR system is to reach an approximation of the set of relevant documents which are ranked using their probabilities. When users provide the retrieval system with relevance feedback, this information is used to re-compute the probabilities so that the probabilities of the documents that have the query terms which appeared in the user list of relevant documents are increased, whereas the probabilities are decreased for those that have the query terms but did not appear in the user relevant list. That is why some people see using probabilistic indexing as related to *relevance feed-back* more than to *term-weighting* schemes as the case with vector space model [2].

The statistical approaches as shown in Figure 2.3, whether the vector space or the probabilistic models, have their benefits as well as drawbacks. As for the benefits, first, they provide rank of relevance that enables the user to specify relevance threshold to reduce the amount of retrieved documents and get the most relevant ones. Second, queries can be formulated using natural language which is easier for the user than using Boolean expressions. Third, the uncertainty nature of query concepts can be represented by these models. However, there are disadvantages as well like having a limited expressive power so (A and B) or C can't be represented by a vector space query. They also lack the structure to express phrases. In addition, their relevance finding has a high computation cost. Moreover, they can't suggest how to modify a query if needed; they only provide the user with limited ranked linear list of results. Additionally, users should use words used in the relevant documents and the queries need to contain a large number of words in order to improve the retrieval performance [2].



Statistical	**Vector Space**	**Probabilistic**
Motivation	Simplify query formulation Ability to control output	Address uncertainty in query representations
Goal	Rank the output based on Similarity	Probability of Relevance
Methods	Cosine measure	Use of different models
Source	**Query Term Statistics** Vector-Space: • similarity(Q,D) $= \Sigma (w_{iq} \times w_{ij}) /$ "normalizer" where $w_{iq} = (0.5 + 0.5\ freq_{iq} / maxfreq_q) \times$ **idf**(i) $w_{ij} = freq_{ij} \times$ **idf**(i) • inverse term freq. in collection **idf**(i) $= \log_2 (N-n(i)) / n(i)$. Probabilistic: • term weight $= \log [(r_t / R-r_t) / ((n_t - r_t) / ((N-n_t) - (R-r_t)))]$ ="(hits / misses) / (false alarms/correct misses)" • similarity $_{jk} = \Sigma (C +$ **idf**(i)$) \times$ **tf**(i,j) where **tf**(i,j) $= K + (1-K)\ (freq(i,j) / maxfreq(j))$.	
Issues	• How to express NOT ? Proximity searches ? • Limited expressive power • Computationally intensive • Assumes that terms are independent.	• Estimation of needed probabilities • Prior knowledge needed. • Independence assumption • Boolean relations lost. • Which model is best ?
	• Lack of structure to represent important linguistic features • How to better visualize the retrieved set ?	

Figure 2.3: Characteristics summary of the statistical retrieval approaches; vector space and probabilistic model [2]

2.1.2.3 Boolean information retrieval (BIR) model:

The Boolean model of information retrieval (BIR) is a classical IR model and, at the same time, the first and most adopted one. It was used by virtually all commercial IR systems [26]. To avoid linear processing of documents, for each query a document index is created in advance which might look as follows:

Table 2.4: Term document matrix

	Term 1	Term 2	Term 3
Doc 1	1	1	1
Doc 2	0	1	1
Doc 3	1	0	1

Where one means that the term exists in the document and zero means it does not. Boolean systems are used heavily in IR systems for searching large document collections. In a Boolean IR system, documents are represented by sets of keywords, usually stored in an inverted file; an inverted file is a list of keywords associated with the identifiers of the documents in which they occur. Boolean queries can be seen as keywords connected by the Boolean logical operators (AND, OR, NOT).

Queries are given in a disjunctive normal form so information retrieval can be done in two steps. If we consider the query as a set of terms t_j's then the first step is to get the set of documents (S_j) where the term t_j exists and do this for each query term then we get the union of the intersection of all these document sets (S_j's) and this is considered the set of relevant documents.

$Q = (W_i \text{ OR } W_k \text{ OR } ...) \text{ AND } ... \text{ AND } (W_j \text{ OR } W_s \text{ OR } ...)$

1. $S_j = \{D_i | W_j \text{ element of } D_i\}$ …depending on whether $W=t_j$ or $W_j=\text{NON } t_j$

2. UNION (INTERSECTION S_j) …this is the set of relevant documents [26]

The advantages of such model can be summarized in its clean formalism, simple implementation, and in being intuitive. Despite of this, it can be seen as a process of *data retrieval* more than *information retrieval* and this is due to its disadvantages that can be evident in its reliance on *exact matches* which can lead to having too few or too many results. Moreover, it deals with all the terms as having equal weights which is unrealistic. Beside the fact that it is difficult to rank retrieved documents although there are documents which are more important than others. Finally, it is not always an easy straight forward process to translate the information need into a well formed Boolean expression [26].

2.1.2.4 Fuzzy model

The Fuzzy logic model combines the advantages of the vector space model as well as the Boolean model. Its flexibility and high performance resemble the vector space where weights are utilized in the ranking function plus having the formalism and simplicity of the Boolean model [19]. Furthermore, fuzzy logic can be seen as a computation with words instead of numbers, due to using linguistic variables, which makes it tolerate the imprecision and uncertainty that is inherent in the representation of textual information which is based on natural language [14]. All this makes fuzzy logic a good candidate model for information retrieval where natural language is used in forming the queries and documents.

The fuzzy model accepts document statistics and produces one relevance value representing the degree of document relevance to the considered query. It has three main tasks as shown in Figure 2.4. The fuzzification, the fuzzy inference system and it ends by the defuzzification. In this section each task will be explained by using an example.

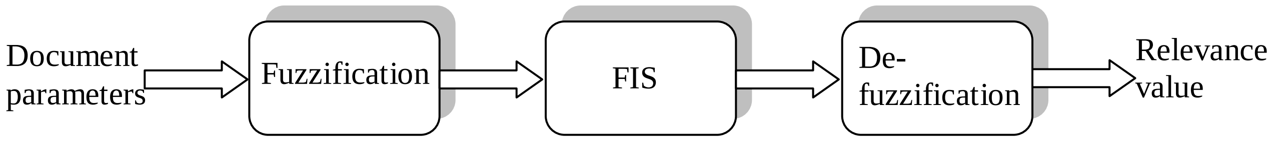


Figure 2.4: Fuzzy system information flow

In 1965, Lotfi Zadeh introduced the theory of fuzzy logic in a paper. He introduced the idea of fuzzy sets where the membership of domain values in any of these sets is represented by a membership function that maps the values of the domain into a number that ranges from zero, which means the value is not a member in the set, and one, which means the value is a full member of the set [14]. The range of membership values that are greater than zero and less than one mean that the domain value has a partial membership in the fuzzy set with a degree proportional to the membership value that is a fraction of one. On the other hand, in the binary or discrete sets the membership function can be represented as follows $\forall\, x|x \in U: f(x) \rightarrow \{0,1\}$; so the range will either be zero or one. In fuzzy logic the range of values will not only be 0 or 1 but will be any value in the interval that starts by zero and ends at one: $\forall\, x|x \in U: f(x) \rightarrow [0,1]$.

The difference between using the discrete and the fuzzy model will be elaborated by giving an example. When using a query term to retrieve relevant documents, if we are using the discrete method then we want a function that will say either the document is relevant or not relevant but will not be able to find documents that are *partially* relevant. If we base the relevance on one variable like the term frequency ratio (tfr) which is the term frequency divided by the document length (tf divided by doc-length) then a discrete method of deciding the relevance can be represented as follows: if (tfr > 0.3) → document relevance = 1

if (tfr ≤ 0.3)→ document relevance = 0

$$\text{relevance(d)} = \begin{cases} 0 & \text{tfr} <= 0.3 \\ 1 & \text{tfr} > 0.3 \end{cases}$$

Here the membership of the document in the set of relevant documents is decided based on the value of the query term ratio in that document. So the term ratio values are mapped only to two discrete values 0 or 1, it is either a non-member or a full member in the set of relevant documents.

This means that any document can only be either relevant or not but there are no degrees of relevance which excludes a whole range of documents that may be partially relevant. Conversely, the fuzzy way of representing the relevance is different in that it allows for degrees of relevance which we call the relevance fuzzy values. So we can have high, medium and low relevance fuzzy values. Other fuzzy values like very high and very low relevance can always be added to achieve higher precision. Moreover, these values are fuzzy which allows any document to have degrees of membership not only be a member or non-member.

We will keep our assumption that the relevance will only be decided based on one variable which is the term frequency ratio. In addition, we will only consider one fuzzy value, which is the high relevance value. The membership would be calculated using a function like the following, which can be seen as an example of using the L-shape membership function.

$$\text{high-relevance (d)} = \begin{cases} 0 & \text{tfr} <= 0.2 \\ \dfrac{(\text{tfr} - 0.2)}{0.5} & 0.2 < \text{tfr} < 0.7 \\ 1 & \text{tfr} >= 0.7 \end{cases} \qquad [2.2]$$

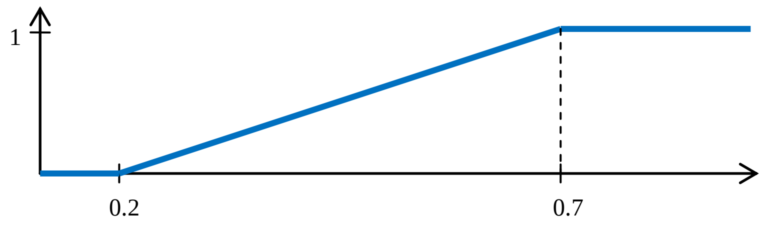


Figure 2.5: membership function in the high relevance fuzzy variable.

Other functions can be investigated as well like the triangular, trapezoidal, and the S-shape function or a combination of some of them [24]. Furthermore, we can add more variables other that the term frequency (tfr) like the document frequency ratio (df) or the overlap. More fuzzy values other than high relevance can also be added like medium and low relevance.

This stage of deciding the fuzzy variables, fuzzy values, the fuzzy membership functions and getting the membership values is called ***fuzzification***. Now if we decide to use more than one fuzzy variable then each variable should have its own fuzzy values. For example, if we consider the df variable beside using the tfr variable then we should have fuzzy values for the tfr as well as the df. The df represents the document frequency ratio, or the number of documents where the term appeared divided by the number of documents in the corpus.

The tfr fuzzy values can be the high tfr, medium tfr, and low tfr. Each document has three values associated with each of its terms. These values represent the degree of document membership in each of these sets depending on the value of the term frequency in the document. The same thing can be said with regard to the document frequency ratio variable (df) where the fuzzy values can be low df, medium df, and high df.

Each term will have three fuzzy values associated with it that represent the degree of the term membership to each of these values depending on the fraction of corpus documents that has the term t appearing in it. These fuzzy values should then be related to the document relevance using fuzzy rules which can be intuitive rules or based on experience. This stage of rule evaluation is the ***FIS*** stage. Assume that we have three rules as a sample of a fuzzy rule base and that we have two relevance fuzzy values; high relevance and low relevance then the rule base will be as follows:

if ((high-tfr) And (low-df)) then high-relevance …rule 1
if ((medium-tfr) And (low-df)) then high-relevance …rule 2
if ((low-tfr) And (high-df)) then low-relevance …rule 3

We can then replace the parameter (high-tfr) by the document high-tfr fuzzy value which is based on the term frequency in the document. We should do the same thing with the rest of the parameters.

It is worth mentioning here that there are two types of operators used in fuzzy logic, numeric and linguistic operators. When using the Boolean logic operators AND, OR, and NOT in fuzzy logic they are redefined according to [9] as the minimum, maximum, and complement respectively and they are called the *Zadeh operators*. In [9], for the fuzzy variables x and y:

NOT x = (1 - truth(x))
x AND y = minimum(truth(x), truth(y))
x OR y = maximum(truth(x), truth(y))

The linguistic operators on the other hand are called hedges. These are generally adverbs such as "very", or "Indeed", which are sometimes called modifiers since they change the meaning of a set by using a mathematical formula as will be elaborated in section 3.1[9].

Since these rules can be seen as disjunctions of conjunctions and according to the definition of Zadeh operators, we take the minimum of any two Anded values then we take the maximum of the relevance values that belong to the same fuzzy value. We should end up with three relevance values high, medium and low relevance. This is best described by the following example. Assume that (high-tfr = 0.7, medium-tfr =0.4, low-tfr =0.1, high-df = 0.3, low-df=0.6), it follows that:

high relevance = *minimum* (0.7, 0.6) = 0.6 ….this is after applying rule 1
high relevance = *minimum* (0.4, 0.6) = 0.4 ….this is after applying rule 2
low relevance = *minimum* (0.1, 0.3) = **0.1** ….this is after applying rule 3

These rules are ORed since it is a disjunction, then the first two rules can be combined together as follows: ***high relevance*** = *maximum* (0.6, 0.4) =**0.6**.

Now we have two fuzzy values for the considered document representing its degree of high and low relevance; in other words the document is **0.1** low relevant and **0.6** high relevant to the query. The last task that should start now is ***defuzzification***, which is moving from fuzzy values into having one value that represents the relevance of the document to the query and can be used to rank the document among all retrieved documents. The defuzzification can be done by applying a centroid function as in [19] or the weighted average method. The weighted average is calculated by combining the deduced values for the high and low relevance after multiplying them by their weights and dividing by the sum of these weights which results in a single value. We will assume the weights; 1 for the high and 0.1 for the low relevance. After using the weighted average , the resulting document relevance is 0.554 which is calculated as follows:

$$\text{Relevance-Value} = \frac{1\text{x}\mathbf{0.6}+0.1\text{x}\mathbf{0.1}}{1+0.1} = 0.554 \qquad [2.3]$$

In this example only two document fuzzy variables were considered; the term frequency ratio (tfr) and the document frequency ratio (df) and we ended up with 0.554 as the relevance value.

2.1.3 Early fuzzy logic retrieval models:

There was more than one fuzzy logic model used before the recent models. The earliest models were the Mixed Min and Max model (MMM) and the Paice model [10]. In the MMM model there is a fuzzy set for each index term. Each document has a weight d_{Ti} proportional to its membership in the fuzzy set T_{i} that is associated with the index term t_i. For the following queries

Query 1- t_i **or** t_j
Query 2- t_i ***and*** t_j

In the first one the relevant document should be in the union of the two fuzzy sets $T_i \cup T_j$ while for the second query it should be in the intersection $T_i \cap T_j$. The degree of relevance of document d to both queries is calculated as follows:

Query 1- $d_{Ti \cup Tj} = min(\mathrm{d}_{Ti\,,}\ \mathrm{d}_{Tj})$
Query 2- $d_{Ti \cap Tj} = max(\mathrm{d}_{Ti\,,},\mathrm{d}_{Tj})$

The resulting values are softened by the MMM model as follows:
Given a document *D* with index-term weights $d_{T1}, d_{T2}, ..., d_{Tn}$ for terms $t_1,t_2, ..., t_n$, and the queries:
$Q_{or} = (t_1 \text{ or } t_2 \text{ or } ... \text{ or } t_n)$
$Q_{and} = (t_1 \text{ and } t_2 \text{ and } ... \text{ and } t_n)$

According to [10], MMM uses softness coefficients are C_{or1}, C_{or2} where $C_{or1} = 1 - C_{or2}$,$C_{and1} = 1 - C_{and2}$ while calculating the similarity between the query and the document as follows:

$S (Q_{or}, D) = C_{or1} * max(d_{T1}, d_{T2}, ..., d_{Tn}) + C_{or2} * min(d_{T1}, d_{T2}, ..., d_{Tn})$ *where* $C_{or1} > C_{or2}$
$S (Q_{and}, D) = C_{and1} * min(d_{T1}, d_{T2}, ..., d_{Tn}) + C_{and2} * max(d_{T1}, d_{T2} ..., d_{Tn})$ *where* $C_{and1} > C_{and2}$

On the other hand, the Paice model was an improvement to the MMM model. It incorporates all of the term weights when calculating the similarity:

$$\text{SIM}(Q, D) = \Sigma\, r^{i-1} * \mathrm{d}_{\mathrm{Ti}} / \Sigma\, r^{i-1}$$

Where r is a constant coefficient and d_{Ti} is arranged in ascending order for And queries and descending order for Or queries[10].

More recent models dealt with the term as a concept rather than a single term. The term is given a weight representing the importance of the term concept to the document. This improved Paice on the average precision and recall at P5[10].

2.1.4 Information retrieval system evaluation:

Usually a test collection is used to evaluate information retrieval systems in a standardized way. This collection consists of a document set, a query set and a record of relevance that relates each query to each available document. It was decided to consider fifty topics in the test collection .We should emphasize here the difference between a query and an information need. The query can find any document that contains any of the query terms or even all of them but there is no guarantee that the found documents satisfy the user information need behind his query [4].

One source for standard test collections commonly used is the Text Retrieval Conference (TREC). In TREC large test collections of documents along with their relevance records to a large set of topics or information needs are made available for competitor systems. For each topic, there is a clear statement written based on a human judgment describing items that should be considered relevant to each topic [4]. Early TRECs consisted of fifty topics with their relevance record evaluated against a variant subset of documents that can have 100,000 different documents in each subset. There are now many test collections that are built with the same format as TREC collections. One such collection is the CLEF 2009 INFILE collection which is used in the IR-FIS experiments as will be elaborated on in section 4.1[22].

Recall and precision measures are frequently used to evaluate the effectiveness of the information retrieval systems. Recall represents the retrieved fraction of relevant documents so it requires that you know the whole set of relevant documents in advance. On the other hand, precision represents the relevant fraction of retrieved documents which does not require any knowledge of the total set of relevant documents. An optimal retrieval system is a system that would return relevant documents only and will not miss any of them. So according to the formula 2.4, it provides precision and recall values of 1, while in real world systems, precision usually decreases with greater recall [28].

$$\text{Precision} = \text{P(relevant|retrieved)} = \frac{\text{\#relevant items retrieved}}{\text{\#retrieved items}}$$

$$\text{Recall} = \text{P(retrieved|relevant)} = \frac{\text{\#relevant items retrieved}}{\text{\#relevant items}} \qquad [4]$$

For ranked sets, the precession at PN, where N is the number of documents retrieved whether relevant or non-relevant, can be used to calculate the precision so far. For example, precision at P10 is calculated by summing the precisions found till the 10th retrieved document and dividing by 50 or the number of queries[4].

The recall would require knowing the total number of all relevant documents so it cannot be found gradually like precision. In Web search the N is usually chosen to represent the number of pages people usually look at. Hence, P10 is important since ten can be considered as the usual number of pages or documents usually considered by the user.

Sample TREC 7 topic:

<num>Number: 396
<title> sick building syndrome
Indentify documents that discuss sick building syndrome or building related illnesses.
<narr> Narrative:
A relevant document would contain any data that refers to the sick building or building-related illnesses, including illnesses cause by asbestos, air conditioning, pollution controls. Work-related illnesses relevant.

Figure 2.6: Sample of a topic in TREC 7 [6].

Precision and Recall according to [3]

Precision = % of first ***n*** ranked documents that are relevant
Recall = % of ***R*** relevant documents among the first ***n*** ranked [2.4]

2.2 Related applications of fuzzy logic

Fuzzy logic was used in more than one way in the field of information retrieval, to serve different aspects of the retrieval process, plus other fields. In the field of information retrieval fuzzy logic was used to measure the similarity between documents as in [23], or to match queries with documents as in [14] and [19]. It was also used in other fields that have nothing to do with the information retrieval as in [24]. In our IR-FIS a concept was borrowed from each of these applications. An overview of how fuzzy logic was used in each case plus how the IR-FIS was affected by each application is given below. IR-FIS can mainly be seen as an enhancement to the system used in [19] by utilizing all learned concepts from all other applications.

In [14], fuzzy logic was used to match queries with relevant documents. The query was perceived as a logical formula and the documents as interpretations of that formula. The terms in the query are given weights according to the importance of the term to the query. The document, on the other hand, is seen as a fuzzy set of keywords and the membership of a keyword represents its importance in representing the meaning of the document. In IR-FIS the importance of the query terms was represented by using hedges and the document features were represented as fuzzy sets as will be seen later.

In [23], fuzzy IR system was used to retrieve documents based on their similarity to the query document. The proposed system was tested using Arabic documents. The aim was to retrieve similar documents to a query document and they used fuzzy logic to measure the similarity between the documents.

The fuzzy values used were “near duplicate”, “very similar”, “dissimilar” and “very dissimilar”. This was one reason behind using more than one fuzzy value in IR-FIS. The model in [23] is different in that all documents that are relevant to the query document are retrieved based on the document content similarity even if the terms used are different. This is considered as a future enhancement to IR-FIS.

One of the applications of fuzzy logic in other fields is explained in [24]. They did not use fuzzy logic for information retrieval but used it to interpret the shoot length of a mustard plant which is decided by certain variables such as humidity, rain fall, temperature, pollution, water, soil, distance and crop management. In IR-FIS this was imitated by looking for the main document features that affect its relevance to query terms and representing these features as fuzzy variables. In [24], because of the uncertainty of plant growth, fuzzification was needed. They experimented with five different membership functions, the bell shaped, S-shaped, L-shaped, trapezoidal and triangular membership functions. They stressed on the importance of selecting a valid membership function. They stated that using a wrong function leads to wrong data being fed to the fuzzy system which leads to wrong fuzzy output that results in having high error in the defuzzified values. This was one of the reasons behind considering all these functions inorder to reach the best one while trying to enhance the IR-FIS performance.

In [19], the fuzzy logic model was actually used in information retrieval in a very similar way to IR-FIS. Rubens used fuzzy logic for information retrieval and he came up with a ranking model that he called R-FIS. We can consider IR-FIS as an enhancement of Ruben’s R-FIS model. The R-FIS ranking model had rules for fuzzification based on three fuzzy variables; tf, idf and overlap. A third variable that considers the document structure was added in IR-FIS which is the title variable that reflects the term frequency in the document title. Rubens idf was calculated by $\log(\frac{N}{n})$, where N is the number of documents in the corpus and n is the number of documents that had the considered term. In IR-FIS the tf was changed to be a ratio relative to the document length. The df was also used instead of the idf. The df is calculated by $\left(\frac{N}{n}\right)$ for simplicity, and its meaning was handled in the IR-FIS rules. Moreover, Rubens used for each of the three used fuzzy variables only one fuzzy value and its negation, high and not high. He also manipulated these variables using only three rules. In IR-FIS three fuzzy values were used, high, medium and low, and eleven rules in the FIS. Rubens used only the triangular and trapezoidal membership functions. The operators use by Rubens were product (*) for and and sum (+) for OR while in IR-FIS the min is used for And and the max for OR. For the defuzzification Rubens used the centroid function while in IR-FIS the weighted average was used. Rubens compared his findings to Lucene’s performance; his system outperformed Lucene at P10 by +0.0092. Therefore Lucene was used as a baseline system to compare IR-FIS performance to Rubens’ R-FIS. This was done by comparing each model’s improvement to Lucene’s precision at P10.

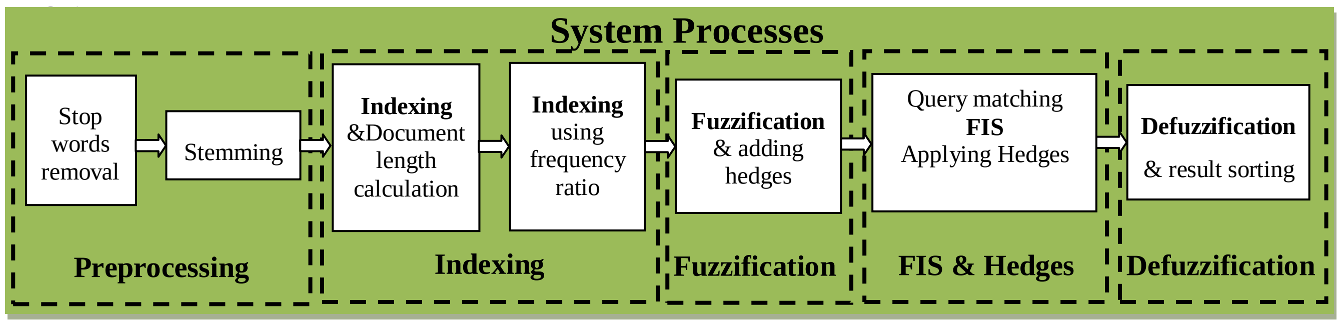
Chapter 3

The Proposed solution

The system architecture and processes will be explained in this chapter. The system has five main tasks which are : 1) Preprocessing 2)Indexing 3)Fuzzification 4)FIS 5)Defuzzification Each of these tasks has processes to achieve it. In section 3.1, an explanation of each of these tasks is give. This is followed by the system chart and algorithm provided in section 3.2.

3.1 Proposed IR system Architecture:

The IR-FIS model has the following processes that achieve five main tasks preprocessing, indexing, fuzzification, rules application and defuzzification as shown in Figure



System Processes
Stop words removal
Stemming
Preprocessing
Indexing &Document length calculation
Indexing using frequency ratio
Indexing
Fuzzification & adding hedges
Fuzzification
Query matching FIS Applying Hedges
FIS & Hedges
Defuzzification & result sorting
Defuzzification

Figure 3.1 System processes

Processing goes as follows; first, removing the stop words from the documents then returning the document terms to their origin or stem. Second, indexing should start. The final index should be reached based on the frequency ratio which should be fuzzified to reach three fuzzy values for each term. These fuzzy values represent the level of term frequency in the document. Also hedges are added to the query to give more weight for the important query terms. The final stage is the query matching and defuzzification to reach a relevance value for each document that is used for ranking. The best 1000 documents should be choosen after sorting the documents in descending order based on their rank. Processes details are provided in the coming sections.

3.1.1 Preprocessing and indexing

The preprocessing of documents in the corpus, as well as the queries, helps improve the efficiency of the retrieval process. Therefore, it was decided to implement the same indexing steps mentioned earlier in the indexing algorithm 2.2 in the IR-FIS system. The first step was to get rid of the stop words, for more elaboration on the stop word removal algorithm used see appendix B. The second is stemming the corpus documents and the queries. The Porter stemmer [25] was chosen for stemming the corpus and queries as it is considered to be the most commonly used stemmer for English documents and proved to be very effective. The Porter stemming algorithm is a process for removing the affixes from words in English to return them to their stem[17]. Stemmers usually use language specific rules.

A stemming example can be having the group of terms (e.g., oper**ate** oper**ating** oper**ates** oper**ation** oper**ative** opera**tives** oper**ational**) stemmed by the Porter stemmer to one common stem word which is “oper”. The suffix stripping program will usually be given an explicit list of suffixes, and the conditions under which each suffix may be removed from a word to leave a valid stem [17]. For more elaboration on the Porter stemmer see appendix B.

3.1.2 Fuzzification:

In the fuzzification stage, the main document features that have direct effect on the relevance had to be chosen to be used as fuzzy variables that quantify the documents before processing. Two types of features were chosen, term related features and document related features. The term related features are the ***term frequency ratio (tfr)***, the ***document frequency (df)***, and the document query ***overlap*** , while the document related one was the ***title***.

It was evident that using the tf directly is not effective and that most of those who used fuzzy logic usually use log in order to make using tf more effective. Therefore, the ***term frequency ratio*** variable was used instead which is calculated as follows :

$$\boldsymbol{Term\ frequency\ ratio(tfr)} = \frac{\text{Term frequency in the document}}{\text{Document length}} \qquad [3.1]$$

Furthermore, the ***document frequency*** variable is used instead of the inverse document frequency since the meaning of these parameters was handled implicitly by using the fuzzy rules.

$$\boldsymbol{Document\ frequency(df)} = \frac{\text{Number of documents the term appeared in}}{Total\ number\ of\ documents} \qquad [3.2]$$

The overlap variable is calculated as follows:

$$\boldsymbol{Overlap} = \frac{\text{number of query terms in the document}}{\text{query length}} \qquad [3.3]$$

Finally, ***title*** term variable is quantified as follows:

$$\boldsymbol{Title} = \text{Number of times the term occured in the title} \qquad [3.4]$$

For each of these fuzzy variables three linguistic values high, medium, and low were used. Different membership functions for each variable like the bell shaped, S-shaped, L-shaped, trapezoidal and triangular functions were tried. Figure 3.2 and 3.3 show the main membership functions used for each of the four fuzzy variables and here is the function used to formulate the used S-shape function [18].

S-shape function:

$$f(x;a,b)=\begin{cases}0, & x\le a\\ 2\left(\dfrac{x-a}{b-a}\right)^2, & a\le x\le \dfrac{a+b}{2}\\ 1-2\left(\dfrac{x-b}{b-a}\right)^2, & \dfrac{a+b}{2}\le x\le b\\ 1, & x\ge b\end{cases} \qquad [3.5]$$

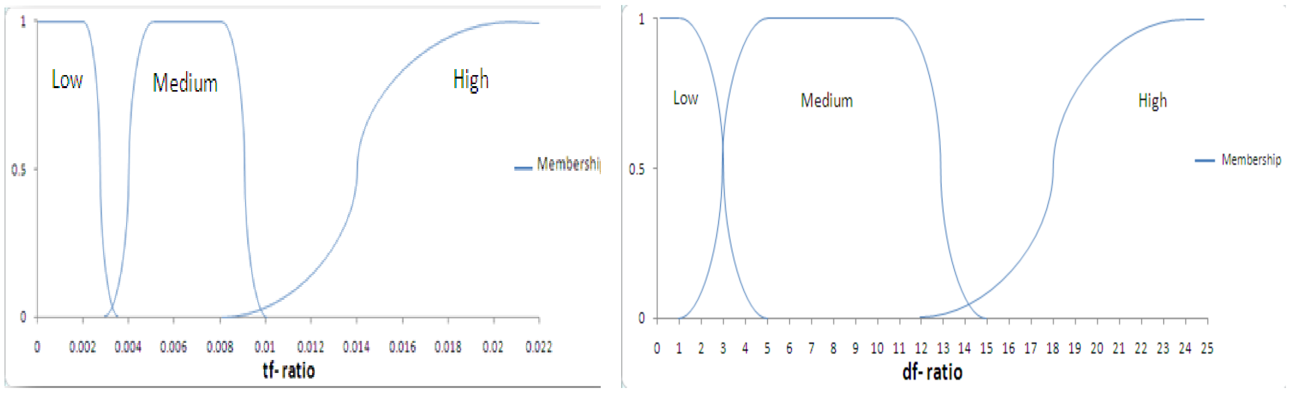


Figure 3.2: The S-shape and bell-shape membership functions for the tf ratio and df ratio

As can be seen in Figure 3.2, the S-shape function was used for the two fuzzy variables, term frequency ratio and the document frequency ratio. Figure 3.3, on the other hand, shows how the title term frequency and the overlap membership values are decided. The medium fuzzy value was the only considered value for the term frequency in the title, the membership is described as follows:

```
if (term frequency in the title =1) then medium-title(tf ) = 0.7
else if (term frequency in the title =2) then medium-title(tf) = 0.8
else if (term frequency in the title = 3 or 4) then  medium-title(tf) = 0.9
else if (term frequency in the title >=5) then medium-title(tf) = 1
```

while the overlap membership in the medium overlap fuzzy set is decided as follows:

```
if (overlap-ratio < 0.7) then medium-overlap(ovlpratio)=ovlpratio+0.3
else medium-overlap(ovlpratio)=1
```

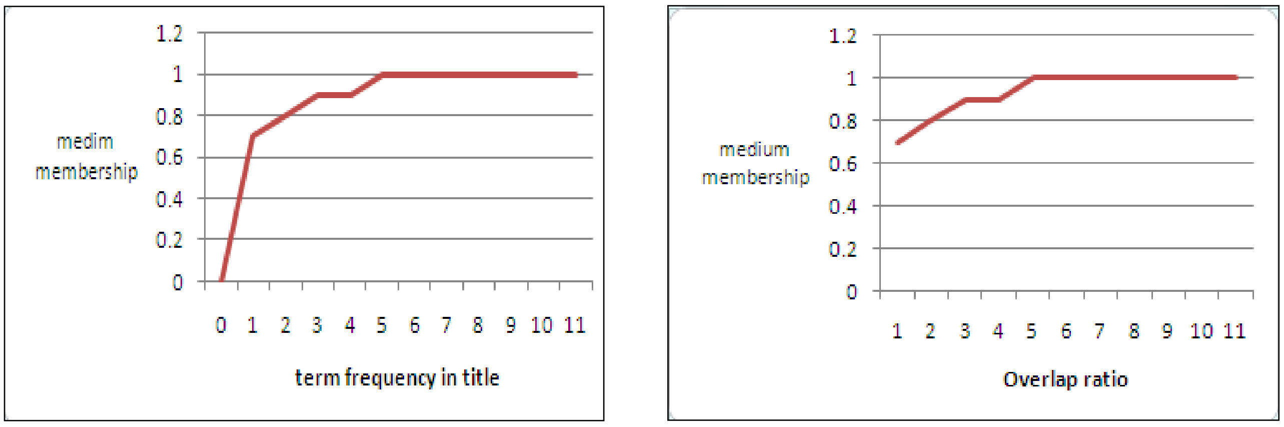


Figure 3.3 : The medium membership functions for the title and overlap

3.1.3 Fuzzy inference system (FIS):

The rules shown in Figure 3.4 were used in the fuzzy inference system of IR-FIS. One of the main objectives of this study was to explicitly specify the assumptions underlying the retrieval system. In the majority of the retrieval systems these assumptions are implicitly defined and buried in their program code. Most of these rules match the common sense behind information retrieval and relevance judgment. Some other rules came after experimenting and getting the best performance like the overlap and title rules; explanation of the experiments conducted with these rules can be found in chapter 4.

The system **rule base**
if (df is low) and (tf is high) then relevance is high
if (df is medium) and (tf is high) then relevance is medium
if (df is high) and (tf is high) then relevance is medium
if (df is low) and (tf is medium) then relevance is high
if (df is medium) and (tf is medium) then relevance is medium
if (df is high) and (tf is medium) then relevance is low
if (df is low) and (tf is low) then relevance is medium
if (df is medium) and (tf is low) then relevance is low
if (df is high) and (tf is low) then relevance is low
if (overlap is medium) then relevance is medium
if (title is medium) then relevance is medium

Figure 3.4: The intuitive rule base of the FIS

3.1.4 Using hedges:

The fuzzy system has the facility of defining linguistic modifiers of fuzzy values like the hedges [30]. Hedges are terms like “a little”, “very”, “slightly”, “extremely”, “somewhat”, or “indeed” that can be added before the query term to emphasize the degree of the term importance to the query. They are used to help keep close to natural language, and to use mathematical expressions representing the truth value of fuzzy statements. One example was mentioned by James in [12] where he transformed the statement "Jane is old" to "Jane is ***very*** old." He explained how the hedge "***very***" is usually defined and how this affects the truth value mathematically as follows:

m"***very***"A(x) = mA(x)^2 Thus, if m**OLD**(Jane) = **0.8**, then m***VERY*****OLD**(Jane) = **0.8**^{2}=0.64[12].

As hedges are used to change the truth values according to mathematical functions, the hedges’ role in the IR-FIS system is to strengthen the impact of query terms and to weaken other terms according to the level of importance of the term to the query stated by the user. Two main hedges were used before the query terms the first is “***slightly***” which means that the query term is not that essential for the query or user information need and the second is “***Indeed***” which was used to stress the importance of essential query terms. The “*indifferent*” hedge was used only as a place holder infront of neutral query terms. At the beginning, the operator for the hedge “***slightly***” was high_relevance(d)$^{1/3}$ which was intended to ***weaken*** the high relevance of the documents that are relevant to the query term preceded by the hedge “***slightly***”. On the other hand, the high_relevance(d)^3 operator was used to ***strengthen*** the high relevance of the documents that have the query term preceded by the hedge “***Indeed***”. This was based on the suggested hedges by Jan Jantzen[13], who used the hedge *extremely* a = a^3 and *slightly* a = a$^{1/3}$; *Indeed* was used instead of *extremely* to match the meaning of the hedge in the query context.

After conducting experiments, as will be elaborated in chapter 4, it was found that the performance after adding hedges has improved only by +0.015 as in Table 4.13, which necessitated removing the hedges and repeating the experiment. It was discovered that removing the hedges improved performance by +0.037 as in Table 4.15. After trying to analyze the results, It was noticed that Jan’s way of handling hedges was counterintuitive to the role of hedges in IR. In the field of IR the main objective behind using hedges is to increase or decrease the relevance of the document that has a query term based on the importance of that term for the query. Therefore, documents retrieved based on a query term that has high importance should have higher relevance than those retrieved based on a less important term. In this context it was found that using hedges in the same way that is used in other fields is counterintuitive. As explained in [13], hedges like ***Indeed*** and ***slightly*** are translated into the mathematical function 3.6. If we apply these hedges in this way to the high relevance value in IR-FIS, it will lower the importance of the important terms and increase the relevance of less important terms which is the opposite of what we want to achieve.

So using hedges in IR and queries as everybody else used it was unreasonable. The reason behind this is that in IR-FIS, hedges were used to affect the high relevance value since this matched the meaning of hedges as an importance modifier. There are two rules in IR-FIS that contribute to the high relevance which are:

if (df is low) and (tf is high) then relevance is high

if (df is low) and (tf is medium) then relevance is high

This means that the high relevance is calculated by taking the maximum of two minimums as in the following formula:

hrlv = max(min(ldf,htf),min(ldf,mtf))

Knowing that the values for *df and tf are between zero and one* since they are ratios as apparent in Figure 3.2 then the expected values for the <u>*high relevance (hrlv) will range also from zero to one*</u>. Therefore, if we use the hedges in the IR field as it is commonly used in other fields like in [13] where :

Indeed a = a^3 and ***slightly a = $a^{1/3}$*** [3.6]

➔ ***Indeed hrlv = $hrlv^3$*** and ***slightly hrlv = $hrlv^{1/3}$***

we will be going against the core goal behind using the hedges. For example, if the user says “I am interested ***indeed*** in information retrieval” and the document’s ***hrlv*** to “information retrieval” is weighted **0.8** then its membership after utilizing the <u>***Indeed***</u> hedge will be **$(0.8)^3$** which is **0.512.** In this case using hedges <u>*lowered*</u> the relevance value. On the other hand, when the user says “I am interested <u>***slightly***</u> in data mining” and the document’s high relevance to “data mining” is 0.8 then after considering the slightly hedge, ***hrlv*** will be **$(0.8)^{1/3}$** which is equal to **0.93.** Here using hedges led to a <u>*higher*</u> relevance. Using hedges in this way pushes down the membership value of important terms and increases the importance of the less important terms which is the ***opposite*** of what we actually need. Therefore the current functions and definition of hedges in fuzzy logic is counterintuitive to what IR needs. This necessitated customizing the use of hedges in IR-FIS to closely match the purpose behind using it in the IR field. The functions used for the two hedges *Indeed* and *slightly* had to be switched in IR-FIS so they are redefined as follows:

Indeed a = $a^{1/3}$ and ***Slightly a = a^3*** [3.7]

The *indifferent* hedge was used in IR-FIS as a place holder for processing purposes with terms that are neutral and should not have any hedges; see Appendix A section A.5. After repeating the experiments, this helped in having performance that is more than *three times* better than that achieved by the old function 3.6 as will be discussed in chapter 4; see Table 4.15.

3.1.5 Defuzzification:

After running the inference engine the system comes out with three values of relevance high, medium and low per document. Now the task is to defuzzify these values and come up with only one value that represents the relevance of the document to the query. For simplicity, the weighted average defuzzification method was chosen to reach one relevance value per document as follows:

$$\text{Relevance} = \frac{1\times\mathbf{hrlv}+0.72\times\mathbf{mrlv}+0.1\times\mathbf{lrlv}}{1+0.72+0.1} \quad [3.8]$$

(where hrlv, mrlv, and lrlv are the high, medium and low relevance values respectively and 1, 0.72 and 0.1 are their weights in IR-FIS). These weights have been experimentally decided and proved to achieve high performance as in Table 4.11.

3.2 Proposed processes :

The IR-FIS code was written using separate programs to avoid memory problems that were inevitable especially when trying to process 100000 text files in memory at the same time. The following figure shows the main system processes and the data flow between them.

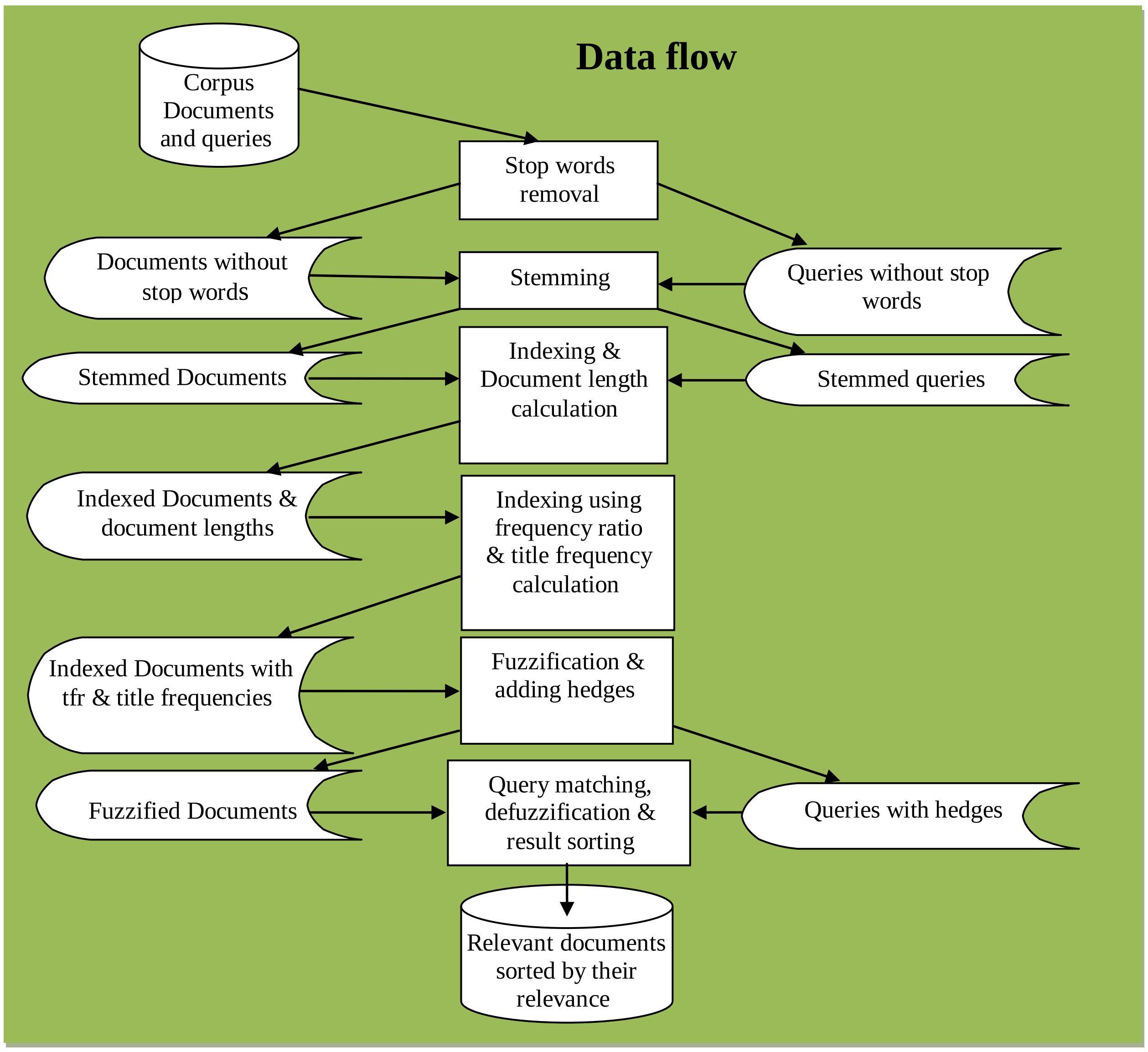

Data flow
Corpus Documents and queries
Stop words removal
Documents without stop words
Stemming
Queries without stop words
Stemmed Documents
Indexing & Document length calculation
Stemmed queries
Indexed Documents & document lengths
Indexing using frequency ratio & title frequency calculation
Indexed Documents with tfr & title frequencies
Fuzzification & adding hedges
Fuzzified Documents
Query matching, defuzzification & result sorting
Queries with hedges
Relevant documents sorted by their relevance

Figure 3.5: Data flow diagram of the system

The processing was divided into four main stages:

1. The first stage was to preprocess the document corpus and queries and to get rid of the stop words which are frequent words that don’t have any effect on the retrieval process
2. Then the second stage was to apply the *Porter stemmer* on these preprocessed files in order to return the words that have the same origin but are available in different formats to their stem.
3. The *third stage was indexing and fuzzifying* and this is by creating two files. The first file is the *index file* that has the list of stemmed terms along with the document ids they appeared in and the frequency of their occurrence in these documents. The second file is the *document length file* that has the document ids and their lengths. The two files are used to create an *index* that includes the stemmed *term*s, document *id*s they appeared in and their *frequencies* as ratios to these document lengths. This index file is used to produce two other files after fuzzification of the term frequency ratio and the document frequency according to the triangle functions shown in Figure 4.3 and 4.4 or the S-shape functions shown in Figure 3.2. The first file is the fuzzified term frequency ratio file that has the term, the document id it appeared in and the three fuzzy values representing the degree of the term frequency ratio (tfr); low, medium and high tfr. The other file is the fuzzified document frequency file that has the term, and three fuzzy values that represent the degree of the term document frequency (df); low, medium and high df. Concurrently, hedges are added to the query files in order to prepare for the query matching process. The title file is also created which has the term, the id of the document it appeared in as part of the title and the number of times the term appeared in the document title.
4. The fourth is *query matching* stage which starts when all the relevant documents to the query terms are retrieved with their (tf) fuzzified values from the *fuzzified tf ratio file* and then their fuzzified (df) values are retrieved from the *fuzzified df file*. The query terms are then checked to see if they are in the title file, if yes then the document title frequency is retrieved and fuzzified according to the function in Figure 3.3. The overlap ratio is also calculated, as in formula 3.3, for each document by counting the number of query terms that appeared in the same document and then dividing this number by the number of terms in the query (query length) and this is considered the overlap ratio which is then fuzzified according to the function in Figure 3.3.

   Now that the fuzzification of all four variables, (term frequency ratio, document frequency ratio, overlap ratio and title frequency), are done the inference stage can start by applying the rules that appear in Figure 3.4 to these fuzzy variables to reach fuzzified values to relevance which gives three membership values (high relevance, medium relevance and low relevance). These three values are then defuzzified using the weighted average to reach one relevance value per document. Finally, the query relevant documents are sorted in descending order according to their relevance values and the best 1000 documents are returned.

```
Stop words removal from the document corpus and the queries
Stemming the document corpus and the queries
Indexing & document length calculation
Indexing using term frequency ratio
 Fuzzification (whether using the triangle or the S-shape function) which has three steps:
     1.  Create fuztf file that has the term, doc, three fuzzy values (high-tf, medium-tf , low-tf)
     2.  Create fuzdf file that has the term, doc, three fuzzy values (high-df, medium-df , low-df)
     3.  Create title file that has the term, doc, number of times the term appeared in document title
Adding hedges to queries
Open fuztf , fuzdf, title
 for each quey (Q) in the corpus
    for each Query_term (qt) in the Q
     {  if (qt = index term (t) in the fuzzified index)
          Relevant_doc_list   = Get the list of documents associated with t
          for each document(d) in Relevant_doc_list do the following
        {  find qt in fuztf , fuzdf, title to get the tf and df fuzzy values the title term frequency
               get low_tf ,     medium_tf,     high_tf from fuztf
               get low_df ,     medium_df,     high_df from fuzdf
           apply the inference rules
               high_rlv       = max(min(low_df,high_tf), min(low_df,medium_tf))
               medium_rlv   = max(min(high_df, high_tf),  min(medium_df, high_tf),
                                   min(low_ldf, low_tf),  min(medium_df, medium_tf))
               low_rlv        = max(max(min(high_df,medium_tf), min(medium_df,low_tf),
                                          min(high_df, low _tf))
           if qt is in the title file then title-f= title frequency
           medium_title = title membership function (title-f)   as in [3.2.3]
           medium_rlv = max(medium_rlv,medium_title)
           consider the overlap parameter
                 overlap_count = count different query terms in the current document
                 overlap_ratio   = overlap_count/number of terms in the query
           fuzzify the overlap parameter
                 if (overlap_ratio < 0.7) then medium_overlap=overlap_ratio+0.3
                                              else medium_overlap=1
           combine the medium_rlv with the medium_overlap
                       medium_rlv= max(medium_rlv, medium_olp)
           consider the hedges and change the high_rlv accordingly
                       if  ( query term hedge ="slightly") then high_rlv=(high_rlv)^3
                       else if (query term hedge ="Indeed") high_rlv=(high_rlv)^(1/3)
            deffuzify to reach one value representing the relevance of the document
                   xh=1,xm=0.72,xl=0.1;
                  docwight = (high_rlv*xh + medium_rlv*xm + low_rlv*xl)/(xh+xm+xl)
            if (doc in document-list) then add the new docwight to the old one
            else add the document and docwight to document-list of relevant documents
          } end of the list of relevant documents based on the term
        } end of query
keep qid and its document-list } end of the queries list
Sort the queries by their ids as well as documents according to relevance in descending order
Save the first 1000 relevant documents for each query in the result file.
Compare the results using TREC eval to Lucene's
```

Algorithm 3.1: System processes

Chapter 4

Experiments and result analysis

In this chapter an overview is given for the main experiments that were done, the purpose behind each, results found and the result discussion. The following section describes the experiment setup which include the document corpus, the queries used for testing and the program used to compare the IR-FIS with Lucene results.

4.1 Evaluation criteria

Around 160 different experiments were conducted on 100,000 news articles from CLEF 2009 INFILE collection. Fifty queries were used in all the experiments and their retrieved documents were ranked. These experiments can be grouped under the following areas:

1. *Finding the **best membership functions** for the four fuzzy variables used which are the df-ratio, tf-ratio, title and overlap*
2. *Experimenting with **FIS rules***
3. *Finding the **best weights** for the weighted average defuzzification function*
4. *Finding effective **Hedge function** for IR*
5. ***Combining** the best found membership functions, FIS rules, weights and hedge function together and **comparing the results** to other IR systems.*

The ranked lists resulting from the experiments were submitted to an evaluation program called TREC-Eval. TREC-Eval is a standard program used in many experiments including those of Text Retrieval Evaluation Conference (TREC) to measure the performance of the systems. Precision was the main measure used in evaluating the performance. The input to TREC-Eval is the ranked retrieved documents and the output is three reports showing the effectiveness of the retrieval process. This evaluation reports have the following statistics:

Table 4.1: The structure of Report one of the Trec-Eval evaluation Software[21]

Number of topics	*This is the total number of topics(50)*
Retrieved	*The number of documents returned by the IR system*
Relevant	*Total number of relevant documents*
Rel-ret	*Total number of relevant documents returned by the IR system.*
R-Precision for a run	*The precision after R documents have been retrieved, where **R is the number of relevant documents for the topic**. computed by taking the mean of the R-Precisions of the individual topics in the run.* *For **example**,* *assume a run consists of three topics:* *1)**Topic1** 20 relevant docs 2)**Topic2** 40 relevant docs 3)**Topic3** 10 relevant docs* *If the retrieval system returns:* *1)**Topic1** 18 relevant docs 2)**Topic2** 8 relevant docs 3)**Topic3** 9 relevant docs* *The run's **R-Precision** = (18/20+8/40+9/10)/3=**0.67***
Average precision	*It is the average of the precision value obtained after each relevant document is retrieved.* *For **example**,* *If the query has three relevant docs at ranks 3, 5, and 7.* *The actual precision of:* *doc1= 0.9, doc2=0.8, doc3=0.6* *Average precision over all relevant documents (the mean)= 0.77.*

Table 4.2: *Precision at different Recall levels (TREC-Eval Report two) [21]*

Recall	*Precision*	
0.0	0.6147	*Each recall-precision average is computed by summing the interpolated precisions at the specified recall cutoff and dividing by the number of topics.*
0.1	0.5478	
0.2	0.4867	
0.3	0.431	
0.4	0.3905	
0.5	0.3577	
0.6	0.3265	
0.7	0.2728	
0.8	0.2435	
0.9	0.1989	
1.0	0.1103	

Table 4.3: Precision at Document cut offs (TREC-Eval Report three) [21]

Document cut off	*Precision*	
5	*0.4087*	*Each document precision average is computed by summing the precisions at the specified document cutoff value and dividing by the number of topics*
10	*0.4043*	
15	*0.3812*	
20	*0.3685*	
30	*0.3391*	
100	*0.1896*	
200	*0.1175*	
500	*0.0556*	
1000	*0.0291*	

4.2 Finding best membership functions

Selecting the best membership function is essential for reducing the error rate in the fuzzy IR system. Using an improper function leads to wrong fuzzy input values, this results in wrong fuzzy output values that will lead to high error when defuzzified [24]. The objective of the first group of experiments was to find the best membership function for each fuzzy variable which was not a straight forward process. Not only different membership functions were tried but also different boundaries for the fuzzy variables before coming out with the most effective ones. Below the experiments for finding the membership functions for the system's fuzzy variables: title, overlap, tf, and df are discussed.

4.2.1 The title membership function

To decide the most effective membership function for the title variable, many functions had to be tried and their effect on performance had to be analyzed. Regarding the **title**, the membership depends on the frequency of term appearance in the document title which is then fuzzified using the membership function which decides the degree of membership in the fuzzy sets. In the first experiment, the *old title function* in Table 4.1 was used which considered the title high and medium fuzzy values in fuzzification. Both of these fuzzy values were used to decide the relevance of the documents. This way didn't prove to be as effective as using what was referred to as the *new title function* in Table 4.1. The new function considered the medium fuzzy value alone with the membership function shown in Figure 3.3. This improved the overall performance as shown in Table 4.4.

Old title function	New title function

if (titletf ==1)
then **high_title(titlef)=0.7** and **medium(titlef)=1**
if (titletf is 2)
then **high_title(titlef)=0.8** and **medium(titlef)=0.9**
if ((titletf is 3)and (titletf is 4))
then **high_title(titlef)=0.9** and **medium(titlef)=0.8**
if (titletf >=5)
then **high_title(titlef)=1** and **medium(titlef)=0.7**
	if (titletf is 1) then **medium(titlef)**=0.7
	if (titletf is2) then **medium(titlef)**=0.8
	if ((titletf is 3)and (titletf is 4)) then
	medium(titlef)=0.9
	if (titletf >=5) then **medium**1

Figure 4.1: Old and new title fuzzy membership functions

Table 4.4: Comparison between old and new title fuzzy membership functions

Experiment results				
				best one
run		Lucene	title-old	title-new
number of queries		46	47	47
Retrieved:		41393	40692	40692
Relevant:		1547	1597	1597
Rel_ret:		1340	1267	1306
Average precision		0.3498	0.3104	0.3167
R-Precision		0.3471	0.3358	0.3432
Interpolated run		Lucene	title-old	title-new
	0	0.6147	0.6125	0.6125
	0.1	0.5478	0.5554	0.5554
	0.2	0.4867	0.4639	0.4639
	0.3	0.431	0.4265	0.4265
	0.4	0.3905	0.3864	0.3864
	0.5	0.3577	0.3495	0.3495
	0.6	0.3265	0.2573	0.2573
	0.7	0.2728	0.2233	0.2233
	0.8	0.2435	0.1744	0.1744
	0.9	0.1989	0.1021	0.1021
	1	0.1103	0.0425	0.0425
Precision at document cutoff:				
run		Lucene	title-old	title-new
docs:	5	0.4087	0.4085	0.4128
docs:	10	0.4043	0.4128	0.4191
docs:	15	0.3812	0.3688	0.3787
docs:	20	0.3685	0.3479	0.3532
docs:	30	0.3391	0.3156	0.3206
docs:	100	0.1896	0.1781	0.1794
docs:	200	0.1175	0.107	0.1078
docs:	500	0.0556	0.0494	0.0509
docs:	1000	0.0291	0.027	0.0278

4.2.2 The overlap membership function

For the overlap, experiments were conducted with four different fuzzy membership functions. Each function was tried, results were compared and the most effective function was chosen. These four functions are the triangle function which was tried twice with different interval boundaries of each of the three fuzzy values. The second function was the S-shape function where the S-shape was used to map the overlap ratio into a degree of the three fuzzy values low, medium and high overlap and all the three values were used in deciding the document relevance.

The last way that was used to estimate the relevance was to concentrate only on one overlap value that serves as a threshold that decides the term degree of membership in the overlap set which was the medium value. If the overlap ratio is less than 0.7 then the medium value of the overlap is the overlap-ratio+0.3 otherwise it is 1. When this proved to be successful, the overlap ratio medium fuzzy value was considered alone to affect the relevance of the document. The comparison between the four membership functions are shown in Figure 4.2. After seeing Table 4.5 it would be clear that the last function which considered only the medium fuzzy value was the most effective one.

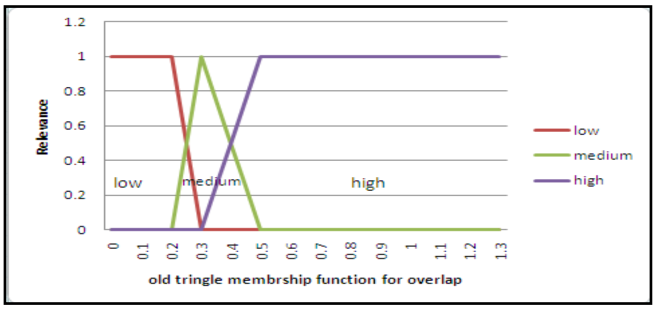
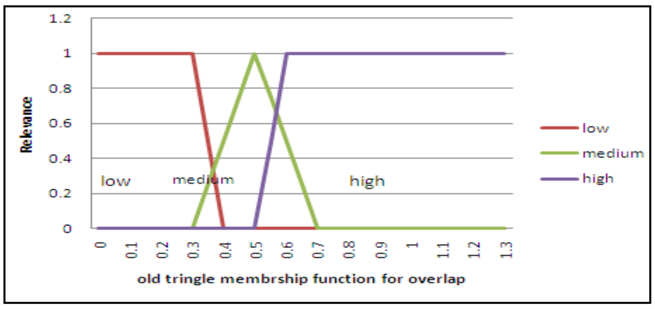
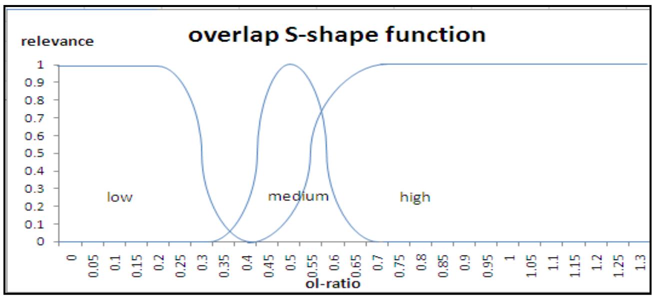
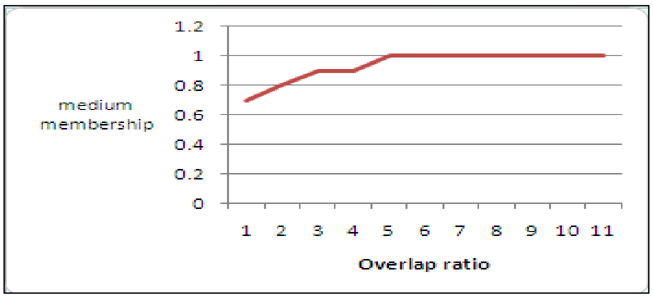
Functions tried
Old triangle function for over lap 
New triangle function for over lap 
S-shape function with over lap 
Overlap using the medium membership value 

Figure 4.2: Comparison between overlap fuzzy membership functions

Table 4.5: Comparison between overlap fuzzy membership functions runs

Experiment results						
						best one
run		Lucene	ol-old-tri	ol-new-tri	ol-S-shape	ol-medium
number of queries		46	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597
Rel_ret:		1340	1247	1278	1303	1306
Average precision		0.3498	0.2815	0.2841	0.2859	0.3167
R-Precision		0.3471	0.3116	0.3093	0.3146	0.3432
Interpolated						
run		Lucene	ol-old-tri	ol-new-tri	ol-S-shape	ol-medium
	0	0.6147	0.5958	0.5927	0.5841	0.6125
	0.1	0.5478	0.5243	0.5146	0.5059	0.5554
	0.2	0.4867	0.447	0.4379	0.4421	0.4639
	0.3	0.431	0.4019	0.4053	0.4185	0.4265
	0.4	0.3905	0.3615	0.3649	0.3782	0.3864
	0.5	0.3577	0.3197	0.3274	0.3351	0.3495
	0.6	0.3265	0.2482	0.2482	0.2466	0.2573
	0.7	0.2728	0.2137	0.2163	0.2138	0.2233
	0.8	0.2435	0.1585	0.1618	0.1623	0.1744
	0.9	0.1989	0.0922	0.0927	0.0952	0.1021
	1	0.1103	0.0332	0.0415	0.0403	0.0425
Precision at document cutoff:						
run		Lucene	ol-old-tri	ol-new-tri	ol-S-shape	ol-medium
docs:	5	0.4087	0.3702	0.3702	0.3745	0.4128
docs:	10	0.4043	0.3489	0.3511	0.3532	0.4191
docs:	15	0.3812	0.3206	0.3234	0.3333	0.3787
docs:	20	0.3685	0.3085	0.3096	0.316	0.3532
docs:	30	0.3391	0.2908	0.2965	0.2993	0.3206
docs:	100	0.1896	0.1683	0.1685	0.1691	0.1794
docs:	200	0.1175	0.1035	0.1043	0.1052	0.1078
docs:	500	0.0556	0.0479	0.0489	0.0494	0.0509
docs:	1000	0.0291	0.0265	0.0272	0.0277	0.0278

Note that fifty queries were tried but Lucene returned no results for 4 queries that was why 46 queries only were considered. With IR-FIS only three queries returned no results that is why 47 queries were considered. This makes IR-FIS stronger than the Lucene search engine since it succeeded in finding results for 47 instead of 46 queries.

It is clear from the above results that using the S-Shape function and the triangle function with the overlap ratio didn't help the overall performance as was done by the suggested membership function shown in the Figure 3.3. When the overlap ratio was considered as the sole fuzzy variable and the experiments were repeated, the same conclusion was reached. The effect of using hedges was investigated with these functions and it was also clear that using hedges with the overlap didn't enhance the performance as shown in the following table. The reason behind this is that we only considered the medium overlap value as shown in Figure 3.3. This affects the medium relevance while the hedges affect the high relevance only.

Table 4.6: Comparison between the overlap fuzzy membership functions runs when only the overlap parameter is considerd

run		Overlap S-shape	Overlap medium	Overlap triangle	Overlap best + hedges
number of queries		47	47	47	47
Retrieved:		40692	40692	40692	40692
Relevant:		1597	1597	1597	1597
Rel_ret:		1099	1100	1099	1100
Average precision		0.1946	0.1946	0.1945	0.1946
R-Precision		0.2173	0.2173	0.2173	0.2173
Precision at document cutoff:					
docs:	10	0.4043	0.2638	0.2638	0.2638
docs:	500	0.0437	0.0437	0.0437	0.0437
docs:	1000	0.0234	0.0234	0.0234	0.0234

The title was also considered as the only fuzzy variable and the experiments were repeated using the title membership function described in Figure 3.3; with hedges and without hedges. Here using the hedges with the title didn't have any effect on the performance as illustrated in the following table. The reason behind this is that we only considered the medium title value as shown in Figure 3.3, while the hedges affect the high relevance only.

Table 4.7: Comparison between the title fuzzy membership function with and without hedges when only the title parameter is considerd

.

run		title	Title+hedges
number of queries		47	47
Retrieved:		40692	40692
Relevant:		1597	1597
Rel_ret:		1098	1098
Average precision		0.1926	0.1926
R-Precision		0.2121	0.2121
Precision at document cutoff:			
run		title	Title+hdges
docs:	10	0.2532	0.2532

4.2.3 The df-ratio membership function

For fuzzification more than one membership function was tried for the (tf) and (df) before the best ones were chosen. For the document frequency ratio (df) the triangle function was tried as shown in Figure 4.3 and the S-shape function shown in Figure 3.2 with and without hedges. It was found that the S-shape function with hedges had a better performance than the triangle function with and without hedges as shown in the Table 4.8.

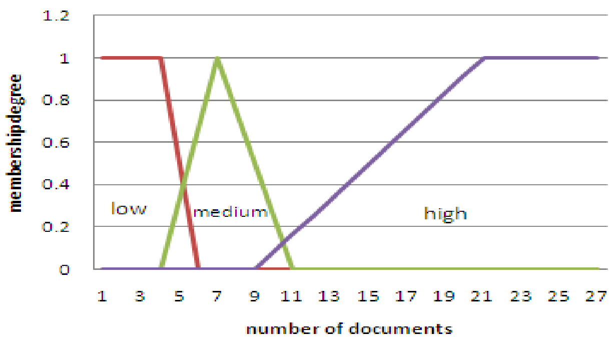


Figure 4.3: the triangle, L-shape membership functions for df ratio

Table 4.8 : Comparison between df fuzzy membership functions (triangle and S-shape) with and without hedges when only the df parameter is considerd

run		df-ratio triangle	df-ratio S-shape	df-ratio triangle with hedge	df-ratio S-shape with hedges
number of queries		47	47	47	47
Retrieved:		40692	40692	40692	40692
Relevant:		1597	1597	1597	1597
Rel_ret:		1103	1098	1103	1106
Average precision		0.1954	0.1948	0.1954	0.202
R-Precision		0.2238	0.2165	0.2238	0.2299
Precision at document cut off:					
run		df-ratio triangle	df-ratio S-shape	df-ratio triangle with hedge	df-ratio S-shape with hedges
docs:	5	0.2766	0.2766	0.2766	0.2894
docs:	10	0.2638	0.2638	0.2638	0.2745

4.2.4 The tf-ratio membership function

The same thing was done with the term frequency ratio (tf), the triangle function and the S-shape function were tried. The triangle function shown in Figure 4.4 performed better with regard to the average precision while the S-shape function shown in Figure 3.2 performed better on the document cutoffs from P5 to P30 and the R-precision with hedges; see table 4.10. Reaching the boundaries of each of the fuzzy values was done after considering the distribution curve for the document frequency and the term frequency ratio as shown in appendix A to allow prediction of the intervals for the low frequencies, the medium ones and the high ranges.

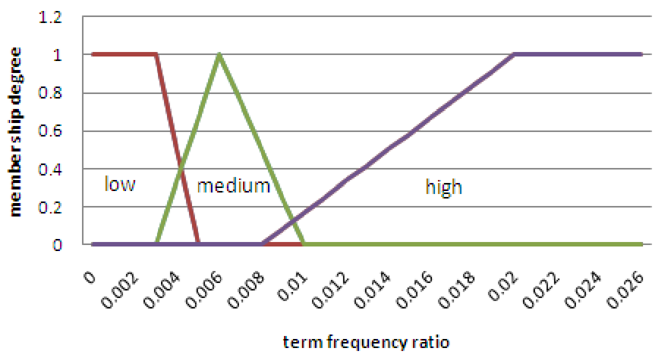


Figure 4.4: the triangle, L-shape membership functions for tf ratio

Table 4.9: How using hedges with tf S-shape function outperformed the function without hedges

tf S-shape	P30	P100	average precision	R-Precision
without hedges	0.3106	0.1689	0.294	0.308
with hedges	**+0.00225**	**+0.00118**	**+0.00034**	**+0.00130**

On the other hand, it was found that using hedges with the S-shape function with the tfr variable did improve the performance on the document cut offs P30 and P100 in addition to the average precision and R-Precision compared to the S-shape function performance without hedges as shown in table 4.9.

Table 4.10: Comparison between tfr fuzzy membership functions (triangle and S-shape) with and without hedges when only the tfr parameter is considerd

run		tf-ratio S-shape	tf-ratio triangle	tf-ratio triangle + hedges	tf-ratio S-shape + hedges
number of queries		47	47	47	47
Retrieved:		40692	40692	40692	40692
Relevant:		1597	1597	1597	1597
Rel_ret:		1255	1247	1247	1255
Average precision		0.294	0.2962	0.296	0.294
R-Precision		0.308	0.3076	0.3076	0.308
Precision at document cut off:					
run		tf-ratio S-shape	tf-ratio triangle	tf-ratio triangle + hedges	tf-ratio S-shape + hedges
docs:	5	0.4043	0.3915	0.3915	0.4043
docs:	10	0.3915	0.3915	0.3894	0.3915
docs:	15	0.3589	0.356	0.356	0.3589
docs:	20	0.3436	0.3426	0.3404	0.3426
docs:	30	0.3106	0.3099	0.3099	0.3113

4.3 Experimenting with FIS rules:

To experiment with different retrieval assumptions, variations of the rules were done based on intuition or possible scenarios hoping for performance improvement. This was done by changing some of the rules as shown in the following table where the changed rule is referred to as the old rule and the new rule is the one tried instead:

Old rule	New rule
(low df and high tf ratio) -> high relevance	(low df and high tf ratio) -> medium relevance
(low df and medium tf ratio) -> high relevance	(low df and medium tf ratio) -> medium relevance

Figure 4.5: Old and new rules

After conducting experiments using the new rules, it was found that this didn't help improve the performance and that the first set of rules performed better so the old ones were used.

4.4 Finding the best weights for the defuzzification

The weights of the weighted average defuzzification function were changes as an attempt to improve performance. There are three weights (h for the high relevance, m for medium relevance and l for low relevance). The following table shows all the tried values, the best three values were 1 for h, 1.72 for m and 0.1 for l since they contributed to the highest precision.The relevance was calculated using the following equation:

$$\text{Relevance} = \frac{\text{h}\times\text{hrlv}+\text{m}\times\text{mrlv}+\text{l}\times\text{lrlv}}{\text{h}+\text{m}+\text{l}} \qquad [4.1]$$

Table 4.11 : Using different weights for defuzzification.

	h=0.9 m=0.9 l=0.1	h=1 m=0.9 l=0.1	h=1 m=0.8 l=0.1	h=1 m=0.7 l=0.1	h=1 m=0.71 l=0.1	**h=1 m=0.72 l=0.1**	h=1 m=0.73 l=0.1	h=1 m=0.74 l=0.1	h=1 m=0.6 l=0.1
run						**best**			
number of queries	47	47	47	47	47	47	47	47	47
Retrieved:	40692	40692	40692	40692	40692	40692	40692	40692	40692
Relevant:	1597	1597	1597	1597	1597	1597	1597	1597	1597
Rel_ret:	1267	1267	1279	1306	1306	1307	1307	1307	1266
Average precision	0.3066	0.31	0.311	0.3163	0.3166	0.3289	0.3165	0.3166	0.3095
R-Precision	0.3323	0.3358	0.3358	0.3422	0.3432	0.3501	0.3432	0.3432	0.337

4.5 Finding Hedges function for IR

As explained in section 3.5 we experimented with more than one hedge function. The original hedge function 3.6 outperformed Lucene at P10 by only + 0.0156. Therefore, it was necessary to remove the hedges and see how the system performs without them. The result of this experiment is shown in Table 4.13 which shows an improvement over Lucene by +0.0366 which is nearly *two and a half* times better than the performance with hedges that act according to the first function 3.6 and *four times better* than Rubens results. After discovering that the standard function used for hedges was counterintuitive when applied as is in the field of information retrieval, the new function 3.7 was tried.

After experimenting, it was discovered that this new approach did help the performance as shown in Table 4.12, where performance at p10 was two times better than 3.6 function results, 29% better than performance without hedges and instead of having my IR-FIS outperform Rubens FIS by four times when hedges were not used, now it is *five times better* due to introducing hedges in a more intuitive way in the IR field; see Table 4.14 and 4.15. Figure 4.6 compares the improvement achieved over Lucene search engine performance at P10 by Rubens R-FIS, my IR-FIS's two different hedge functions and without hedges. It is evident that using the 3.7 function led to results that are *five times* better than Rubens R-FIS. On the other hand, Figure 4.7 shows how my IR-FIS precision at P10 outperformed that of Lucene by 4.7% when the 3.7 hedge function was used and how the precision has improved steadily after removing the old 3.6 function and applying the 3.7 one instead; see Table 4.12.

Table 4.12 : Comparing the new hedge function 3.7 with performance without hedges

Run		Lucene	hedges function 3.6	Hedges function 3.7
number of queries		46	47	47
Retrieved:		41393	40692	40692
Relevant:		1547	1597	1597
Rel_ret:		1340	1252	1307
Average precision		0.3498	0.2952	0.3289
R-Precision		0.3471	0.3145	0.3501
Precision at document cutoff:				
run		Lucene	hedges function 3.6	Hedges function 3.7
docs:	5	0.4087	0.4128	0.4255
docs:	10	0.4043	0.4106	0.4234
docs:	15	0.3812	0.3787	0.3801

Table 4.13 : Comparing the new hedges function 3.7 with the old one 3.6.

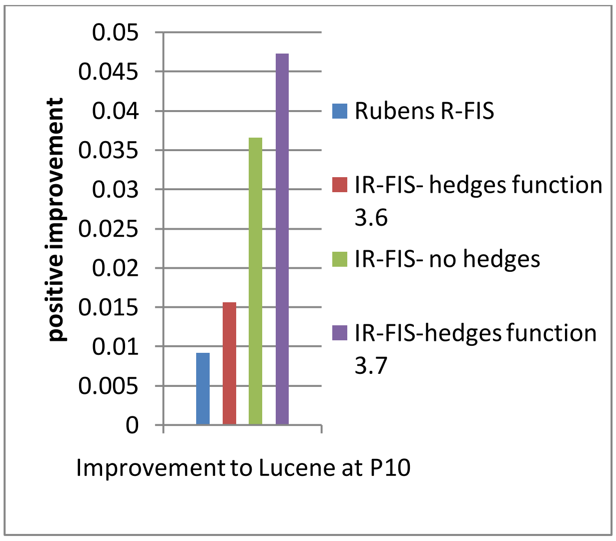
Run		Lucene	No hedges	hedges function 3.7
number of queries		46	47	47
Retrieved:		41393	40692	40692
Relevant:		1547	1597	1597
Rel_ret:		1340	1306	1307
Average precision		0.3498	0.3167	0.3289
R-Precision		0.3471	0.3432	0.3501
Precision at document cutoff:				
run		Lucene	No hedges	hedges function 3.7
docs:	5	0.4087	0.4128	0.4255
docs:	10	0.4043	**0.4191**	0.4234
docs:	15	0.3812	0.3787	0.3801

Table 4.14: Rubens R-FIS improvement over Lucene's

Tag	Topic variable	P10
Lucene	Combined	0.3984
R-FIS	Combined	+0.0092

Table 4.15 : My IR-FIS improvement over Lucene's with the different hedge functions

Tag	P10
Lucene	0.4043
IR-FIS- hedges 3.6 function	+0.0156
IR-FIS- no hedges	+0.0366
IR-FIS-hedges 3.7 function	***+0.04724***



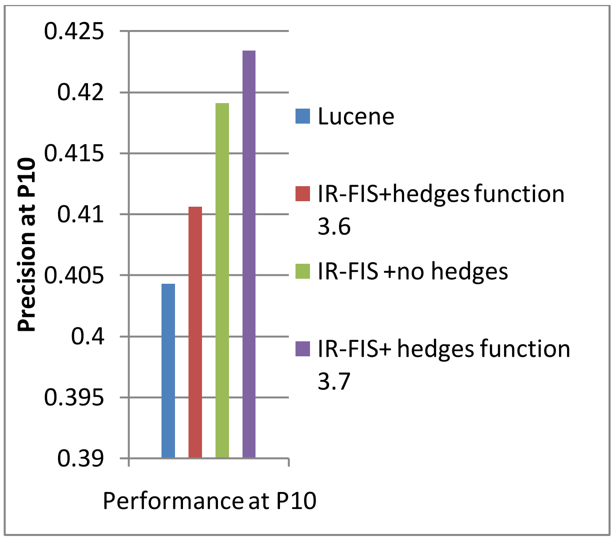


Figure 4.6 : Precision at P10 contributed by Lucene and my IR-FIS different hedge functions at P10

Figure 4.7: The improvement contributed by Rubens R-FIS and my IR-FIS different hedge functions to Lucene at P10

4.6 Combining the best of all trials

To get the best overall performance, the best found configurations so far were combined together. After experimenting with different weights, as shown in Table 4.11, the weights h = 1, m = 0.72, l = 0.1 were chosen as the weights of the defuzzification weighted average function. The best functions found so far for the four parameters tfr, df, overlap and title were combined. The triangle and S-shape membership functions were tried and the S-shape was chosen as it proved to have the highest precision especially at P10 with or without hedges. Using the hedges did improve performance when the 3.7 function was used so it was decided to use it. The results of the experiments can be summarized in the following table.

Table 4.16: Comparison between system performance using different function combinations

				best one		
run		**Lucene**	**tf-S-shape df-S-shape+no hedges**	**tf-S-shape df-S-shape + hedges**	**tf-triangle df-triangle**	**tf-triangle df-S-shape**
number of queries		46	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597
Rel_ret:		1340	1306	1307	1307	1305
Average precision		0.3498	0.3167	**0.3289**	0.318	0.318
R-Precision		0.3471	0.3432	**0.3501**	0.3448	0.3448
Precision at document cutoff:						
run		**Lucene**	**tf-S-shape df-S-shape**	**tf-S-shape df-S-shape**	**tf-triangle df-triangle**	**tf-triangle df-S-shape**
docs:	5	0.4087	0.4128	0.4255	0.417	0.417
docs:	**10**	**0.4043**	**0.4191**	**0.4234**	0.4043	0.4043
docs:	15	0.3812	0.3787	0.3801	0.3702	0.3702

Legend item	meaning
tf-S-shape df-S-shape	Using the S-shape function for both the term frequency and document frequency
tf-triangle df-triangle	Using the triangle function for both the term frequency and document frequency
tf-triangle df-S-shape	Using the triangle function for the term frequency while using frequency S-shape for the doc

The df and tf S-shape membership functions were used together and two experiments were conducted one that had the four fuzzy variables and another one where the title was excluded. It was discovered that the results didn't change and that whether the title is considered or ignored the performance is not affected. After analyzing the results, it was found that may be it is not enough to consider the title but we should try to investigate more aspects of the document structure like having the word in the first sentence of the paragraph or the position within the body of the document

Table 4.17: Comparison between system performances with and without title fuzzy variable

run		tf-S-shape df-S-shape + title	tf-S-shape df-S-shape + no title
docs:	5	0.4128	0.4128
docs:	**10**	0.4191	0.4191

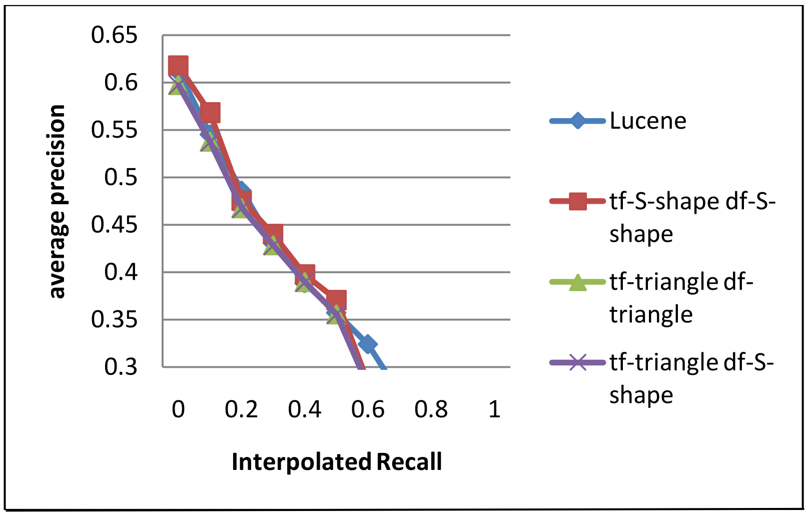


Figure 4.8: Performance based on precision at interpolated recall. The best is the S-shape membership function

As can be seen in Figure 4.8, using the S-shape function for the term frequency as well as the document frequency helped in improving precision at different interpolated recall levels over the Lucene runs.

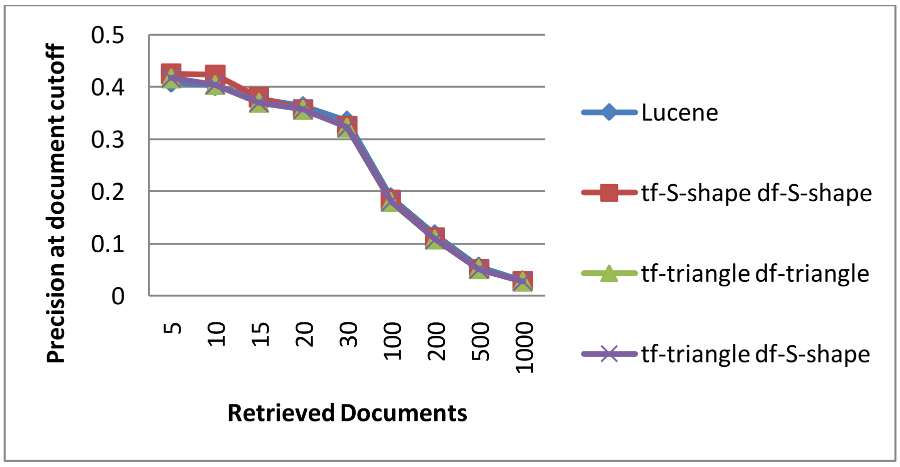


Figure 4.9: Performance based on precision at documents cut offs. The best is the S-shape membership function

Figure 4.9 also shows that using the S-shape function for the term frequency as well as the document frequency helped in improving the precision at document cutoff over the Lucene runs. One can conclude that using the S-shape leads to higher precision when compared to other membership functions and to Lucene's search engine. Therefore, it was chosen as the membership function for the tfr and df fuzzy variables. In all cases the best overlap and title functions as shown in Figure 3.3 were used with hedges.

4.7 Discussion

The main contribution of my study is that my IR-FIS system succeeded in enhancing the results achieved by previous IR systems like Lucene which did not use fuzzy logic and even the R-FIS system by Rubens in [19] which used fuzzy logic. Table 4.18 compares the performances of my IR-FIS and Rubens R-FIS approaches against Lucene's performance.

Table 4.18: how my IR-FIS outperformed Lucene & R-FIS

Tag	P10	Tag	P10
Lucene	0.3984	Lucene	0.4043
R-FIS	**+0.0092**	IR-FIS	**+0.04724**

As one can see, my ranking has outperform Lucene at documents cutoff p10 by ***4.7%*** which is ***5 times better*** improvement than the one reached by Rubens with his R-FIS which improved the performance by only *+0.0092*. If we compare my system to Rubens R-FIS, we will see that Rubens didn't use hedges. Moreover, his rule concentrated only on high fuzzy values and the negation of high by using *not* high while my rules considered the low and medium values as well which contributed to the higher performance.

My second contribution is the discovery that the standard use of hedges as it is used in other fields slightly improved the performance in the information retrieval field by 1.6% only. When only the term frequency ratio was considered it was found that using hedges with the S-shape fuzzy membership function did improve the performance as shown in table 4.9. When the way hedges were used was changed the improvement was evident and instead of outperforming Lucene by only 1.6% it became 4.7% which is nearly three times better as shown in the following table.

Table 4.19: Comparing the percentages of improvement over Lucene's by the two tried hedge functions

Tag	P10 performance improvement over Lucene's
IR-FIS-Standard hedges function 3.6	***1.6%***
IR-FIS-New hedges function 3.7	***4.7%***

Chapter 5

Conclusion and future work

5.1 Conclusion:

The main scope of this dissertation is to enhance the IR precision by using the fuzzy logic model. The Rubens R-FIS was studies and considered for improvement. To improve this system the three stages of fuzzy logic had to be improved: 1) The fuzzification 2) The FIS 3) The defuzzification. **First**, to improve the ***fuzzification***, different fuzzy membership functions for each of the four fuzzy variables tf, df, title and overlap were considered in experiments. It was discovered that using the S-shape function and the bell function lead to better performance compared to other functions like the triangle and L-shape functions. **Second**, to improve the ***FIS,*** <u>*two more fuzzy values*</u> which are medium and low were used plus the high value that was used in Rubens R-FIS. Experiments were conducted to study the effect of considering these values in the rule base. Other experiments concentrated on changing their manipulation by rules. It was concluded that using these fuzzy values lead to better results than Rubens R-FIS [19] which ignored them. Also using <u>*hedges*</u> as linguistic fuzzy operators was studied as at attempt to improve the results of the FIS. Experiments were conducted after using hedges operators as they are used in the other fields. It was discovered that this way was counterintuitive when used in the IR field. After changing the operators' definition the experiments were repeated. Precision was improved only when the definition of the hedges operators matched the reason behind using them in the IR field. Therefore, it was deduced that the hedges function should be adjusted to match the goal behind using it in IR. This change in the hedges function from 3.6 to the 3.7 function improves the performance by nearly three times. **Third**, experiments were conducted to find the best weights for the ***defuzzification*** function and the values 1 for high, 0.72 for medium, and 0.1 for low value were used as the best weights. However, using document structure, namely title, didn't contribute to performance. It was realized that it is important to investigate more contextual factors like the term position in relation to the paragraph it appears in, to the document in general or to other query terms that appear in the same document.

Through experimentation, IR-FIS system proved to outperform industry standard search engines, such as Lucene, and other similar academic systems, such as Rubens R-FIS. This was accomplished by introducing more fuzzy variables and values, implementing more fuzzy rules and using the S-shape membership function for the term frequency ratio and document frequency. It was also proven that the common definition of hedges doesn't apply to the IR situation so new interpretations of hedges that are suitable for IR were introduced.

In conclusion, the IR-FIS system proved to be a successful enhancement of Ruben's R-FIS system. It outperformed Lucene at document cut off P10 by 4.7% which is ***five times better*** improvement than the one reached by Rubens R-FIS [19].

5.2 Future scope:

More experiments were needed but due to time limitation it was decided to leave them as future enhancements for the current model. These are experiments to explore the effect of more contextual parameters like the position of the query term in the document and whether it is in the first line of the paragraph or in the introductory paragraph. Also the distance between query terms in the document was not investigated; so the idea of having more than one query term occurring in the same sentence in the document should have increased the relevance of the document to the query compared to the relevance of another document that has the query terms but far apart. The synonyms of the query terms can also be considered so that relevant documents are retrieved even if they don't have the exact query terms but they have their synonyms. More inference rules and variables need to be added to reflect the context of the terms. Machine learning can also be utilized to ***automate*** two areas in IR-FIS: 1) Finding the best parameters for the membership functions for each of the fuzzy values. For example, predicting the frequency intervals for each fuzzy value based on the term frequency distribution in the corpus; the interval for the low frequencies, the medium and the high ones. 2) Finding the best rules instead of explicitly specifying them in the rule base. Finally, the whole system should also be tested using other languages as Arabic so we can see how the performance is affected.

References

[1] Alexander Dekhtyar. *Probabilistic Information Retrieval Part II*: *In Depth.* Department of Computer Science. University of Maryland

[2] Anselm Spoerri. InfoCrystal : *A Visual Tool For Information Retrieval* .Submitted to the Department of Civil and Environmental Engineering on January 20, 1995

[3] Art B. Owen. *Information Retrieval and the Vector Space Model*. Stanford University owen@stat.stanford.edu

[4] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *An Introduction to Information Retrieval*, Cambridge University Press. 2008.

[5] Donna Harman. *RANKING ALGORITHMS*. National Institute of Standards & Technology

[6] E. Garcia. *The Classic Vector Space Model Description, Advantages and Limitations of the Classic Vector Space Model*. MiIslita.com. Last Update: 10/27/06

[7] E. M. Voorhees and D. "*Overview of the Seventh Text Retrieval Conference (TREC-7)"* .Harman, in NIST Special Publication 500-242: The Seventh Text Retrieval Conference, 98.

[8] Eibe Frank, Ashraf M. Kibriya.Stop words code. Machine Learning Group at University of Waikato.

[9] *"Fuzzy_logic",* Wikipedia, the Free Encyclopedia. Retrieved 17 June 2011 from <http://en.wikipedia.org/wiki/Fuzzy_logic>

[10] *"Fuzzy retrieval",* Wikipedia, the Free Encyclopedia. Retrieved 17 June 2011 from <http://en.wikipedia.org/wiki/Fuzzy_retrieval>

[11] Information Retrieval Lecture 5 - *The vector space model*. Seminar f¨ur Sprachwissenschaft. International Studies in Computational Linguistics. semester 2007 1

[12] James F.Brule. Fuzzy systems –A tutorial. 1985, < http://www.austinlinks.com/Fuzzy/tutorial.html>

[13] Jan Jantzen. *Tutorial on fuzzy logic*. Technical university of Denmark, Department of Automation, DENMARK. Aug 1998.

[14] K.Nowacka, S.Zadrozny, J.Kacprzyk. *A new fuzzy logic based information retrieval model*. Proceedings of IPMU'08, pp.1749-1756.June 22-27, 2008

[15] Khalaf Khatatneh, Iman Hussain. *Information Retrievals tries tree V.s Inverted File Word Method for Arabic language*. Prince Abdu Allah Bin Ghazi for IT, Al-Balqa Applied University Salt, Jordan

[16] *"Lucene"*. Wikipedia, the Free Encyclopedia. Retrieved 29 April 2011 from <http://en.wikipedia.org/wiki/Apache_Lucene Citation>

[17] Martin Porter. *The Porter Stemming Algorithm*. Last revised January 2006. <http://tartarus.org/~martin/PorterStemmer/>

[18] MathWorks.S-shape function. 3 Apple Hill Drive Natick, United States http://www.mathworks.com/help/toolbox/fuzzy/smf.html.

[19] N.O.Rubens. *The Application of fuzzy logic to the construction of the ranking function of information retrieval systems*. Computer Modeling and New Technologies, 2006.

[20] Norbert Fuhr. *Probabilistic Models in Information Retrieval*. The computer journal 1992.

[21] *Reasoning with uncertainty-Fuzzy Reasoning* , Lesson 32 :Version 2 CSE IIT, Kharagpur, http://classle.s3.amazonaws.com/sites/default/files/25053/Lesson_32.pdf

[22] Romaric Besançon, Stéphane Chaudiron, Djamel Mostefa, Ismaïl Timimi, Khalid Choukri, Meriama Laïb. *Overview of CLEF 2009 INFILE track*. Université de Lille 3 – GERiiCO

[23] Salha Mohammed Alzahrani, and Naomie Salim. *On the Use of Fuzzy Information Retrieval for Gauging similarity of Arabic Documents*

[24] Satyendra Nath Mandal, J.Pal Choudhury, Dilip De, and S. R. Bhadra Chaudhuri. *Role of membership functions in fuzzy logic for prediction of shoot length of mustard plant based on residual analysis*. World academy of science, engineering and technology 2008.

[25] Snowball. *Porter stemmer algorithm*. http://snowball.tartarus.org/algorithms/porter/stemmer.html

[26] *"Standard Boolean model"*. Wikipedia, the Free Encyclopedia. Retrieved 29 April 2011 from <http://en.wikipedia.org/wiki/Standard_Boolean_model>

[27] Stefan Pohl, Justin Zobel, Alistair Moffat. *Extended Boolean Retrieval for Systematic Biomedical Reviews*. NICTA Victoria Research Laboratory, Department of Computer Science and Software Engineering. The University of Melbourne, Victoria 3010, Australia

[28] Todd A. Letsche and Michael W. Berry. *Large-Scale Information Retrieval with Latent Semantic Indexing*. University of Tennessee. Preprint, 1996.

[29] V.Balamurugan1 and K.Senthamarai Kannan. *A Framework for Computing Linguistic Hedges in Fuzzy Queries*. Department of Computer Science & Engineering, SCAD College of Engineering &Technology, Cheranmahadevi, Tirunelveli, India

[30] V.N.Huynh, T.B.Ho,Y. *A parametric representation of linguistic hedges in Zadeh's fuzzy logic*. Nakamori. International journal of Approximate Reasoning 30 (2002) 203-223

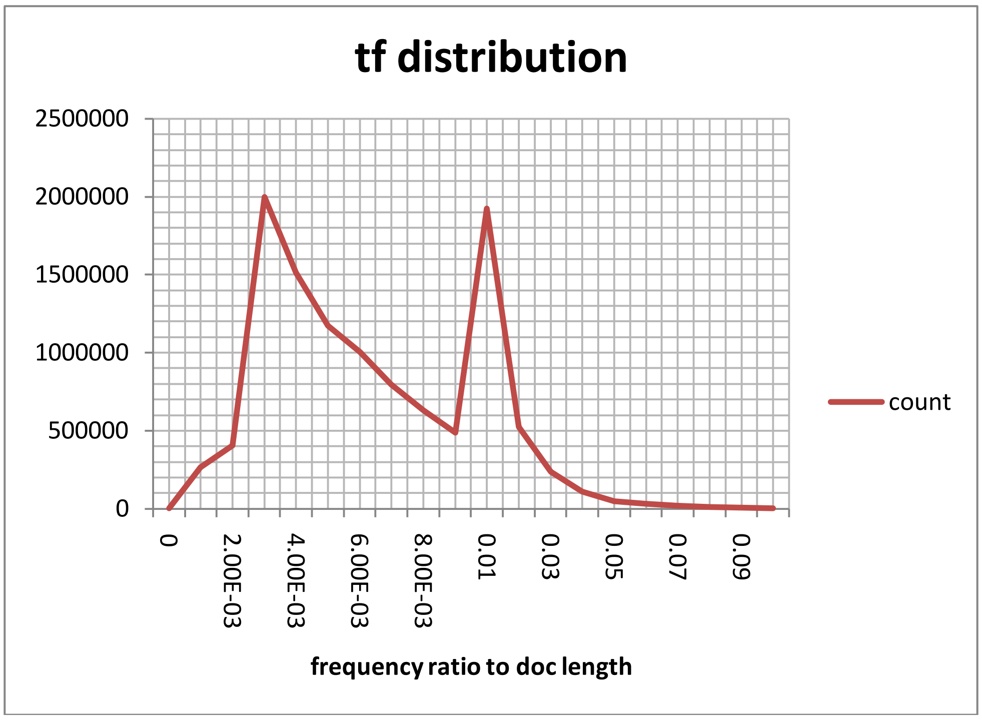
[31] *"Vector space model"*. Wikipedia, the Free Encyclopedia. Retrieved 29 April 2011 from <http://en.wikipedia.org/wiki/Vector_space_model>

APPENDIX A:

STATISTICAL DISTRIBUTIONS AND EXPREMENTS RESULTS

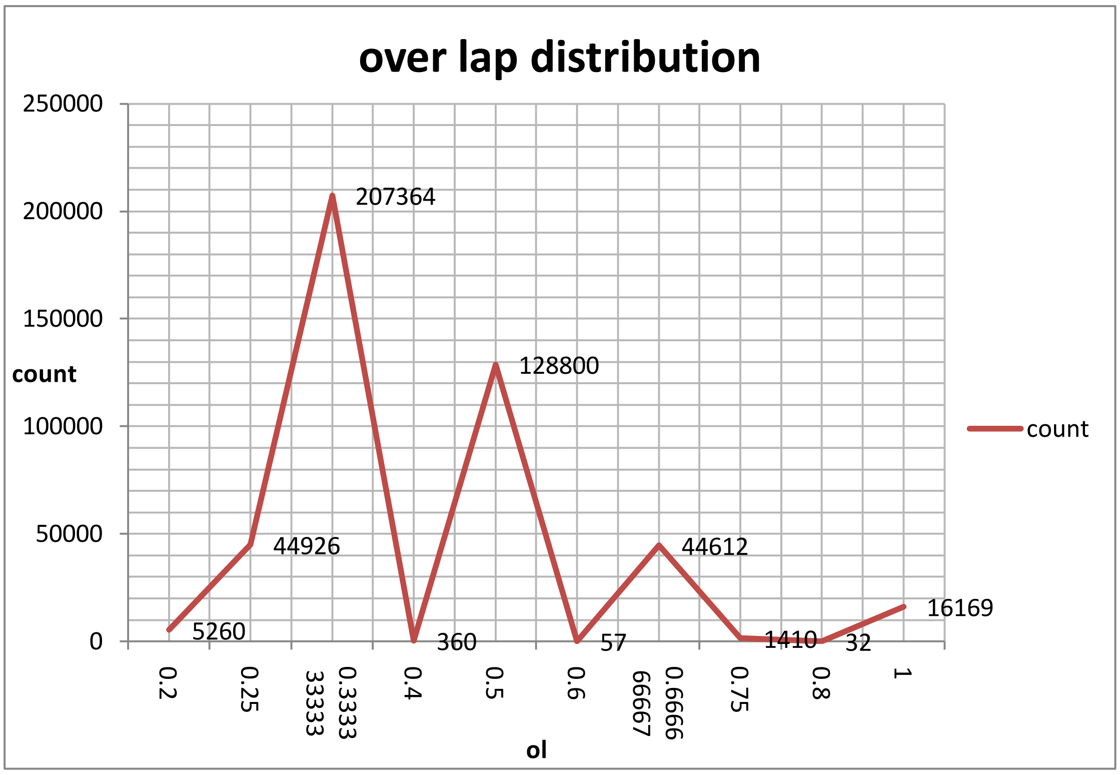
A.1 TERM FREQUENCY DISTRIBUTION:PROVIDE SOME TEXT TO EXPLAIN THETABLES AND FIGURES

tfratio	count
0	4409
1.00E-03	265722
2.00E-03	406452
3.00E-03	1998669
4.00E-03	1512649
5.00E-03	1173642
6.00E-03	1002578
7.00E-03	795967
8.00E-03	632978
9.00E-03	486384
0.01	1925156
0.02	524905
0.03	237574
0.04	111000
0.05	49428
0.06	29469
0.07	18338
0.08	11418
0.09	5834
0.1	4400



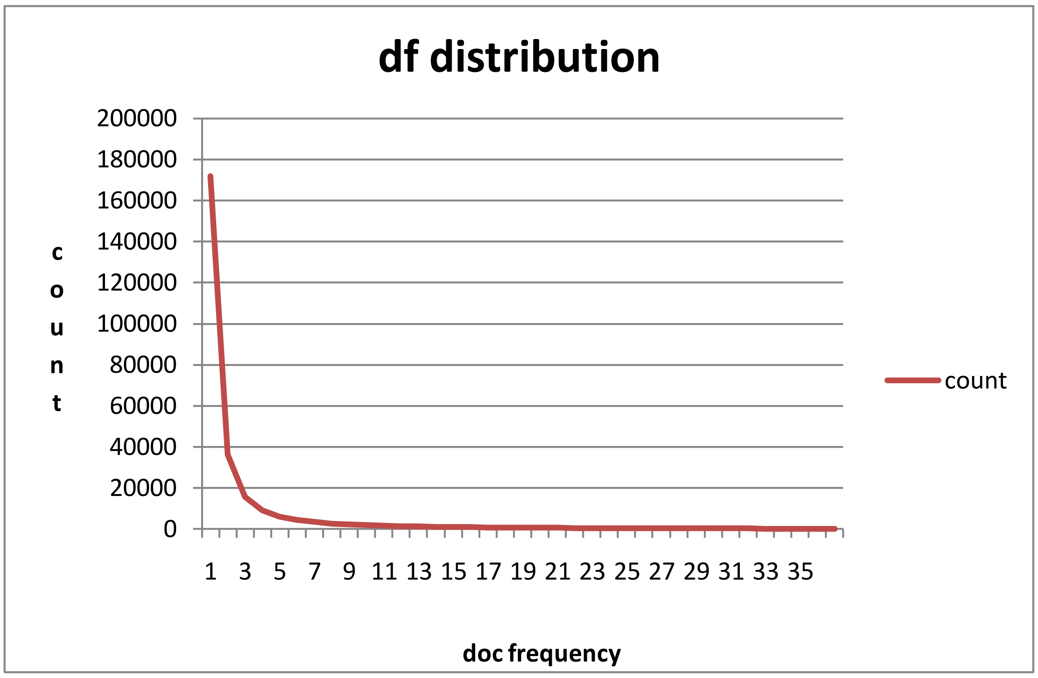
A.2 **OVER LAP DISTRIBUTION:**

frequency	count
0.2	5260
0.25	44926
0.333333333	207364
0.4	360
0.5	128800
0.6	57
0.666666667	44612
0.75	1410
0.8	32
1	16169



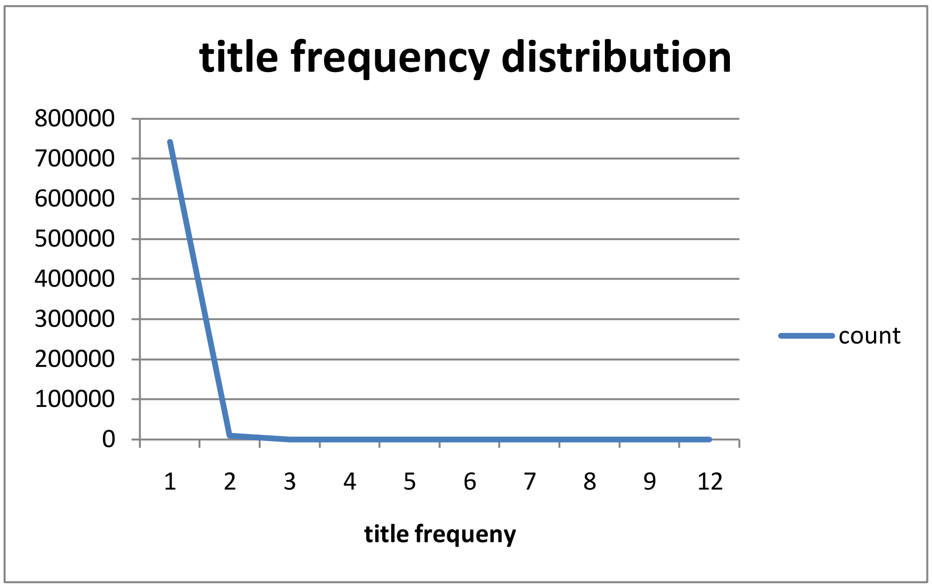
A.3 **DOCUMENT FREQUENCY DISTRIBUTION:**

df	count
1	171739
2	36397
3	15738
4	9041
5	6134
6	4508
7	3477
8	2701
9	2170
10	1935
11	1589
12	1429
13	1235
14	1098
15	984
16	930
17	813
18	714
19	712
20	672
21	608
22	541
23	478
24	502
25	446
26	422
27	409
28	367
29	330
30	338
31	331
32	315
33	263
34	275
35	278
36	247
.	.
.	.
.	.
99299	1



A.4 **TITLE FREQUENCY DISTRIBUTION:**

titlef	count
1	742090
2	9104
3	268
4	100
5	11
6	2
7	6
8	3
9	1
12	1



A.5 **QUERIES WITH HEDGES: Slightly** means the term is not that important for relevance **Indifferent** means the term has neutral and the hedge is only a place holder & **Indeed** means the term is very important for the relevance**.**

query number	**Query with hedges**
101	slightly fight indifferent dope slightly sport
102	indifferent sport Indeed economi
103	indifferent organ slightly intern indifferent sport indifferent competit
104	slightly stake slightly popular indifferent sport
105	indifferent sport Indeed violenc Indeed racism
106	Indeed disabl slightly adapt indifferent sport
107	indifferent electron indifferent vote
108	Indeed quinquennat slightly polit slightly institut
109	indifferent politician indifferent media
111	indifferent poll indifferent polici
112	indifferent slam indifferent poetri slightly democrat
113	indifferent digit indifferent divid
114	indifferent arthous indifferent cinema
115	slightly free indifferent museum
116	slightly free indifferent newspap indifferent franc
117	Indeed cartoon
118	indifferent rise Indeed oil indifferent price
119	Indeed subprim indifferent crisi
120	indifferent sale indifferent franc
121	Indeed supermarket
122	slightly fate indifferent local indifferent shop
123	slightly challeng indifferent ecommerc
124	slightly fight Indeed aid indifferent commun
125	slightly issu Indeed tibet
126	indifferent european indifferent immigr slightly polici
127	slightly crisi indifferent darfur
128	indifferent israelipalestinian slightly conflict
129	Indeed farc indifferent rebelion
130	indifferent olympic indifferent game Indeed organ
131	indifferent egovern slightly stake
132	slightly wireless indifferent network indifferent health
133	slightly competit indifferent soccer indifferent broadcat slightly right
135	slightly issu Indeed kyoto indifferent protocol
136	slightly air indifferent pollut slightly air indifferent qual
137	slightly fight indifferent climat slightly chang
138	indifferent drug indifferent biotechnologi
139	Indeed biofuel
140	indifferent fruit indifferent veget slightly intak Indeed cancer slightly prevent
141	indifferent dope slightly athletess indifferent health
142	Indeed biodivers indifferent preserv
143	Indeed avian indifferent influenza
144	indifferent nanotechnologi indifferent nanosci
145	indifferent genet slightly modifi indifferent plant
146	indifferent prostat indifferent cancer
148	indifferent renew indifferent energi
149	slightly scientif indifferent research Indeed arctic
150	indifferent woman slightly world indifferent work

A.6 RESULTS OF USING THE TITLE PARAMETER ALONE WITH & WITHOUT HEDGES

run		Lucene	title	Title with hedges
number of queries		46	47	47
Retrieved:		41393	40692	40692
Relevant:		1547	1597	1597
Rel_ret:		1340	1098	1098
Average precision		0.3498	0.1926	0.1926
R-Precision		0.3471	0.2121	0.2121
Interpolated	Recall			
run		**Lucene**	**title**	**Title with hedges**
	0	0.6147	0.4724	0.4724
	0.1	0.5478	0.3835	0.3835
	0.2	0.4867	0.3065	0.3065
	0.3	0.431	0.2675	0.2675
	0.4	0.3905	0.2516	0.2516
	0.5	0.3577	0.2038	0.2038
	0.6	0.3265	0.1533	0.1533
	0.7	0.2728	0.1173	0.1173
	0.8	0.2435	0.1062	0.1062
	0.9	0.1989	0.0619	0.0619
	1	0.1103	0.0241	0.0241
Precision at document cutoff:				
run		**Lucene**	**title**	**Title with hedges**
docs:	5	0.4087	0.2723	0.2723
docs:	10	0.4043	0.2532	0.2532
docs:	15	0.3812	0.2426	0.2426
docs:	20	0.3685	0.2362	0.2362
docs:	30	0.3391	0.2206	0.2206
docs:	100	0.1896	0.1398	0.1398
docs:	200	0.1175	0.0887	0.0887
docs:	500	0.0556	0.0437	0.0437
docs:	1000	0.0291	0.0234	0.0234

A.7 RESULTS OF USING THE OVERLAP FUNCTIONS WITH & WITHOUT HEDGES

run		Lucene	Overlap S-shape	Overlap best	Overlap triangle	Overlap high + hedges
number of queries		46	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597
Rel_ret:		1340	1099	1100	1099	1100
Average precision		0.3498	0.1946	0.1946	0.1945	0.1946
R-Precision		0.3471	0.2173	0.2173	0.2173	0.2173
Interpolated	Recall					
run		Lucene	Overlap S-shape	Overlap best	Overlap triangle	Overlap high + hedges
	0	0.6147	0.4777	0.4777	0.4777	0.4777
	0.1	0.5478	0.392	0.392	0.392	0.392
	0.2	0.4867	0.31	0.31	0.31	0.31
	0.3	0.431	0.268	0.268	0.268	0.268
	0.4	0.3905	0.2523	0.2523	0.2523	0.2523
	0.5	0.3577	0.2045	0.2045	0.2045	0.2045
	0.6	0.3265	0.1522	0.1522	0.1522	0.1522
	0.7	0.2728	0.1163	0.1163	0.1163	0.1163
	0.8	0.2435	0.1048	0.1048	0.1048	0.1048
	0.9	0.1989	0.0608	0.0608	0.0605	0.0608
	1	0.1103	0.0226	0.0226	0.0226	0.0226
Precision at document cutoff:						
run		Lucene	Overlap S-shape	Overlap best	Overlap triangle	Overlap high + hedges
docs:	5	0.4087	0.2766	0.2766	0.2766	0.2766
docs:	10	0.4043	0.2638	0.2638	0.2638	0.2638
docs:	15	0.3812	0.2468	0.2468	0.2468	0.2468
docs:	20	0.3685	0.2394	0.2394	0.2394	0.2394
docs:	30	0.3391	0.2227	0.2227	0.2227	0.2227
docs:	100	0.1896	0.1396	0.1396	0.1396	0.1396
docs:	200	0.1175	0.0893	0.0893	0.0891	0.0893
docs:	500	0.0556	0.0437	0.0437	0.0437	0.0437
docs:	1000	0.0291	0.0234	0.0234	0.0234	0.0234

A.8 RESULTS OF USING DF-RATIO FUNCTIONS WITH & WITHOUT HEDGES

run		**Lucene**	**df-ratio triangle**	**df-ratio S-shape**	**df-ratio triangle with hedge**	**df-ratio S-shape with hedges**
number of queries		46	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597
Rel_ret:		1340	1103	1106	1103	1106
Average precision		0.3498	0.1954	0.202	0.1954	0.202
R-Precision		0.3471	0.2238	0.2299	0.2238	0.2299
Interpolated	Recall					
run		**Lucene**	**df-ratio triangle**	**df-ratio S-shape**	**df-ratio triangle with hedge**	**df-ratio S-shape with hedges**
	0	0.6147	0.4687	0.4899	0.4687	0.4899
	0.1	0.5478	0.3972	0.4227	0.3972	0.4227
	0.2	0.4867	0.318	0.3478	0.318	0.3478
	0.3	0.431	0.2758	0.2758	0.2758	0.2758
	0.4	0.3905	0.2519	0.2519	0.2519	0.2519
	0.5	0.3577	0.2044	0.2044	0.2044	0.2044
	0.6	0.3265	0.1521	0.1521	0.1521	0.1521
	0.7	0.2728	0.1162	0.1162	0.1162	0.1162
	0.8	0.2435	0.1047	0.1047	0.1047	0.1047
	0.9	0.1989	0.0604	0.0604	0.0604	0.0604
	1	0.1103	0.0226	0.0226	0.0226	0.0226
Precision at document cutoff:						
run		**Lucene**	**df-ratio triangle**	**df-ratio S-shape**	**df-ratio triangle with hedge**	**df-ratio S-shape with hedges**
docs:	5	0.4087	0.2766	0.2894	0.2766	0.2894
docs:	10	0.4043	0.2638	0.2745	0.2638	0.2745
docs:	15	0.3812	0.2496	0.2567	0.2496	0.2567
docs:	20	0.3685	0.2415	0.2457	0.2415	0.2457
docs:	30	0.3391	0.2255	0.2277	0.2255	0.2277
docs:	100	0.1896	0.1404	0.1411	0.1404	0.1411
docs:	200	0.1175	0.0896	0.0899	0.0896	0.0899
docs:	500	0.0556	0.0439	0.044	0.0439	0.044
docs:	1000	0.0291	0.0235	0.0235	0.0235	0.0235

A.9 RESULTS OF USING TF-RATIO FUNCTIONS WITH & WITHOUT HEDGES

run		Lucene	**tf-ratio S-shape**	**tf-ratio triangle**	**tf-ratio triangle + hedges**	**tf-ratio S-shape + hedges**
number of queries		46	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597
Rel_ret:		1340	1255	1247	1247	1255
Average precision		0.3498	0.294	0.2962	0.296	0.2941
R-Precision		0.3471	0.308	0.3076	0.3076	0.3084
Interpolated	Recall					
run		Lucene	**tf-ratio S-shape**	**tf-ratio triangle**	**tf-ratio triangle + hedges**	**tf-ratio S-shape + hedges**
	0	0.6147	0.5719	0.5773	0.5774	0.5717
	0.1	0.5478	0.5089	0.4801	0.4797	0.5089
	0.2	0.4867	0.439	0.4296	0.4273	0.439
	0.3	0.431	0.4048	0.4073	0.4081	0.4044
	0.4	0.3905	0.3756	0.3819	0.3821	0.3759
	0.5	0.3577	0.3263	0.3297	0.3299	0.326
	0.6	0.3265	0.2414	0.2506	0.2508	0.2416
	0.7	0.2728	0.2208	0.2316	0.232	0.2211
	0.8	0.2435	0.1817	0.1885	0.1883	0.1823
	0.9	0.1989	0.1024	0.1063	0.1065	0.1026
	1	0.1103	0.0397	0.0411	0.0411	0.0399
Precision at document cutoff:						
run		Lucene	**tf-ratio S-shape**	**tf-ratio triangle**	**tf-ratio triangle + hedges**	**tf-ratio S-shape + hedges**
docs:	5	0.4087	0.4043	0.3915	0.3915	0.4043
docs:	10	0.4043	0.3915	0.3915	0.3894	0.3915
docs:	15	0.3812	0.3589	0.356	0.356	0.3589
docs:	20	0.3685	0.3436	0.3426	0.3404	0.3426
docs:	30	0.3391	0.3106	0.3099	0.3099	0.3113
docs:	100	0.1896	0.1689	0.1709	0.1709	0.1691
docs:	200	0.1175	0.1032	0.1016	0.1017	0.1032
docs:	500	0.0556	0.0498	0.0497	0.0497	0.0498
docs:	1000	0.0291	0.0267	0.0265	0.0265	0.0267

A.10 RESULTS OF USING COMBINATIONS OF FOUR VARIABLES WITH HEDGES

							best one
run		**Lucene**	**tf-S-shape df-S-shape +hedges function 3.6**	**tf-S-shape df-S-shape +hedges[4.1]**	**tf-triangle df-triangle**	**tf-triangle df-S-shape**	**tf-S-shape df-S-shape + hedges function 3.7**
number of queries		46	47	47	47	47	47
Retrieved:		41393	40692	40692	40692	40692	40692
Relevant:		1547	1597	1597	1597	1597	1597
Rel_ret:		1340	1252	1306	1307	1305	1307
Average precision		0.3498	0.2952	0.3167	0.318	0.318	0.3289
R-Precision		0.3471	0.3145	0.3432	0.3448	0.3448	0.3501
Interpolated	Recall						
run		**Lucene**	**tf-S-shape df-S-shape +hedge[3.6]**	**tf-S-shape df-S-shape +hedges[4.1]**	**tf-triangle df-triangle**	**tf-triangle df-S-shape**	**tf-S-shape df-S-shape + hedges[3.7]**
	0	0.6147	0.5857	0.6125	0.5975	0.5973	0.6177
	0.1	0.5478	0.5323	0.5604	0.5377	0.5375	0.5681
	0.2	0.4867	0.4422	0.4674	0.4674	0.4672	0.4752
	0.3	0.431	0.4061	0.4308	0.4283	0.4281	0.4399
	0.4	0.3905	0.3615	0.3894	0.3894	0.3894	0.3972
	0.5	0.3577	0.315	0.3558	0.3553	0.3553	0.3705
	0.6	0.3265	0.254	0.2674	0.2794	0.2798	0.283
	0.7	0.2728	0.2115	0.2341	0.2398	0.2398	0.252
	0.8	0.2435	0.1643	0.1825	0.1892	0.1892	0.1986
	0.9	0.1989	0.1002	0.1055	0.1134	0.1134	0.1217
	1	0.1103	0.0383	0.0411	0.0418	0.0418	0.0463
Precision at document cutoff:							
run		**Lucene**	**tf-S-shape df-S-shape +hedge[3.6]**	**tf-S-shape df-S-shape +hedges[4.1]**	**tf-triangle df-triangle**	**tf-triangle df-S-shape**	**tf-S-shape df-S-shape + hedges[3.7]**
docs:	5	0.4087	0.4128	0.4128	0.417	0.417	0.4255
docs:	10	0.4043	0.4106	0.4191	0.4043	0.4043	0.4234
docs:	15	0.3812	0.3787	0.3787	0.3702	0.3702	0.3801
docs:	20	0.3685	0.3479	0.3532	0.3574	0.3574	0.3564
docs:	30	0.3391	0.3177	0.3206	0.3227	0.3227	0.3248
docs:	100	0.1896	0.17	0.1794	0.18	0.1802	0.1838
docs:	200	0.1175	0.1033	0.1078	0.1082	0.1082	0.1112
docs:	500	0.0556	0.0486	0.0509	0.0514	0.0512	0.0512
docs:	1000	0.0291	0.0266	0.0278	0.0278	0.0278	0.0278

APPENDIX B:

SOURCE CODE OF SYSTEM MODULES

B.1 STOP WORDS BASED ON [8]:

```
package javaApplication15;
import java.io.*;
import java.util.*;

/**
 * Class that can test whether a given string is a stop word.
 * Lowercases all words before the test. <p/>
 * The format for reading and writing is one word per line, lines starting
 * with '#' are interpreted as comments and therefore skipped. <p/>
 * The default stop words are based on <a href="http://www.cs.cmu.edu/~mccallum/bow/rainbow/"
target="_blank">Rainbow</a>. <p/>
*/
public class Stopwords {

  /** The hash variable containing the list of stopwords */
  protected HashSet m_Words = null;

  /** The default stop words object (stoplist based on Rainbow) */
  protected static Stopwords m_Stopwords;

  static {
    if (m_Stopwords == null) {
      m_Stopwords = new Stopwords();
    }
  }

  /**
   * initializes the stopwords (based on <a href="http://www.cs.cmu.edu/~mccallum/bow/rainbow/"
target="_blank">Rainbow</a>).   */

 public Stopwords() {
    m_Words = new HashSet();

    //Stopwords list from Rainbow
<use more than one columin>
    add("a");
    add("able");
    add("about");
    add("above");
    add("according");
    add("accordingly");
    add("across");
    add("actually");
    add("after");
    add("afterwards");
    add("again");
    add("against");
    add("all");
    add("allow");
    add("allows");
    add("almost");
    add("alone");
    add("along");
    add("already");
    add("also");
    add("although");
    add("always");
    add("am");
    add("among");
    add("amongst");
    add("an");
add("and");
add("another");
add("any");
add("anybody");
add("anyhow");
add("anyone");
add("anything");
add("anyway");
add("anyways");
add("anywhere");
add("apart");
add("appear");
add("appreciate");
add("appropriate");
add("are");
add("around");
add("as");
add("aside");
add("ask");
add("asking");
add("associated");
add("at");
add("available");
add("away");
add("awfully");
add("b");
add("be");
add("became");
add("because");
add("become");
add("becomes");
add("becoming");
add("been");
add("before");
add("beforehand");
add("behind");
add("being");
add("believe");
add("below");
add("beside");
add("besides");
add("best");
add("better");
add("between");
add("beyond");
add("both");
add("brief");
add("but");
add("by");
add("c");
add("came");
add("can");
add("cannot");
add("cant");
add("cause");
add("causes");
add("certain");
add("certainly");
add("changes");
add("clearly");
add("co");
add("com");
add("come");
add("comes");
add("concerning");
add("consequently");
add("consider");
add("considering");
add("contain");
add("containing");
add("contains");
add("corresponding");
add("could");
add("course");
add("currently");
add("d");
add("definitely");
add("described");
add("despite");
add("did");
add("different");
add("do");
add("does");
add("doing");
add("done");
add("down");
add("downwards");
add("during");
add("e");
add("each");
add("edu");
add("eg");
add("eight");
add("either");
add("else");
add("elsewhere");
add("enough");
add("entirely");
add("especially");
add("et");
add("etc");
add("even");
add("ever");
add("every");
add("everybody");
add("everyone");
add("everything");
add("everywhere");
add("ex");
add("exactly");
add("example");
add("except");
add("f");
add("far");
add("few");
add("fifth");
add("first");
add("five");
add("followed");
add("following");
add("follows");
add("for");
add("former");
add("formerly");
add("forth");
add("four");
add("from");
add("further");
add("furthermore");
add("g");
add("get");
add("gets");
add("getting");
add("given");
add("gives");
add("go");
add("goes");
add("going");
add("gone");
add("got");
add("gotten");
add("greetings");
add("h");
add("had");
add("happens");
add("hardly");
add("has");
add("have");
add("having");
add("he");
add("hello");
add("help");
add("hence");
add("her");
add("here");
add("hereafter");
add("hereby");
add("herein");
add("hereupon");
add("hers");
add("herself");
add("hi");
add("him");
add("himself");
add("his");
add("hither");
add("hopefully");
add("how");
add("howbeit");
add("however");
add("i");
add("ie");
add("if");
add("ignored");
add("immediate");
add("in");
add("inasmuch");
add("inc");
add("indeed");
add("indicate");
add("indicated");
add("indicates");
add("inner");
add("insofar");
add("instead");
add("into");
add("inward");
add("is");
add("it");
add("its");
add("itself");
add("j");
add("just");
add("k");
add("keep");
add("keeps");
add("kept");
add("know");
add("knows");
add("known");
add("l");
add("last");
add("lately");
add("later");
add("latter");
add("latterly");
add("least");
add("less");
add("lest");
add("let");
add("like");
add("liked");
add("likely");
add("little");
add("ll"); //added to avoid words like you'll,I'll etc.
add("look");
add("looking");
add("looks");
add("ltd");
add("m");
add("mainly");
add("many");
add("may");
add("maybe");
add("me");
add("mean");
add("meanwhile");
add("merely");
add("might");
add("more");
add("moreover");
add("most");
add("mostly");
add("much");
add("must");
add("my");
add("myself");
add("n");
add("name");
add("namely");
add("nd");
add("near");
add("nearly");
add("necessary");
add("need");
add("needs");
add("neither");
add("never");
add("nevertheless");
add("new");
add("next");
add("nine");
add("no");
add("nobody");
add("non");
add("none");
add("noone");
add("nor");
add("normally");
add("not");
add("nothing");
add("novel");
add("now");
add("nowhere");
add("o");
add("obviously");
add("of");
add("off");
add("often");
add("oh");
add("ok");
add("okay");
add("old");
add("on");
add("once");
add("one");
add("ones");
add("only");
add("onto");
add("or");
add("other");
add("others");
add("otherwise");
add("ought");
add("our");
add("ours");
add("ourselves");
add("out");
add("outside");
add("over");
add("overall");
add("own");
add("p");
add("particular");
add("particularly");
add("per");
add("perhaps");
add("placed");
add("please");
add("plus");
add("possible");
add("presumably");
add("probably");
add("provides");
add("q");
add("que");
add("quite");
add("qv");
add("r");
add("rather");
add("rd");
add("re");
add("really");
add("reasonably");
add("regarding");
add("regardless");
add("regards");
add("relatively");
add("respectively");
add("right");
add("s");
add("said");
add("same");
add("saw");
add("say");
add("saying");
add("says");
add("second");
add("secondly");
add("see");
add("seeing");
add("seem");
add("seemed");
add("seeming");
add("seems");
add("seen");
add("self");
add("selves");
add("sensible");
add("sent");
add("serious");
add("seriously");
add("seven");
add("several");
add("shall");
add("she");
add("should");
add("since");
add("six");
add("so");
add("some");
add("somebody");
add("somehow");
add("someone");
add("something");
add("sometime");
add("sometimes");
add("somewhat");
add("somewhere");
add("soon");
add("sorry");
add("specified");
add("specify");
add("specifying");
add("still");
add("sub");
add("such");
add("sup");
add("sure");
add("t");
add("take");
add("taken");
add("tell");
add("tends");
add("th");
add("than");
add("thank");
add("thanks");
add("thanx");
add("that");
add("thats");
add("the");
add("their");
add("theirs");
add("them");
add("themselves");
add("then");
add("thence");
add("there");
add("thereafter");
add("thereby");
add("therefore");
add("therein");
add("theres");
add("thereupon");
add("these");
add("they");
add("think");
add("third");
add("this");
add("thorough");
add("thoroughly");
add("those");
add("though");
add("three");
add("through");
add("throughout");
add("thru");
add("thus");
add("to");
add("together");
add("too");
add("took");
add("toward");
add("towards");
add("tried");
add("tries");
add("truly");
add("try");
add("trying");
add("twice");
add("two");
add("u");
add("un");
add("under");
add("unfortunately");
add("unless");
add("unlikely");
add("until");
add("unto");
add("up");
add("upon");
add("us");
add("use");
add("used");
add("useful");
add("uses");
add("using");
add("usually");
add("uucp");
add("v");
add("value");
add("various");
add("ve"); //added to avoid words like I've,you've etc.
add("very");
add("via");
add("viz");
add("vs");
add("w");
add("want");
add("wants");
add("was");
add("way");
add("we");
add("welcome");
add("well");
add("went");
add("were");
add("what");
add("whatever");
add("when");
add("whence");
add("whenever");
add("where");
add("whereafter");
add("whereas");
add("whereby");
add("wherein");
add("whereupon");
```

```
  add("wherever");
  add("whether");
  add("which");
  add("while");
  add("whither");
  add("who");
  add("whoever");
  add("whole");
  add("whom");
  add("whose");
  add("why");
  add("will");
  add("willing");
  add("wish");
  add("with");
  add("within");
  add("without");
  add("wonder");
  add("would");
  add("would");
  add("x");
  add("y");
  add("yes");
  add("yet");
  add("you");
  add("your");
  add("yours");
  add("yourself");
  add("yourselves");
  add("z");
  add("zero");
 }

 /**
  * removes all stopwords
  */
 public void clear() {
  m_Words.clear();
 }

 /**
  * adds the given word to the stopword list (is automatically converted to
  * lower case and trimmed)
  *
  * @param word the word to add
  */
 public void add(String word) {
  if (word.trim().length() > 0)
    m_Words.add(word.trim().toLowerCase());
 }

 /**
  * removes the word from the stopword list
  *
  * @param word the word to remove
  * @return true if the word was found in the list and then removed
  */
 public boolean remove(String word) {
  return m_Words.remove(word);
 }

 /**
  * Returns true if the given string is a stop word.
  *
  * @param word the word to test
  * @return true if the word is a stopword
  */
```

```
public boolean is(String word) {
  return m_Words.contains(word.toLowerCase());
}

/**
 * Returns a sorted enumeration over all stored stopwords
 *
 * @return the enumeration over all stopwords
 */
public Enumeration elements() {
  Iterator    iter;
  Vector      list;

  iter = m_Words.iterator();
  list = new Vector();

  while (iter.hasNext())
    list.add(iter.next());

  // sort list
  Collections.sort(list);

  return list.elements();
}

/**
 * Generates a new Stopwords object from the given file
 *
 * @param filename the file to read the stopwords from
 * @throws Exception if reading fails
 */
public void read(String filename) throws Exception {
  read(new File(filename));
}

/**
 * Generates a new Stopwords object from the given file
 *
 * @param file the file to read the stopwords from
 * @throws Exception if reading fails
 */
public void read(File file) throws Exception {
  read(new BufferedReader(new FileReader(file)));
}

/**
 * Generates a new Stopwords object from the reader. The reader is
 * closed automatically.
 *
 * @param reader the reader to get the stopwords from
 * @throws Exception if reading fails
 */
public void read(BufferedReader reader) throws Exception {
  String      line;

  clear();

  while ((line = reader.readLine()) != null) {
    line = line.trim();
    // comment?
    if (line.startsWith("#"))
      continue;
    add(line);
  }

  reader.close();
}
```

```
/**
 * Writes the current stopwords to the given file
 *
 * @param filename the file to write the stopwords to
 * @throws Exception if writing fails
 */
public void write(String filename) throws Exception {
  write(new File(filename));
}

/**
 * Writes the current stopwords to the given file
 *
 * @param file the file to write the stopwords to
 * @throws Exception if writing fails
 */
public void write(File file) throws Exception {
  write(new BufferedWriter(new FileWriter(file)));
}

/**
 * Writes the current stopwords to the given writer. The writer is closed
 * automatically.
 *
 * @param writer the writer to get the stopwords from
 * @throws Exception if writing fails
 */
public void write(BufferedWriter writer) throws Exception {
  Enumeration   enm;

  // header
  writer.write("# generated " + new Date());
  writer.newLine();

  enm = elements();

  while (enm.hasMoreElements()) {
    writer.write(enm.nextElement().toString());
    writer.newLine();
  }

  writer.flush();
  writer.close();
}

/**
 * returns the current stopwords in a string
 *
 * @return the current stopwords
 */
public String toString() {
  Enumeration   enm;
  StringBuffer  result;

  result = new StringBuffer();
  enm    = elements();
  while (enm.hasMoreElements()) {
    result.append(enm.nextElement().toString());
    if (enm.hasMoreElements())
      result.append(",");
  }

  return result.toString();
}

/**
```

```
   * Returns true if the given string is a stop word.
   *
   * @param str the word to test
   * @return true if the word is a stopword
   */
  public static boolean isStopword(String str) {
    return m_Stopwords.is(str.toLowerCase());
  }

  /**
   * Accepts the following parameter: <p/>
   *
   * -i file <br/>
   * loads the stopwords from the given file <p/>
   *
   * -o file <br/>
   * saves the stopwords to the given file <p/>
   *
   * -p <br/>
   * outputs the current stopwords on stdout <p/>
   *
   * Any additional parameters are interpreted as words to test as stopwords.
   *
   * @param args commandline parameters
   * @throws Exception if something goes wrong
   */
  public static void main(String[] args) {
   File directory = new File("C:/parsed");
   //File directory = new File("C:/docs/DOCS/parsed");
    // File directory = new File("C:/querydoc");
     File files[] = directory.listFiles();
     String filename[] = directory.list();
     int count = 0;
     String titlestr;
     boolean title;
       for (File f : files)
       {
String nf=f.getName();
           BufferedReader reader = null;
           String titleline="";
            Vector words = new Vector();
          //  titlestr = "";
            try
{
reader = new BufferedReader(new FileReader(f));
String text = null;
// repeat until all lines is read

title=true;
while ((text = reader.readLine()) != null)
{
       String str = text;
        String strAr[] = str.split(" ");
String wrd="";
        for(int i=0;i<strAr.length;i++)
                {
                String word = strAr[i];
                for(int j=0;j<word.length();j++)
                {
                char c = (char) word.charAt(j);
                int num = (int)c;

             if (((num >= (int)'A') && (num <= (int)'Z')) ||
                 ((num >= (int)'a') && (num <= (int)'z'))) {
               wrd = wrd+c;

             } }
if (wrd.length()>0){
              if (title) {titleline=titleline+wrd+" ";
}
              else words.add(wrd);wrd = "";}}

       title=false;

    }

     count++;           } catch  (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{ if (reader != null)
{ reader.close();}
} catch (IOException e){
e.printStackTrace();}
           }
    Stopwords stopwords = new Stopwords();
    if ((words.size() > 0) ||(titleline.length()>0)) {

    try {

           FileWriter outFile = new FileWriter("c:\\nsfiles\\ns"+nf);
              PrintWriter out = new PrintWriter(outFile);
              String titleAr[] = titleline.split(" ");
              String st="";
for (int j = 0; j < titleAr.length; j++) {

                if (!(stopwords.is(titleAr[j].toString()))){

                 st = st+ titleAr[j].toString()+" ";


                }
      }
           if (titleline.length()>0){out.println(st);}
              for (int i = 0; i < words.size(); i++) {

                if (!(stopwords.is(words.get(i).toString()))){

                String s = words.get(i).toString();

                out.print(s+" ");}
      }
            out.close();
          } catch (IOException e){
           e.printStackTrace();

    }
  }
           }
    }}
```

B.2 Porter stemmer:

/*according to **[25]**

Step 1 removes the i-suffixes, and
Step 1a

```
cats      ->  cat
          'sses' (<-'ss')
          'ies'  (<-'i')
          'ss'   ()
          's'    (delete) or <- ''
```

step 1b,

```
   Step 1b:
      EED -> EE
      ED  ->
      ING ->
```

If the second or third of the rules in Step 1b is successful, the following is done:

```
      AT -> ATE
      BL -> BLE
      IZ -> IZE
      (*d and not (*L or *S or *Z)) -> single letter
      (m = 1 and *o) -> E
```

The first part of the rule means that eed maps to ee if eed is in R1 (which is equivalent to m > 0), or ed and ing are removed if they are preceded by a vowel.

```
   define Step_1b as (
      [substring] among (
         'eed'  (R1 <-'ee')
         'ed'
         'ing'  (test gopast v  delete)
      )
   )
```

steps 2 to 4 the d-suffixes. Composite d-suffixes are reduced to single d-suffixes one at a time. So for example if a word ends icational, step 2 reduces it to icate and step 3 to ic. Three steps are sufficient for this process in English.
Step 5 does some tidying up.
But this must be modified by the second part of the rule. *d indicates a test for double letter consonant — bb, dd etc. *L, *S, *Z are tests for l, s, z. *o is a short vowel test — it is matched by consonant-vowel-consonant, where the consonant on the right is not w, x or y. If the short vowel test is satisfied, m = 1 is equivalent to the cursor being at p1. So the second part of the rule means, map at, bl, iz to ate, ble, ize; map certain double letters to single letters; and add e after a short vowel in words of one syllable.

We first need two extra groupings,

```
   define v       'aeiouy'
   define v_WXY    v + 'wxY'   // v with 'w', 'x' and 'y'-consonant
   define v_LSZ    v + 'lsz'   // v with 'l', 's', 'z'
```

and a test for a short vowel,

```
   define shortv as ( non-v_WXY v non-v )
```

(The v_WXY test comes first because we are scanning backwards, from right to left.)

The double to single letter map can be done as follows: first define the slice as the next non-v_LSZ and copy it to a string, ch, as a single character,

```
   strings ( ch )

   ....

   [non-v_LSZ] ->ch
```

A further test, ch, tests that the next letter of the string is the same as the one in ch, and if this gives signal t, delete deletes the slice,

```
   [non-v_LSZ] ->ch  ch  delete
```

Step_1b can then be written like this,

```
   define Step_1b as (
      [substring] among (
         'eed'  (R1 <-'ee')
```

```
           'ed'
           'ing' (
               test gopast v  delete
               (test among('at' 'bl' 'iz')  <+ 'e')
               or
               ([non-v_LSZ]->ch  ch  delete)
               or
               (atmark p1  test shortv  <+ 'e')
           )
       )
   )
letters that need undoubling are b, d, f, g, m, n, p, r and t,
   define Step_1b as (
       [substring] among (
)
   )*/
```

```
public class PorterStemmer {

   /*
*   The input is a file that has the stop words removed
* The output a file with stemmed terms
*[25]
*/
 public String stem(String str) {
        // check for zero length
        if (str.length() > 0) {
            // all characters must be letters
            char[] c = str.toCharArray();
            for (int i = 0; i < c.length; i++) {
                if (!Character.isLetter(c[i]))
                    return "Invalid term";
            }
        } else {
            return "No term entered";
        }
        str = step1a(str);
        str = step1b(str);
        str = step1c(str);
        str = step2(str);
        str = step3(str);
        str = step4(str);
        str = step5a(str);
        str = step5b(str);
        return str;
    } // end stem

    protected String step1a (String str) {
        // SSES -> SS
        if (str.endsWith("sses")) {
            return str.substring(0, str.length() - 2);
        // IES -> I
        } else if (str.endsWith("ies")) {
            return str.substring(0, str.length() - 2);
        // SS -> S
        } else if (str.endsWith("ss")) {
            return str;
        // S ->
        } else if (str.endsWith("s")) {
            return str.substring(0, str.length() - 1);
        } else {
            return str;
        }
    } // end step1a

    protected String step1b (String str) {
        // (m > 0) EED -> EE
        if (str.endsWith("eed")) {
            if (stringMeasure(str.substring(0, str.length() - 3)) > 0)
                return str.substring(0, str.length() - 1);
            else
                return str;
        // (*v*) ED ->
        } else if ((str.endsWith("ed")) &&
                   (containsVowel(str.substring(0, str.length() - 2)))) {if (str.length()>3) {
            return step1b2(str.substring(0, str.length() - 2));}
        // (*v*) ING ->
        } else if ((str.endsWith("ing")) &&
                   (containsVowel(str.substring(0, str.length() - 3)))) {if (str.length()>4) {
            return step1b2(str.substring(0, str.length() - 3));}
        } // end if
        return str;
    } // end step1b
```

```
protected String step1b2 (String str) {
   // AT -> ATE
   if (str.endsWith("at") ||
      str.endsWith("bl") ||
      str.endsWith("iz")) {
      return str + "e";
   } else if ((endsWithDoubleConsonent(str)) &&
            (!(str.endsWith("l") || str.endsWith("s") || str.endsWith("z")))) {
      return str.substring(0, str.length() - 1);
   } else if ((stringMeasure(str) == 1) &&
            (endsWithCVC(str))) {
      return str + "e";
   } else {
      return str;
   }
} // end step1b2

protected String step1c(String str) {
   // (*v*) Y -> I
   if (str.endsWith("y")) {
      if (containsVowel(str.substring(0, str.length() - 1)))
         return str.substring(0, str.length() - 1) + "i";
   } // end if
   return str;
} // end step1c

protected String step2 (String str) {
   // (m > 0) ATIONAL -> ATE
   if ((str.endsWith("ational")) &&
      (stringMeasure(str.substring(0, str.length() - 5)) > 0)) {
      return str.substring(0, str.length() - 5) + "e";
   // (m > 0) TIONAL -> TION
   } else if ((str.endsWith("tional")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
   // (m > 0) ENCI -> ENCE
   } else if ((str.endsWith("enci")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
   // (m > 0) ANCI -> ANCE
   } else if ((str.endsWith("anci")) &&
      (stringMeasure(str.substring(0, str.length() - 1)) > 0)) {
      return str.substring(0, str.length() - 1) + "e";
   // (m > 0) IZER -> IZE
   } else if ((str.endsWith("izer")) &&
      (stringMeasure(str.substring(0, str.length() - 1)) > 0)) {
      return str.substring(0, str.length() - 1);
   // (m > 0) ABLI -> ABLE
   } else if ((str.endsWith("abli")) &&
      (stringMeasure(str.substring(0, str.length() - 1)) > 0)) {
      return str.substring(0, str.length() - 1) + "e";
   // (m > 0) ENTLI -> ENT
   } else if ((str.endsWith("alli")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
   // (m > 0) ELI -> E
   } else if ((str.endsWith("entli")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
   // (m > 0) OUSLI -> OUS
   } else if ((str.endsWith("eli")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
   // (m > 0) IZATION -> IZE
   } else if ((str.endsWith("ousli")) &&
      (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
      return str.substring(0, str.length() - 2);
```

```
    // (m > 0) IZATION -> IZE
    } else if ((str.endsWith("ization")) &&
        (stringMeasure(str.substring(0, str.length() - 5)) > 0)) {
        return str.substring(0, str.length() - 5) + "e";
    // (m > 0) ATION -> ATE
    } else if ((str.endsWith("ation")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3) + "e";
    // (m > 0) ATOR -> ATE
    } else if ((str.endsWith("ator")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
        return str.substring(0, str.length() - 2) + "e";
    // (m > 0) ALISM -> AL
    } else if ((str.endsWith("alism")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
       return str.substring(0, str.length() - 3);
    // (m > 0) IVENESS -> IVE
    } else if ((str.endsWith("iveness")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 0)) {
        return str.substring(0, str.length() - 4);
    // (m > 0) FULNESS -> FUL
    } else if ((str.endsWith("fulness")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 0)) {
        return str.substring(0, str.length() - 4);
    // (m > 0) OUSNESS -> OUS
    } else if ((str.endsWith("ousness")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 0)) {
        return str.substring(0, str.length() - 4);
    // (m > 0) ALITII -> AL
    } else if ((str.endsWith("aliti")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3);
    // (m > 0) IVITI -> IVE
    } else if ((str.endsWith("iviti")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3) + "e";
    // (m > 0) BILITI -> BLE
    } else if ((str.endsWith("biliti")) &&
        (stringMeasure(str.substring(0, str.length() - 5)) > 0)) {
        return str.substring(0, str.length() - 5) + "le";
    } // end if
    return str;
} // end step2


protected String step3 (String str) {
    // (m > 0) ICATE -> IC
    if ((str.endsWith("icate")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3);
    // (m > 0) ATIVE ->
    } else if ((str.endsWith("ative")) &&
        (stringMeasure(str.substring(0, str.length() - 5)) > 0)) {
        return str.substring(0, str.length() - 5);
    // (m > 0) ALIZE -> AL
    } else if ((str.endsWith("alize")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3);
    // (m > 0) ICITI -> IC
    } else if ((str.endsWith("iciti")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3);
    // (m > 0) ICAL -> IC
    } else if ((str.endsWith("ical")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 0)) {
        return str.substring(0, str.length() - 2);
    // (m > 0) FUL ->
    } else if ((str.endsWith("ful")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 0)) {
        return str.substring(0, str.length() - 3);
    // (m > 0) NESS ->
    } else if ((str.endsWith("ness")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 0)) {
        return str.substring(0, str.length() - 4);
    } // end if
    return str;
} // end step3


protected String step4 (String str) {
    if ((str.endsWith("al")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 1)) {
        return str.substring(0, str.length() - 2);
        // (m > 1) ANCE ->
    } else if ((str.endsWith("ance")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 1)) {
        return str.substring(0, str.length() - 4);
    // (m > 1) ENCE ->
    } else if ((str.endsWith("ence")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 1)) {
        return str.substring(0, str.length() - 4);
    // (m > 1) ER ->
    } else if ((str.endsWith("er")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 1)) {
        return str.substring(0, str.length() - 2);
    // (m > 1) IC ->
    } else if ((str.endsWith("ic")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 1)) {
        return str.substring(0, str.length() - 2);
    // (m > 1) ABLE ->
    } else if ((str.endsWith("able")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 1)) {
        return str.substring(0, str.length() - 4);
    // (m > 1) IBLE ->
    } else if ((str.endsWith("ible")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 1)) {
        return str.substring(0, str.length() - 4);
    // (m > 1) ANT ->
    } else if ((str.endsWith("ant")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) EMENT ->
    } else if ((str.endsWith("ement")) &&
        (stringMeasure(str.substring(0, str.length() - 5)) > 1)) {
        return str.substring(0, str.length() - 5);
    // (m > 1) MENT ->
    } else if ((str.endsWith("ment")) &&
        (stringMeasure(str.substring(0, str.length() - 4)) > 1)) {
        return str.substring(0, str.length() - 4);
    // (m > 1) ENT ->
    } else if ((str.endsWith("ent")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) and (*S or *T) ION ->
    } else if ((str.endsWith("sion") || str.endsWith("tion")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) OU ->
    } else if ((str.endsWith("ou")) &&
        (stringMeasure(str.substring(0, str.length() - 2)) > 1)) {
        return str.substring(0, str.length() - 2);
    // (m > 1) ISM ->
    } else if ((str.endsWith("ism")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
```

```
        return str.substring(0, str.length() - 3);
    // (m > 1) ATE ->
    } else if ((str.endsWith("ate")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) ITI ->
    } else if ((str.endsWith("iti")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) OUS ->
    } else if ((str.endsWith("ous")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) IVE ->
    } else if ((str.endsWith("ive")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    // (m > 1) IZE ->
    } else if ((str.endsWith("ize")) &&
        (stringMeasure(str.substring(0, str.length() - 3)) > 1)) {
        return str.substring(0, str.length() - 3);
    } // end if
    return str;
} // end step4


protected String step5a (String str) {
    // (m > 1) E ->
    if ((stringMeasure(str.substring(0, str.length() - 1)) > 1) &&
        str.endsWith("e"))
        return str.substring(0, str.length() -1);
    // (m = 1 and not *0) E ->
    else if ((stringMeasure(str.substring(0, str.length() - 1)) == 1) &&
            (!endsWithCVC(str.substring(0, str.length() - 1))) &&
            (str.endsWith("e")))
        return str.substring(0, str.length() - 1);
    else
        return str;
} // end step5a


protected String step5b (String str) {
    // (m > 1 and *d and *L) ->
    if (str.endsWith("l") &&
        endsWithDoubleConsonent(str) &&
        (stringMeasure(str.substring(0, str.length() - 1)) > 1)) {
        return str.substring(0, str.length() - 1);
    } else {
        return str;
    }
} // end step5b

/*      -------------------------------------------------------
   The following are functions to help compute steps 1 - 5
   -------------------------------------------------------    */

// does string end with 's'?
protected boolean endsWithS(String str) {
    return str.endsWith("s");
} // end function

// does string contain a vowel?
protected boolean containsVowel(String str) {
    char[] strchars = str.toCharArray();
    for (int i = 0; i < strchars.length; i++) {
        if (isVowel(strchars[i]))
            return true;
    }
    // no aeiou but there is y
    if (str.indexOf('y') > -1)
        return true;
    else
        return false;
} // end function

// is char a vowel?
public boolean isVowel(char c) {
    if ((c == 'a') ||
        (c == 'e') ||
        (c == 'i') ||
        (c == 'o') ||
        (c == 'u'))
        return true;
    else
        return false;
} // end function

// does string end with a double consonent?
protected boolean endsWithDoubleConsonent(String str) {
    if (str.length()>0){char c = str.charAt(str.length() - 1);

    if (str.length()>2)
    {
        if (c == str.charAt(str.length() - 2))
        if (!containsVowel(str.substring(str.length() - 2))) {
            return true;
    }}}
    return false;
} // end function

// returns a CVC measure for the string
protected int stringMeasure(String str) {
    int count = 0;
    boolean vowelSeen = false;
    char[] strchars = str.toCharArray();

    for (int i = 0; i < strchars.length; i++) {
        if (isVowel(strchars[i])) {
            vowelSeen = true;
        } else if (vowelSeen) {
            count++;
            vowelSeen = false;
        }
    } // end for
    return count;
} // end function

// does stem end with CVC?
protected boolean endsWithCVC (String str) {
    char c, v, c2 = ' ';
    if (str.length() >= 3) {
        c = str.charAt(str.length() - 1);
        v = str.charAt(str.length() - 2);
        c2 = str.charAt(str.length() - 3);
    } else {
        return false;
    }

    if ((c == 'w') || (c == 'x') || (c == 'y')) {
        return false;
    } else if (isVowel(c)) {
        return false;
    } else if (!isVowel(v)) {
        return false;
        } else if (isVowel(c2)) {
            return false;
        } else {
            return true;
        }
    } // end function



    /**
     * @param args the command line arguments
     */
    public static void main(String[] args){
        PorterStemmer  ps= new PorterStemmer();
        File directory = new File("c:/nsfiles");
     //File directory = new File("C:/nsqueries");
      File files[] = directory.listFiles();
     String filename[] = directory.list();
     int count = 0;
       for (File f : files)
       {
          String nf=f.getName();
          BufferedReader reader = null;

try
{
    FileInputStream fin=null;
    fin = new FileInputStream(f);
    reader = new BufferedReader(new InputStreamReader(fin));
    String text = null;
// repeat until all lines is read

 try {

          FileWriter outFile = new FileWriter("c:\\stemfiles\\stm"+nf);
          // FileWriter outFile = new FileWriter("C:\\stemqueries\\stm"+nf);
            PrintWriter out = new PrintWriter(outFile);
count++;
boolean title=true;
while(reader.ready())
{
       text=reader.readLine();
       String str = text;
       StringBuffer wordBuffer = new StringBuffer();
        for(int i=0;i<str.length();i++)
                  {
                  char c = (char) str.charAt(i);
                  int num = (int)c;

              if (((num >= (int)'A') && (num <= (int)'Z')) ||
                  ((num >= (int)'a') && (num <= (int)'z'))) {
                wordBuffer.append(c);
              } else {
                if (wordBuffer.length() > 0) {
                  String wb= wordBuffer.toString().toLowerCase();
                 out.print(ps.stem(wb));
                  wordBuffer = new StringBuffer();               }
                out.print(c);
              }
               }
       if (title){out.println();
       title= false;
       }
  }
          out.close();
         } catch (IOException e){
          e.printStackTrace();
    }
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (reader != null)
{
reader.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
          }
       }
    }}
```

B.3 Finding term frequencies and document lengths:

```
/*
*   The input is a stemmed file
*   The output is two files one has the term and the document id of the document it appeared in and the frequency of
 *  appearance in that document  and the otherone has the document id and the document length
*/
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package termfreqdoclength;
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import java.io.BufferedReader;
import java.io.*;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Variable;
import java.util.*;
/**
 *
 * @author TOSHIBA
 */
public class Main {
public static LinkedHashMap sortHashMapByValuesD(HashMap <String,Double> passedMap) {
    List mapKeys = new ArrayList(passedMap.keySet());
    List mapValues = new ArrayList(passedMap.values());
    //Collections.sort(mapValues);
    Collections.sort(mapKeys);
    Comparator descending = Collections.reverseOrder();
    Collections.sort( mapValues, descending );
    LinkedHashMap sortedMap =
        new LinkedHashMap();

    Iterator valueIt = mapValues.iterator();
    while (valueIt.hasNext()) {
        Object val = valueIt.next();
        Iterator keyIt = mapKeys.iterator();

        while (keyIt.hasNext()) {
            Object key = keyIt.next();
            String comp1 = passedMap.get(key).toString();
            String comp2 = val.toString();

            if (comp1.equals(comp2)){
                passedMap.remove(key);
                mapKeys.remove(key);
                sortedMap.put((String)key, (Double)val);
                break;
            }       }
    }
    return sortedMap;
}
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
      File directory = new File("F:/stemfiles");
      //String filename[] = directory.list();
```

```
         File files[] = directory.listFiles();
         Map<String,String> fmp=new HashMap<String,String >();
        Map<String, Map<String,Object>> dirmp = new HashMap<String, Map<String,Object>>();
        String filename[] = directory.list();
        Map<String, Map<String,Integer>> word_index = new HashMap<String, Map<String,Integer>>();
        int count = 0;
        Map<String,Map<String,Integer>> titletmp = new HashMap<String,Map<String,Integer>>();
        try {    FileWriter outFile = new FileWriter("F:\\tabletf.txt");
         PrintWriter out = new PrintWriter(outFile);
        try {    FileWriter outFile2 = new FileWriter("F:\\doclength.txt");
         PrintWriter outdlength = new PrintWriter(outFile);
         for (File f : files)
        {
            boolean title=true;
            String nf = f.getName();
            fmp.put(count+"", nf);
            //System.out.println(count+" "+nf);
             BufferedReader reader = null;
         Map<String,Object> mp=new HashMap<String,Object >();
         try
{
reader = new BufferedReader(new FileReader(f));
String text = null;
// repeat until all lines is read
while ((text = reader.readLine()) != null)
{

         String str = text;
         String strAr[] = str.split(" ");

         for(int i=0;i<strAr.length;i++)
                    {
                    String word = strAr[i];
                   if (mp.containsKey(word))    //mp is the term and  its frequency in the file
                   {
                      Integer I = (Integer) mp.get(word);
                      mp.put(word, new Integer(I.intValue()+1));
                   }
                  else
                  {
                     mp.put(word, new Integer(1));
                  }
          if (title){
              Map<String,Integer> value=new HashMap<String,Integer >();
              String fle=count+"";
             if (titletmp.containsKey(word))    //titletmp is the term and  the documents where it appears in title
             {
                value = (HashMap <String,Integer>) titletmp.get(word);
                if (value.containsKey(fle))    //mp is the term and  its frequency in the file
               {
                 Integer I = (Integer) value.get(fle);
                value.put(fle, new Integer(I.intValue()+1));
               }
                else
              {
               value.put(fle, new Integer(1));
              }
             }
             else
             {
              value.put(fle, new Integer(1));
             }
          titletmp.put(word,  value);}
                        }
title=false;
}
} catch (FileNotFoundException e)
```

```
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (reader != null)
{
reader.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
}
        //Get Map in Variable interface to get key and value
        Variable s=mp.entrySet();

        //Move next key and value of Map by iterator
        Iterator it=s.iterator();

        while(it.hasNext())
        {
            // key=value separator this by Map.Entry to get key and value
            Map.Entry m =(Map.Entry)it.next();

            // getKey is used to get key of Map
            int value=(Integer)m.getValue();

            // getValue is used to get value of key in Map
            String key=(String)m.getKey();

    int c=count;
    if (word_index.containsKey(key))     //fills the word index with the term and freq in docs
{
Map<String,Integer> mp2=new HashMap<String,Integer >();
        mp2 = (HashMap <String,Integer> )word_index.get(key);
             int value3;
            //for each file
            Variable s4=mp2.entrySet();
            //Move next key and value of Map by iterator
            Iterator it4= s4.iterator();
            int filelength = 0;
            while(it4.hasNext())
            { // key=value separator this by Map.Entry to get key and value
            Map.Entry m3 =(Map.Entry)it4.next();
            String key3=(String)m3.getKey();

            // getValue is used to get value of key in Map
             value3=(Integer)m3.getValue();
             mp2.put(key3, value3);
              }
             mp2.put(c+"",new Integer(value));
             word_index.put(key, mp2);

            }
           else
             {
            Map<String,Integer> mp2=new HashMap<String,Integer >();
            mp2.put(c+"",new Integer(value));
            word_index.put(key, mp2);
            }
          }
       dirmp.put(count+"", mp);  //fills the file num and the terms frequencies
       count++;
    }
       Map <String,Integer> doclength= new HashMap <String,Integer>(); //each doc and its length
        Variable s2=dirmp.entrySet();
        String key3;
        //Move next key and value of Map by iterator
        Iterator it=s2.iterator();
        while(it.hasNext())
        {Map <String,Object> mvalue =  new HashMap <String,Object> ();
           Map.Entry m =(Map.Entry)it.next();
            String key=(String)m.getKey();
            mvalue =(HashMap <String,Object> ) m.getValue();
             int value3;
            //for each file
            Variable s3=mvalue.entrySet();
            //Move next key and value of Map by iterator
            Iterator it3= s3.iterator();
            int filelength = 0;
            while(it3.hasNext())
            {  Map.Entry m3 =(Map.Entry)it3.next();
            key3=(String)m3.getKey();
            // getValue is used to get value of key in Map
           value3=(Integer)m3.getValue();
           filelength = filelength+ value3;
        }
    doclength.put(key, filelength);     //adds the file and its length
       }
       Variable wrd_dx=word_index.entrySet();
        //Move next key and value of Map by iterator
        Iterator it_x=wrd_dx.iterator();
        while(it_x.hasNext())    //will iterate over the word index to get the term and doc frequencies
        {
           Map.Entry xm =(Map.Entry)it_x.next();
            String key=(String)xm.getKey();
            Map<String,Integer> mp3=new HashMap<String,Integer >();
            mp3 = (HashMap <String,Integer> )xm.getValue();
            Variable mp3_dx=mp3.entrySet();
            Iterator it_x2=mp3_dx.iterator();
             while(it_x2.hasNext())    //will iterate over the word index to get the term and doc frequencies
        {
            Map.Entry mp =(Map.Entry)it_x2.next();
            String doc=(String) mp.getKey();
            int freq=(Integer) mp.getValue();
            out.println(key+" "+doc+" "+freq);
             }
    }
    Variable dl=doclength.entrySet();
    //Move next key and value of Map by iterator
        Iterator it_x3=dl.iterator();
        while(it_x.hasNext())    //will iterate over the word index to get the term and doc frequencies
        {
            Map.Entry dlm =(Map.Entry)it_x3.next();
            String key=(String)dlm.getKey();
            int dlen=(Integer) dlm.getValue();
            outdlength.println(key+" "+dlen);
             }
          outdlength.close();
         }
          catch (IOException e){
           e.printStackTrace();
        }
out.close();
}
catch (IOException e){
           e.printStackTrace();
        }
    }}
```

B.4 Finding title terms and their frequencies :

```
/*
*   The input is the stemmed files
*   The output is a file that has the term and the document id of the document it appeared in its title and the frequency
*   of appearance in that document's title
*/
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package titlefreq;
/*
 * title
 * and open the template in the editor.
 */
import java.io.BufferedReader;
import java.io.*;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Variable;
import java.util.*;

public class Main {
    public static void main(String[] args) {
        File directory = new File("F:/stemfiles");
        File files[] = directory.listFiles();
        Map<String,String> fmp=new HashMap<String,String >();
        String filename[] = directory.list();
        BufferedReader reader = null;
        int count,flength=0;
try {   FileWriter outFile = new FileWriter("F:\\titlet.txt");
        PrintWriter out = new PrintWriter(outFile);
           count=0;
           for (File f : files)
           {Map<String,Integer> titletmp = new HashMap<String,Integer>();
try
{
            reader = new BufferedReader(new FileReader(f));
            String text = null;
            flength=0;
            boolean title=true;
            while (((text = reader.readLine()) != null)&& title)
            {
            String str = text;
            String strAr[] = str.split(" ");
            for(int i=0;i<strAr.length;i++)
             {
               if (title){
                   if (titletmp.containsKey(strAr[i]))   {
                           int value =  titletmp.get(strAr[i]);
                           Integer I = (Integer) value;
                           titletmp.put(strAr[i], new Integer(I.intValue()+1));
                        }
                     else
                       {titletmp.put(strAr[i], new Integer(1));
                       }
                 }
               }
           title=false;
           }
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (reader != null)
{
reader.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
}
Variable s=titletmp.entrySet();
Iterator it=s.iterator();
while(it.hasNext())
{
   Map.Entry m =(Map.Entry)it.next();
   String key =(String)m.getKey();
   int value =(Integer) m.getValue();
   out.println(key+" "+count+" "+value);
}
 count++;   }
out.close();
}
catch (IOException e){
          e.printStackTrace();
        }
}
}
```

B.5 Fuzzification of term frequency and document frequencies ratios using the triangle membership functions:

```
package termfudocfuzmod;
/*
*   The input is a file that has the term and the document it appeared in and the frequency of appearance in that
 * document.  This file does the fuzzification of the term frequencies as well as the document frequency according to
* the triangle function
* The output is two files one is the tf fuzzified and the df fuzzified...the first has each term and the document it
* appeared in and three fuzzification values for the degree of membership in each variable of low and medium and
high
*/
import java.io.*;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        BufferedReader readertf = null;
      try {   FileWriter outFiletf = new FileWriter("F:\\fuztfmodtri.txt");
        PrintWriter outf = new PrintWriter(outFiletf);
      try {   FileWriter outFiledf = new FileWriter("F:\\fuzdfmodtri.txt");
        PrintWriter outdf = new PrintWriter(outFiledf);
try

{           int nofdoc=0;
            readertf = new BufferedReader(new FileReader("f:\\tabletf.txt"));
            String text = null;
            String prevTrem="aa";
            while (((text = readertf.readLine()) != null))     //the loop reads the index line by line each line has the term, the
                                                                   //document it appeared in and its frequency.
           {
            String str = text;
            String strAr[] = str.split(" ");
            String term=strAr[0];
            //check the doc frequency
            if (term.compareTo(prevTrem)==0){nofdoc++;}
            else                          //here the fuzzification according to df which is translated to three  fuzzification
                                          // values representing the membership in the three fuzzy values (low df, medium df, high
df)
            {
             double l, m, h;
             String sd="";
             if (nofdoc <= 1) {l=1;}
             else if((nofdoc > 1) &&  (nofdoc <= 3))
            {   double lx = 1-((nofdoc-1)/2.0);
               int il = (int)( lx * 1000.0); // scale it
               l = ((double)il)/1000.0;
            }else l=0;
            if (nofdoc <= 3) {m=0;}
            else if((nofdoc > 3) &&  (nofdoc <= 9))
            {
            double mx = ((nofdoc - 3) / 6.0);
            int im = (int)( mx * 1000.0); // scale it
            m = ((double)im)/1000.0;
           }else if((nofdoc > 9) &&  (nofdoc <= 15))
               {double mx = 1-((nofdoc - 9) / 6.0);
                int im = (int)( mx * 1000.0); // scale it
                m = ((double)im)/1000.0;}
           else m=0;
          if (nofdoc <= 12) {h=0;}
          else if((nofdoc > 12) &&  (nofdoc <= 24))
          {   double hx = ((nofdoc-12)/12.0);
              int ih = (int)( hx * 1000.0); // scale it
              h = ((double)ih)/1000.0;
          }else h=1;
           sd=sd+l+" "+m+" "+h;   // values representing the membership in the three fuzzy values (low df, medium df,
high df)
```

```
            outdf.println(prevTrem+" "+sd);
            nofdoc=1;
            }
             String doc=strAr[1];
             double freqratio= Double.parseDouble(strAr[2]);
             double lt=0, mt=0, ht=0;
             String s="";
            if (freqratio <= 0.003) {lt=1;}
             else if((freqratio > 0.003) &&  (freqratio <= 0.005))
             {   lt = 1-((freqratio-0.003)/0.002);
                 int il = (int)( lt * 1000.0); // scale it
                 lt = ((double)il)/1000.0;
                 }
            else lt=0;
           if((freqratio > 0.003) &&  (freqratio <= 0.006))
                 {
                   mt = ((freqratio-0.003)/0.003);
                   int im = (int)( mt * 1000.0); // scale it
                   mt = ((double)im)/1000.0;}
           else if((freqratio > 0.006) &&  (freqratio <= 0.01))
            {      mt = 1-((freqratio-0.006)/0.004);
                  int im = (int)( mt * 1000.0); // scale it
                  mt = ((double)im)/1000.0;
            }
           else mt=0;
           if (freqratio <=0.008) ht=0;
           else
           if((freqratio > 0.008) &&  (freqratio <= 0.02))
           {
                  ht = ((freqratio-0.008)/0.003);
                  int ih = (int)( ht * 1000.0); // scale it
                  ht = ((double)ih)/1000.0;}
           else ht = 1;
                   s=lt+" "+mt+" "+ht; // values representing the membership in the three fuzzy values (low tf, medium tf,
high tf)
                  outf.println(term+" "+doc+" "+s);
            prevTrem=term;
           }
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
e.printStackTrace();

} finally{
try{
if (readertf != null)
readertf.close();}
} catch (IOException e){
e.printStackTrace();}
}
outdf.close();}
catch (IOException e){           e.printStackTrace();
        }
outf.close();
}
catch (IOException e){
           e.printStackTrace();

        }

}
}
```

B.6 Fuzzification of term frequency and document frequencies ratios using the S-shape membership functions:

```
/*
*   The input is a file that has the term and the document it appeared in and the frequency of appearance in that
 * document.  This file does the fuzzification of the term frequencies as well as the document frequency according to
*  the S-shape function
* The output is two files one is the tf fuzzified and the df fuzzified...the first has each term and the document it
* appeared in and three fuzzification values for the degree of membership in each variable of low and medium and
high
*/
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package modmembershipfunc;

import java.io.BufferedReader;
import java.io.*;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Variable;
import java.util.*;
/**
 *
 * @author TOSHIBA
 */
public class Main {

    public static void main(String[] args) {


        BufferedReader readertf = null;


try {   FileWriter outFiletf = new FileWriter("F:\\fuztfmodS.txt");
        PrintWriter outf = new PrintWriter(outFiletf);
try {   FileWriter outFiledf = new FileWriter("F:\\fuzdfmodS.txt");
        PrintWriter outdf = new PrintWriter(outFiledf);
try

{            int nofdoc=0;
            readertf = new BufferedReader(new FileReader("f:\\tabletf.txt"));
            String text = null;
            String prevTrem="aa";
            while (((text = readertf.readLine()) != null))
            {
            String str = text;
            String strAr[] = str.split(" ");
            String term=strAr[0];
            //check the doc frequency
            if (term.compareTo(prevTrem)==0){nofdoc++;}
            else
            {
             String sd="";
             if (nofdoc <= 1) {l=1;}
             else if((nofdoc > 1) &&  (nofdoc <= 3))
            {   double lx = 1-((nofdoc-1)/2.0);
                int il = (int)( lx * 1000.0); // scale it
                l = ((double)il)/1000.0;
            }else l=0;
```

```
      if (nofdoc <= 3) {m=0;}
      else if((nofdoc > 3) &&  (nofdoc <= 9))
      {
      double mx = ((nofdoc - 3) / 6.0);
      int im = (int)( mx * 1000.0); // scale it
      m = ((double)im)/1000.0;
     }else if((nofdoc > 9) &&  (nofdoc <= 15))
          {double mx = 1-((nofdoc - 9) / 6.0);
           int im = (int)( mx * 1000.0); // scale it
           m = ((double)im)/1000.0;}
     else m=0;
    if (nofdoc <= 12) {h=0;}
    else if((nofdoc > 12) &&  (nofdoc <= 24))
    {   double hx = ((nofdoc-12)/12.0);
        int ih = (int)( hx * 1000.0); // scale it
        h = ((double)ih)/1000.0;
    }else h=1;
     sd=sd+l+" "+m+" "+h;
    // outdf.print(prevTrem+" "+nofdoc+" ");
     outdf.println(prevTrem+" "+sd);
     //System.out.println(prevTrem+" "+sd);
     nofdoc=1;*/
        //////////////////////Sdf
           double l=0, m=0, h=0;
           String sd="";
   ////lowdf
      if (nofdoc <= 1) {l=1;}
       else if((nofdoc > 1) &&  (nofdoc <= 3))
      {   double lx = 1-2*Math.pow(((nofdoc-1)/4.0),2);
          int il = (int)( lx * 1000.0); // scale it
          l = ((double)il)/1000.0;
      }
       else if((nofdoc > 3) &&  (nofdoc <= 5))
      {   double lx = 2*Math.pow(((nofdoc-5)/4.0),2);
          int il = (int)( lx * 1000.0); // scale it
          l = ((double)il)/1000.0;
      }
       else l=0;
  ////mediumdf
      if (nofdoc <= 1) {m=0;}
      else if((nofdoc > 1) &&  (nofdoc <= 3))
      {
      double mx = 2*Math.pow(((nofdoc-1)/4.0),2);
      int im = (int)( mx * 1000.0); // scale it
      m = ((double)im)/1000.0;
     }else if((nofdoc > 3) &&  (nofdoc <= 5))
          {double mx = 1-2*Math.pow(((nofdoc-5)/4.0),2);
           int im = (int)( mx * 1000.0); // scale it
           m = ((double)im)/1000.0;}
     else if((nofdoc > 5) &&  (nofdoc <= 13))m=1;
    else if((nofdoc > 13) &&  (nofdoc <= 15))
     {      m = 1-2*Math.pow(((nofdoc-13)/4.0),2);
            int im = (int)( m * 1000.0); // scale it
            m = ((double)im)/1000.0;
     }
    else if((nofdoc > 15) &&  (nofdoc <= 17))
          {
             m = 2*Math.pow(((nofdoc-17)/4.0),2);
             int im = (int)( m * 1000.0); // scale it
             m = ((double)im)/1000.0;}
    else m=0;
 ////highdf
    if (nofdoc <= 13) {h=0;}
    else if((nofdoc > 13) &&  (nofdoc <= 19))
    {   double hx = 2*Math.pow(((nofdoc-13)/12.0),2);
        int ih = (int)( hx * 1000.0); // scale it
        h = ((double)ih)/1000.0;
    }
    else
       if((nofdoc > 19) &&  (nofdoc <= 25))
          {
            h = 1-2*Math.pow(((nofdoc-25)/12.0),2);
             int ih = (int)( h * 1000.0); // scale it
             h = ((double)ih)/1000.0;}
       else h=1;

     sd=sd+l+" "+m+" "+h;
    // outdf.print(prevTrem+" "+nofdoc+" ");
     outdf.println(prevTrem+" "+sd);
     //System.out.println(prevTrem+" "+sd);
     nofdoc=1;
     }
      String doc=strAr[1];
      double freqratio= Double.parseDouble(strAr[2]);
      double lt=0, mt=0, ht=0;
      String s="";
     if (freqratio <= 0.002) {lt=1;}
      else if((freqratio > 0.002) &&  (freqratio <= 0.003))
      {   lt = 1-2*Math.pow(((freqratio-0.002)/0.002),2);
          int il = (int)( lt * 1000.0); // scale it
          lt = ((double)il)/1000.0;
          }
    else if((freqratio > 0.003) &&  (freqratio <= 0.004))
      {   lt = 2*Math.pow(((freqratio-0.004)/0.002),2);
          int il = (int)( lt * 1000.0); // scale it
          lt = ((double)il)/1000.0;
          }else lt=0;
  if((freqratio > 0.003) &&  (freqratio <= 0.004))
          {
             mt = 2*Math.pow(((freqratio-0.003)/0.002),2);
             int im = (int)( mt * 1000.0); // scale it
             mt = ((double)im)/1000.0;}
    else if((freqratio > 0.004) &&  (freqratio <= 0.005))
     {      mt = 1-2*Math.pow(((freqratio-0.005)/0.002),2);
            int im = (int)( mt * 1000.0); // scale it
            mt = ((double)im)/1000.0;
     }
    else if((freqratio > 0.005) &&  (freqratio <= 0.008))mt=1;
    else if((freqratio > 0.008) &&  (freqratio <= 0.009))
     {      mt = 1-2*Math.pow(((freqratio-0.008)/0.002),2);
            int im = (int)( mt * 1000.0); // scale it
            mt = ((double)im)/1000.0;
     }
    else if((freqratio > 0.009) &&  (freqratio <= 0.01))
          {
             mt = 2*Math.pow(((freqratio-0.01)/0.002),2);
             int im = (int)( mt * 1000.0); // scale it
             mt = ((double)im)/1000.0;}
    else mt=0;
      if (freqratio <=0.008) ht=0;
    else
     if((freqratio > 0.008) &&  (freqratio <= 0.014))
    {
            ht = (2*Math.pow(((freqratio-0.008)/0.012),2));
            int ih = (int)( ht * 1000.0); // scale it
            ht = ((double)ih)/1000.0;}
    else
      if((freqratio > 0.014) &&  (freqratio <= 0.02))
    {
            ht = 1-2*Math.pow(((freqratio-0.02)/0.012),2);
            int ih = (int)( ht * 1000.0); // scale it
            ht = ((double)ih)/1000.0;}
    else ht = 1;
             s=lt+" "+mt+" "+ht;
```

```
            outf.println(term+" "+doc+" "+s);
            prevTrem=term;
           }
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (readertf != null)
{
readertf.close();}
} catch (IOException e)
{
e.printStackTrace();}
}
outdf.close();
}
catch (IOException e){
           e.printStackTrace();
        }
outf.close();}
catch (IOException e){
           e.printStackTrace();       }}
```

B.7 Query matching, fuzzification of title frequency and over lap ratio, inference then deffuzificatioin and result ranking:

```
package termratiomod;
/*
*   The input is two fuzzified files for tf and df  plus the title file
*The fuzzy inference rules are applied and then defuzzification and sorting
* The output is a file with the first 1000 relevant document for each query sorted in descending order according to
*  relevance
*/

/*
 * To match queries
 */
import java.io.*;
import java.util.*;
/**
 *
 *  */
public class Main {
//--------------------------------------------------------------Sorting method----------------------------------------------------
public static LinkedHashMap sortHashMapByValuesD(HashMap <String,Double> passedMap) {
    List mapKeys = new ArrayList(passedMap.keySet());
    List mapValues = new ArrayList(passedMap.values());
    Collections.sort(mapKeys);
    Comparator descending = Collections.reverseOrder();
    Collections.sort( mapValues, descending );
    LinkedHashMap sortedMap =
        new LinkedHashMap();

    Iterator valueIt = mapValues.iterator();
    while (valueIt.hasNext()) {
        Object val = valueIt.next();
        Iterator keyIt = mapKeys.iterator();

        while (keyIt.hasNext()) {
            Object key = keyIt.next();
            String comp1 = passedMap.get(key).toString();
            String comp2 = val.toString();

            if (comp1.equals(comp2)){
                passedMap.remove(key);
                mapKeys.remove(key);
                sortedMap.put((String)key, (Double)val);
                break;
            }

        }

    }
    return sortedMap;
}
//-------------------------------------------------------------------------------------------------------------------------------------
-------
    public static void main(String[] args) {

        BufferedReader readerfn = null;
        BufferedReader readertf = null;
        BufferedReader readerdf = null;
        BufferedReader readertl= null;

        Map <String,Integer> lrlvd= new HashMap <String,Integer>();
        Map <String,Integer> mrlvd= new HashMap <String,Integer>();
//these files are used to see the distribution of high rlv , medium rlv, and low relevance
```

```
try {    FileWriter outFilehrlvdstrb = new FileWriter("F:\\hrlvdstrb.txt");
        PrintWriter outhrlvdstrb = new PrintWriter(outFilehrlvdstrb);
try {    FileWriter outFilemrlvdstrb = new FileWriter("F:\\mrlvdstrb.txt");
        PrintWriter outmrlvdstrb = new PrintWriter(outFilemrlvdstrb);
 try {    FileWriter outFilelrlvdstrb = new FileWriter("F:\\lrlvdstrb.txt");
        PrintWriter outlrlvdstrb = new PrintWriter(outFilelrlvdstrb);
       File qdirectory = new File("c:/47stemqueries with hedges");
       File qfiles[] = qdirectory.listFiles();
       Map<String,String> qfmp=new HashMap<String,String >();
       Map<String,Map<String,Double>> qryrelvdoc=new HashMap<String,Map<String,Double>>(); //the query and
list of
                                                                                              //relevant docs and their
relevance value
        int qcount = 0;
        boolean t;
        for (File qf : qfiles)                         //for each query do the following
       {    //String qdw="";

           double docwight;
           t=true;
           String qn="";
           Map <String,String> qfuz_df= new HashMap <String,String>();
           Map<String,  Map<String,String>> qfuz_tf = new HashMap<String, Map<String,String>>();
           Map<String,Variable> qoverlapmp = new HashMap<String,Variable>();
           Map <String,Double> qfuz_tlh= new HashMap <String,Double>();
           Map <String,Double> qfuz_tlm= new HashMap <String,Double>();
           Map <String,Map<String,Double>> qtrelvdoc =  new HashMap <String,Map<String,Double>> ();

           String nqf = qf.getName();
           BufferedReader qreader = null;
           Variable qtitledocset = new HashSet();

           try
           {
            qreader = new BufferedReader(new FileReader(qf));
            String qtext = null;
           while ((qtext = qreader.readLine()) != null)
           {
            if (t) {qn=qtext;
            qfmp.put(qn+"", nqf);
           }
            else
           {                                          //separate the query terms and their hedges to consider them one by one
and
            String querystr = qtext;
            String qstrAr1[] = querystr.split(" ");
            String qstr="";
            String Hstr="";
            for(int i=1;i<qstrAr1.length;i=i+2)
             {qstr=qstr+qstrAr1[i]+" ";}
             for(int i=0;i<qstrAr1.length;i=i+2)
             {Hstr=Hstr+qstrAr1[i]+" ";}
             qstr=qstr.trim();
             Hstr=Hstr.trim();
             String qstrAr[] = qstr.split(" ");
             String HstrAr[]=Hstr.split(" ");
                  for(int i=0;i<qstrAr.length;i++)          //for each term in the query
                 {
                  String Hedge=HstrAr[i];
                  try {        readertf = new BufferedReader(new FileReader("f:\\fuztfmodtri.txt"));
                  try {        readerdf = new BufferedReader(new FileReader("f:\\fuzdfmodS.txt"));
                  try {        readertl = new BufferedReader(new FileReader("f:\\titletrm.txt"));
                  String qword = qstrAr[i];
                 //calculating overlap fuz values

   /////////////////filling the overlap map and tf
                 String text1 = null;
```

```
                Variable docset=new HashSet();
                Map<String,String> qtfvalue=new HashMap<String,String >();
                boolean in=false;
                while (((text1 = readertf.readLine()) != null)) //will iterate over the fuzzified term frequency  to get the
term,
                                                                                    //doc and fuzzified frequencies and fill the tf map only
when the
                                                                                    //query term matches the index term

                {
                 String str = text1;
                 String strAr[] = str.split(" ");
                 String fl =strAr[1];
                 if (qword.compareTo(strAr[0])==0){

                     qtfvalue.put(fl,strAr[2]+" "+strAr[3]+" "+strAr[4]);
                     docset.add(fl);
                     in=true;
                 }
                }
                qoverlapmp.put(qword,docset);
                if (in){qfuz_tf.put(qword,qtfvalue);}
 ////////////////////////////////////////ol&tf
  //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\filling df
                String qdfvalue="";
                String text2=null;
                while (((text2 = readerdf.readLine()) != null)) //will iterate over the fuzzified document  frequency  to get
the
                                                                                    //term and fuzzified doc frequencies and fill the df map
only
                                                                                    //when the query term matches the index term
                {
                String str = text2;
                String strAr[] = str.split(" ");
                if (qword.compareTo(strAr[0])==0){

                     qdfvalue=strAr[1]+" "+strAr[2]+" "+strAr[3];

                  }
                }
                if (qdfvalue.compareTo("")!=0){qfuz_df.put(qword,qdfvalue);}

 //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\filling titlemp with terms relevant to the query term
               String text3 = null;
               double mtitle=0;
                //double htitle=0;
               while (((text3 = readertl.readLine()) != null))    //will iterate over the term title  frequencies  to get the
                                                                                    //term, document and title frequencies and fuzzify the
                                                                                    //frequencies and fill the title map only when the query
term
                                                                                    //matches the index term

               {
                String str = text3;
                String strAr[] = str.split(" ");
                String file=strAr[1];
                String freq=strAr[2];
                int titletf =Integer.parseInt(freq);
                if (qword.compareTo(strAr[0])==0){
               //this function was tried but didn't give good results
               /* if (titletf ==1){htitle=0.7;mtitle=1;}
                   else if (titletf ==2){htitle=0.8;mtitle=0.9;}
                   else if ((titletf ==3)||(titletf ==4)){htitle=0.9;mtitle=0.8;}
                   else if (titletf >=5){htitle=1;mtitle=0.7;}*/
               //this function was tried and improved the results
               if (titletf ==1){mtitle=0.7;}
                   else if (titletf ==2){mtitle=0.8;}
```

```
                    else if ((titletf ==3)||(titletf ==4)){mtitle=0.9;}
                    else if (titletf >=5){mtitle=1;}
                  }
                }
               if (mtitle!=0){qfuz_tlm.put(qword, mtitle);}
             //  if (htitle!=0){qfuz_tlh.put(qword, htitle);}

  //\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\end filling titlemp
//tl catch
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (readertl != null)
{
readertl.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
    }
//df catch
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (readerdf != null)
{
readerdf.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
   }
//tf catch
} catch (FileNotFoundException e)

{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (readertf != null)
{
readertf.close();
}
} catch (IOException e)
{
e.printStackTrace();
```

```
}
    }
             }//foreach query term
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\start calculations
                String qr="";
                String Hr="";
                String qAr1[] = qtext.split(" ");
                for(int i=1;i<qAr1.length;i=i+2)
                  {qr=qr+qAr1[i]+" ";}
                for(int i=0;i<qAr1.length;i=i+2)
                  {Hr=Hr+qAr1[i]+" ";}
                   qr=qr.trim();
                   Hr=Hr.trim();
                   String qAr[] = qr.split(" ");
                   String HAr[]=Hr.split(" ");
                //System.out.println(qr+" "+Hr);
                for(int j=0;j<qAr.length;j++)                              // for each query term
                {
                Map<String,Double> tdw=new HashMap<String,Double >();
                String qw=qAr[j];
                double ldf = 0;
                double mdf=0;
                double hdf=0;
                if (qfuz_df.containsKey(qw))   //locating the query term in in the df map to get the ldf,mdf and hdf
                                                 //membership values
{
                String qdfvalue=qfuz_df.get(qw);

                int jdf=0;
                String qdoc_fzdfAr[] = qdfvalue.split(" ");
                ldf=Double.parseDouble(qdoc_fzdfAr[jdf]);
                mdf=Double.parseDouble(qdoc_fzdfAr[jdf+1]);
                hdf=Double.parseDouble(qdoc_fzdfAr[jdf+2]);
                Map<String,String> qfztf=new HashMap<String,String >();
                //will get the tf of the current term//locating the current  query term in the tf map to get the ltf,mtf and htf
                                                      //membership values and
                qfztf = (HashMap <String,String> ) qfuz_tf.get(qw);

                int jtf=0;
               //will iterate over the documents and the frequencies and jump to the next doc and frequency
                 Variable s5=qfztf.entrySet();
                 String key5;
               //Move next key and value of Map by iterator
                 Iterator it5=s5.iterator();
                 while(it5.hasNext())
                 {//Map <String,Object> mvalue =  new HashMap <String,Object> ();
                  // key=value separator this by Map.Entry to get key and value
                  Map.Entry m5 =(Map.Entry)it5.next();
                  String doc=(String)m5.getKey();
                  //System.out.println("file :"+doc+"  file index :");
                  String fuzfreq =(String)m5.getValue();
                  docwight=0;
                  String fuzfreqAr[] = fuzfreq.split(" ");
                 /* get each doc and its fuzzified tf*/
                  //String doc = qdoc_fztfAr[jtf];
                  double ltf=Double.parseDouble(fuzfreqAr[0]);
                  double mtf=Double.parseDouble(fuzfreqAr[1]);
                  double htf=Double.parseDouble(fuzfreqAr[2]);
                  /*experiment area for tf and df*/
                   /* tf*/
                 // double hrlv = 0;
                //  double hrlv = 0;
                  //double mrlv = 0;
                  //double lrlv = 0;
                  /*end of experiment area*/
                  //System.out.println("hrlv="+hrlv+"mrlv"+mrlv+"lrlv"+lrlv);
                /* combine with df for the current term*/
                 // double hrlv = Math.max(Math.min(ldf,htf),Math.min(ldf,mtf));

                  double hrlv = Math.max(Math.min(ldf,htf),Math.min(ldf,mtf));
                  double mrlv = Math.max(Math.max(Math.min(hdf,htf),Math.min(mdf,htf)),Math.max(Math.min(ldf,
ltf),Math.min(mdf, mtf)));
                  double lrlv = Math.max(Math.max(Math.min(hdf,mtf),Math.min(mdf,ltf)),Math.min(hdf,ltf));
                 /* double hrlv = ldf*htf+ldf*mtf;
                  double mrlv = hdf*htf + mdf*htf + ldf*ltf + mdf* mtf;
                  double lrlv = hdf*mtf+mdf*ltf+hdf*ltf;*/
                  //System.out.println("hrlv="+hrlv+"mrlv"+mrlv+"lrlv"+lrlv);

                  double mtitle=0;
                  if (qfuz_tlm.containsKey(qw))    //for each termtitletmp is the term and  the documents where it appears
in title
                  {
                     mtitle=qfuz_tlm.get(qw);
                  }
                   mrlv = Math.max(mrlv,mtitle);

                 /* double htitle=0;
                  if (qfuz_tlh.containsKey(qw))    //for each termtitletmp is the term and  the documents where it appears in
title
                  {
                     htitle=qfuz_tlh.get(qw);
                  }
                   hrlv = Math.max(hrlv,htitle);*/
//experiment area for title*/


                 //if (HAr[j].compareTo("slightly")==0) {hrlv=Math.pow(hrlv,1.7);}
                 // else if (HAr[j].compareTo("Indeed")==0) {hrlv=Math.pow(hrlv,3);}
                     //docwight=hrlv; //only for title
              // calculate fuz values of overlap
                double lolp=0;
                double molp=0;
                double holp=0;
                double ovlpcount=0;
                for(int o=0;o<qAr.length;o++)
                { //for each term in the query
                  if (qoverlapmp.containsKey(qAr[o]))
                     {
                      Variable docs = (Variable) qoverlapmp.get(qAr[o]);
                      if (docs.contains(doc)){ ovlpcount = ovlpcount+1; }
                     }
                 }
                double ovlpratio=ovlpcount/qAr.length;
               /* if (ovlpratio <= 0.2) {lolp=1;}
                else if((ovlpratio <= 0.3))
                    {   lolp = 1-((ovlpratio-0.2)/0.1);
                       int sl = (int)( lolp * 1000.0); // scale it
                       lolp = ((double)sl)/1000.0;
                       molp = ((ovlpratio-0.2)/0.1);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
                       holp = 0;
                    }
                    else if((ovlpratio <= 0.5) )
                          {   lolp = 0;
                             molp = 1-((ovlpratio-0.3)/0.2);
                             int sm = (int)( molp * 1000.0); // scale it
                             molp = ((double)sm)/1000.0;
                             holp = ((ovlpratio-0.3)/0.2);
                             int sh = (int)( holp * 1000.0); // scale it
                             holp = ((double)sh)/1000.0;
                          }
                         else {holp=1;}*/
                    /*combine overlap fuz values with other fuz values
                   hrlv=Math.max(hrlv,holp);
                    mrlv=Math.max(mrlv,molp);
                    lrlv=Math.max(lrlv,lolp);*/


/////////////////////////////////////////////////////////////new overlap

        /*  if (ovlpratio < 0.7) {molp=ovlpratio+0.3;}
              else {molp=1;}
              mrlv=Math.max(mrlv,molp);*/

             /*if (ovlpratio > 0.8) {holp=1;}
              else if (ovlpratio <=0.8) {holp= ovlpratio+ 0.2;}*/

//m2
            /*  if (ovlpratio > 0.9) {molp=0.2;}
              else if (ovlpratio >= 0.3) {molp= ovlpratio+ 0.2;}
              else  molp = 0.2;*/
//m3
             /* if (ovlpratio > 0.6) {molp=1;}
              else {molp= ovlpratio+ 0.2;}*/


           // hrlv=Math.max(hrlv,holp);
           //  mrlv=Math.max(mrlv,molp);
              /////////////this is the best
               if (ovlpratio < 0.7) {molp=ovlpratio+0.3;}
              else {molp=1;}
              mrlv=Math.max(mrlv,molp);
               //mrlv=Math.max(mrlv, molp);
                  //only for ol*/
  /////////////////////////////////////////////////////////////new overlap'

           /* if (ovlpratio <= 0.3) {lolp=1;}
              else if(ovlpratio <= 0.4)
                   {   lolp = 1-((ovlpratio-0.3)/0.1);
                       int sl = (int)( lolp * 1000.0); // scale it
                       lolp = ((double)sl)/1000.0;
                    }
              else lolp=0;


              if (ovlpratio <= 0.3){molp=0;}
              else if ( ovlpratio <=0.5)
              {
                       molp = ((ovlpratio - 0.3) / 0.2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
              } else if ( ovlpratio <=0.7){
                       molp = 1-((ovlpratio - 0.5) / 0.2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
              }else molp=0;



                   if (ovlpratio <= 0.5)  {   holp = 0;}
                   else if (ovlpratio <= 0.6)
                         {
                             holp = ((ovlpratio-0.5)/0.1);
                             int sh = (int)( holp * 1000.0); // scale it
                             holp = ((double)sh)/1000.0;
                          }
                         else {holp=1;}*/
                   /*combine overlap fuz values with other fuz values
                   hrlv=Math.max(hrlv,holp);
                  mrlv=Math.max(mrlv,molp);
                  lrlv=Math.max(lrlv,lolp);*/

 ////////////////////////////Sfunction ol/////////////////////

/*if (ovlpratio <= 0.2) {lolp=1;}
              else if(ovlpratio <= 0.3)
                   {   lolp = 1-2*Math.pow(((ovlpratio-0.2)/0.2),2);
                       int sl = (int)( lolp * 1000.0); // scale it
                       lolp = ((double)sl)/1000.0;
                    }
             else if(ovlpratio <= 0.4)
                   {   lolp = 2*Math.pow(((ovlpratio-0.4)/0.2),2);
                       int sl = (int)( lolp * 1000.0); // scale it
                       lolp = ((double)sl)/1000.0;
                    }
else
          lolp=0;

              if (ovlpratio <= 0.3){molp=0;}
              else if ( ovlpratio <=0.4)
              {
                       molp = 2*Math.pow(((ovlpratio-0.3)/0.2),2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
              } else if ( ovlpratio <=0.5){
                       molp = 1-2*Math.pow(((ovlpratio-0.5)/0.2),2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
              }                 else if ( ovlpratio <=0.6)
              {
                       molp = 1-2*Math.pow(((ovlpratio-0.5)/0.2),2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;
              } else if ( ovlpratio <=0.7){
                       molp = 2*Math.pow(((ovlpratio-0.7)/0.2),2);
                       int sm = (int)( molp * 1000.0); // scale it
                       molp = ((double)sm)/1000.0;}
               else molp=0;

                   if (ovlpratio <= 0.4)  {   holp = 0;}
                   else if (ovlpratio <= 0.55)
                         {
                             holp = 2*Math.pow(((ovlpratio-0.4)/0.3),2);
                             int sh = (int)( holp * 1000.0); // scale it
                             holp = ((double)sh)/1000.0;
                          }
                         else if (ovlpratio <= 0.7)
                         {
                             holp = 1-2*Math.pow(((ovlpratio-0.7)/0.3),2);
                             int sh = (int)( holp * 1000.0); // scale it
                             holp = ((double)sh)/1000.0;
                          }
                       else {holp=1;} */
/*combine overlap fuz values with other fuz values
                  hrlv=Math.max(hrlv,holp);
                  mrlv=Math.max(mrlv,molp);
                   lrlv=Math.max(lrlv,lolp);*/



              /* if (HAr[j].compareTo("slightly")==0) {hrlv=Math.pow(hrlv,1.3);}
              else if (HAr[j].compareTo("indifferent")==0) {hrlv=Math.pow(hrlv,2);}
              else {if ((hrlv >=0) && (hrlv<=0.5)) {hrlv=2*Math.pow(hrlv,2);}else {hrlv=1-2*Math.pow(1-hrlv,2);}}*/
```

```
                    //using hedeges as operators to improve the retrieval results
                if (HAr[j].compareTo("slightly")==0) {hrlv=Math.pow(mrlv,3);}
               else if (HAr[j].compareTo("Indeed")==0) {hrlv=Math.pow(mrlv,1/3);}

               /////////////////for relvance distribution
/*
String h=hrlv+"";
String m=mrlv+"";
String l=lrlv+"";
           if (hrlvd.containsKey(h))    //titletmp is the term and  the documents where it appears in title
              {
               Integer I = (Integer) hrlvd.get(h);
               hrlvd.put(h, new Integer(I.intValue()+1));
              }
             else
              {
               hrlvd.put(h, new Integer(1));
              }

           if (mrlvd.containsKey(m))    //titletmp is the term and  the documents where it appears in title
              {
               Integer I = (Integer) mrlvd.get(m);
               mrlvd.put(m, new Integer(I.intValue()+1));
              }
             else
              {
               mrlvd.put(m, new Integer(1));
              }

           if (lrlvd.containsKey(l))    //titletmp is the term and  the documents where it appears in title
              {
               Integer I = (Integer) lrlvd.get(l);
               lrlvd.put(l, new Integer(I.intValue()+1));
              }
             else
              {
               lrlvd.put(l, new Integer(1));
              }

*/
double xh=1,xm=0.72,xl=0.1;
               /* defuzzify relevance  */
                  docwight=(hrlv*xh + mrlv*xm + lrlv*xl)/(hrlv+mrlv+lrlv);
                  int sdw = (int)( docwight * 1000000.0); // scale it
                  docwight = ((double)sdw)/1000000.0;


                  tdw.put(doc, docwight);//write to file qt,doc,w

                   }

            qtrelvdoc.put(qw, tdw);
                   }         }
         }
       t = false;
             }

 Map<String,Double> rlvmp=new HashMap<String,Double >();
 Variable qtrlv=qtrelvdoc.entrySet();
       //Move next key and value of Map by iterator
       Iterator it_qtrlv=qtrlv.iterator();
       while(it_qtrlv.hasNext())
       {
          Map.Entry qt =(Map.Entry)it_qtrlv.next();
          String qterm=(String)qt.getKey();
          Map<String,Double> qtrlvdcs=new HashMap<String,Double >();
          qtrlvdcs = (HashMap <String,Double> )qt.getValue();
```

```
            Variable h=qtrlvdcs.entrySet();
            Iterator it_qtrlvdocs=h.iterator();
            while(it_qtrlvdocs.hasNext())
            {
              Map.Entry qrd =(Map.Entry)it_qtrlvdocs.next();
              String qtrlvdoc = (String) qrd.getKey();
              double ct = (Double) qrd.getValue();
              if (rlvmp.containsKey(qtrlvdoc))    //rlvmp is the term and  its frequency in the file
              {
               Double w = (Double) rlvmp.get(qtrlvdoc);
               rlvmp.put(qtrlvdoc, new Double(w.doubleValue()+ct));

               }
               else
               {
                rlvmp.put(qtrlvdoc, new Double(ct));

               }
            }
           }
rlvmp=sortHashMapByValuesD((HashMap <String,Double> )rlvmp);
       qryrelvdoc.put(qn,rlvmp );
qcount++;
          //end of line of query

//q catch
} catch (FileNotFoundException e)
{
e.printStackTrace();
} catch (IOException e)
{
e.printStackTrace();
} finally
{
try
{
if (qreader != null)
{
qreader.close();
}
} catch (IOException e)
{
e.printStackTrace();
}
}
System.out.println(qcount);
       }

///output the results
  try {
          FileWriter outF = new FileWriter("c:\\qrel5\\ttrids-h.txt");
          PrintWriter out = new PrintWriter(outF);
          Variable qrvd=qryrelvdoc.entrySet();
       //Move next key and value of Map by iterator
          Iterator it_qvd=qrvd.iterator();
          while(it_qvd.hasNext())    //will iterate over the query fuzzified term frequency  to get the term and doc
                                     //fuzzified  frequencies
          { HashMap<String,Double> rvmp=new HashMap<String,Double >();
            Map.Entry q =(Map.Entry)it_qvd.next();
            String query=(String)q.getKey();
             rvmp= (HashMap <String,Double> ) q.getValue();
            Variable vd=rvmp.entrySet();
            //Move next key and value of Map by iterator
            Iterator it_vd=vd.iterator();
            String document="";
            int documentcount=1;
            while(it_vd.hasNext()&& documentcount<=1000)    //will iterate over the query fuzzified term frequency  to
```

```
                                                                        // get  the term and doc fuzzified frequencies
          {
           Map.Entry d =(Map.Entry)it_vd.next();
           String dc=(String)d.getKey();
           double rvw= (Double) d.getValue();
           try {        readerfn = new BufferedReader(new FileReader("f:\\filnams.txt"));
               String text4=null;
               int ln=Integer.parseInt(dc);
               for(int i = 0; i < ln; i++)   readerfn.readLine();
               text4=readerfn.readLine();
               String str = text4;
               String strAr[] = str.split(" ");
               if (dc.compareTo(strAr[0])==0){

                 document=strAr[1].substring(5);
               }
               } catch (FileNotFoundException e)
               {
                e.printStackTrace();
               } catch (IOException e)
               {
               e.printStackTrace();
               } finally
               {
               try
               {
               if (readerfn != null)
               {
               readerfn.close();
               }
               } catch (IOException e)
               {
               e.printStackTrace();
               }
               }
          String docnm=document.substring(0,38)+"xml";
          out.println(query+" "+0+" "+docnm+" "+documentcount+" "+rvw+" run1");
          documentcount++;
          }
         }
        out.close();        } catch (IOException e){
        e.printStackTrace();
   }
}}
```