A
Dissertation
On

Document Ranking and Classification using Cosine Similarity and Parameter Free Threshold

Submitted as a course in Partial fulfilment of the requirements
for the award of degree of
MASTER OF ENGINEERING
in
Computer Technology & Application

Submitted By:
GOURAV BATHLA
College Roll No: 06/CTA/08
University Roll No. 8404

Under the Guidance of:
Mrs. Rajni Jindal
Assistant Professor
Department of Computer Engineering
Delhi College of Engineering, Delhi

DEPARTMENT OF COMPUTER ENGINEERING
DELHI COLLEGE OF ENGINEERING
DELHI UNIVERSITY

JUNE 2010

CERTIFICATE

DELHI COLLEGE OF ENGINEERING
(Delhi University)
BAWANA ROAD, DELHI - 110042

This is certified that the work contained in this dissertation entitled **Document Ranking and Classification using Cosine Similarity and Parameter Free Threshold** submitted by Gourav Bathla, University Roll No- 8404 in the requirement for the partial fulfilment for the award of degree of Master of Engineering in Computer Technology & Application at Delhi College of Engineering is an account of his work carried out under my guidance and supervision in the academic year 2009-2010.

He has proposed and implemented technique for threshold calculation in Document Ranking and Classification. His work is found to be satisfactory. His enthusiasm, attitude and aptitude towards dissertation are appreciated.

I wish him success in future.

Mrs. Rajni Jindal
Assistant Professor
Dissertation Guide
Department Of Computer Engineering
Delhi College of Engineering, Delhi

Acknowledgement

It is a great pleasure to have the opportunity to extend my heartiest felt gratitude to everybody who helped me throughout the course of this dissertation.

It is distinct pleasure to express my deep sense of gratitude and indebtedness to my learned supervisor Mrs. Rajni Jindal, Assistant Professor, Department of Computer Engineering for her invaluable guidance, encouragement and patient reviews. With her continuous inspiration only, it becomes possible to complete this dissertation.

I would also like to take this opportunity to present my sincere regards to all the faculty members of the Department for their support and encouragement.

I am grateful to my parents for their moral support all the time; they have been always around to cheer me up, in the odd times of this work. I am also thankful to my classmates for their unconditional support and motivation during this work. Last but not least, my special thanks to community who is active in the field of Data Mining and Information Retrieval.

GOURAV BATHLA
Master of Engineering
(Computer Technology & Application)
College Roll No. 06/CTA/08
University Roll No. 8404
Department of Computer Engineering
Delhi College of Engineering,
Bawana Road, Delhi-110042

Abstract

Information Retrieval is the science of searching information within documents. Documents are in huge quantity and still growing. It is very difficult to find the information according to requirements of user. So different algorithms are being proposed based on long research in information retrieval and data mining.

Search Engines are important application of Information Retrieval and programs which are used for effective and efficient retrieval of information as required by the user. Search Engine gives results based on some algorithms to index and rank documents and calculates the similarity of query with the corpus of documents. Vector Space Model are used to index documents with documents represented as vectors and ranking is calculated by Term Frequency/Inverse Document Frequency (TF/IDF) and Cosine Similarity. In this Thesis, Keyword based Search are used for ranking of documents

Documents are ranked as required by the user, but there are wide categories of documents like business, sports, education etc. Document Classification is used to categorize a document into predefined class according to its content. In this Thesis, Nearest Neighbor Algorithm is used for classification.

Similarity scores are calculated between a testing document and predefined classes. Some threshold is set so that the document which is having score above that threshold only be classified into that class. But after a lot of research in calculating threshold value, still problems like false positive and false negative exist. In this thesis S-Cut algorithm for calculating threshold value is used and a novel approach for calculation

of threshold value is given which is adaptive and parameter free. Local threshold is calculated based on number of terms matched for each class. Documents are also ranked for each class individually. TF/IDF and Vector Space Model is used for Document Indexing and Document Ranking and cosine similarity is used as distance measure for Document Classification.

Organization of the Thesis

Chapter 1 gives details about Information Retrieval, its requirements and applications in real life. Performance is computed of algorithm by precision and recall .Mathematical models are explained which are used for extracting information from corpus of documents.

Chapter 2 gives details about Search Engine, the most important application of Information Retrieval. Search Engine gives output after sequence of Crawling, Indexing and Ranking of Documents. Static Algorithm i.e. Keyword Search Algorithm and Dynamic Algorithm i.e. PageRank Algorithm is explained .In this thesis work static approach is used for document ranking.

Chapter 3 gives details about Term Frequency/Inverse Document Frequency which is calculated based on number of terms in a document and significance of calculating it is explained and use of it in thesis is also shown.

Chapter 4 gives details about Vector Space Model. Documents are shown to be represented in the form of vectors. Example of indexing of documents and then ranking of documents based on cosine similarity and TF/IDF is explained and the use of this model in thesis is explained.

Chapter 5 gives details about Document Classification, its requirements and its applications. Nearest Neighbor algorithm is explained in detail and the use of this algorithm and threshold calculation in thesis is explained.

Chapter 6 gives details about proposed technique for threshold calculation in Document Classification. Previous work for threshold calculation is explained. Algorithm for calculation is proposed.

Chapter 7 gives details about implementation of document ranking and classification in Java Netbeans. Documents are ranked using vector space model and cosine similarity. Documents are classified using Nearest Neighbor and Novel approach to Threshold calculation. Results are shown in modules.

Conclusion section describes the algorithms and techniques used in thesis and the applications of this thesis work.

Future Work section describes the modifications and changes that can be done in algorithms and approaches used in this thesis work. It gives details about advancement that can be done in current techniques.

References section lists research papers from conferences, journals etc.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Information Retrieval:-

Information Retrieval (IR) is the science of searching for documents, for information within documents, and for metadata about documents, as well as that of searching relational databases and the World Wide Web. There is overlap in the usage of the terms data retrieval, document retrieval, information retrieval, and text retrieval, but each also has its own body of literature, theory and technologies. IR is interdisciplinary, based on computer science, mathematics, library science, information science, information architecture, cognitive psychology, linguistics, statistics, and physics[**Information Retrieval Wikipedia**].

Web search engines are the most visible IR applications. Web search engines implementation of many features formerly found only in experimental IR systems. Search engines become the most common and maybe best instantiation of IR models, research, and implementation. In this study, search engine approaches for ranking and classify documents are studied and improvements are being done by novel approach for classification.

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

Most IR systems compute a numeric score on how well each object in the database matches the query, and rank the objects according to this value. The top ranking objects are then shown to the user. The process may then be iterated if the user wishes to refine the query. Similarity scores are given to each query after calculating it within corpus of documents. Cosine Similarity is used in this study for this calculation [**David Grossman et al**.].

1.2 Performance Measures in Information Retrieval:-

Many different measures for evaluating the performance of information retrieval systems have been proposed. The measures require a collection of documents and a query. Every document is known to be either relevant or non-relevant to a particular query **[Information Retrieval Wikipedia]**.

1.2.1 Precision:-

Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system.

1.2.2 Recall:-

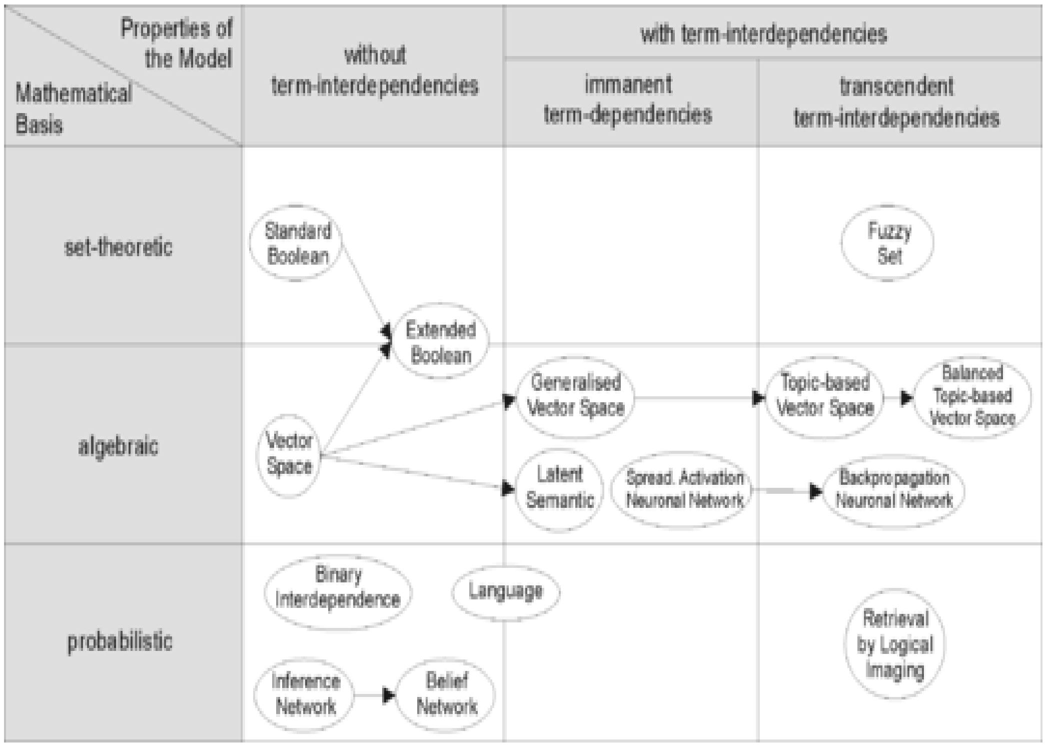
Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

It can be looked at as *the probability that a relevant document is retrieved by the query*.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

1.3 Model types:-



For the information retrieval to be efficient, the documents are typically transformed into a suitable representation. There are several representations. The picture above illustrates the relationship of some common models. In the picture, the models are categorized according to two dimensions: the mathematical basis and the properties of the model. In this thesis, the documents are represented in vector form [G Salton et. al.].

1.3.1 First dimension: mathematical basis:-

- **Set-theoretic models** represent documents as sets of words or phrases. Similarities are usually derived from set-theoretic operations on those sets. Common models are:

  - Standard Boolean model
  - Extended Boolean model
  - Fuzzy retrieval

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

  - Vector space model
  - Generalized vector space model
  - Topic-based vector space model
  - Extended Boolean model
  - Enhanced topic-based vector space model
  - Latent semantic indexing

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

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

- **Machine-learned ranking models** view documents as vectors of ranking features (some of which often incorporate other ranking models mentioned above) and try to find the best way to combine these features into a single relevance score by machine learning methods.

1.3.2 Second dimension: properties of the model:-

- **Models without term-interdependencies** treat different terms/words as independent. This fact is usually represented in vector space models by the orthogonality assumption of term vectors or in probabilistic models by an independency assumption for term variables.
- **Models with immanent term interdependencies** allow a representation of interdependencies between terms. However the degree of the interdependency between two terms is defined by the model itself. It is usually directly or indirectly derived (e.g. by dimensional reduction) from the co-occurrence of those terms in the whole set of documents.
- **Models with transcendent term interdependencies** allow a representation of interdependencies between terms, but they do not allege how the interdependency between two terms is defined. They relay an external source for the degree of interdependency between two terms. (For example a human or sophisticated algorithms.)

In this thesis work, documents are represented in the form of vectors i.e. Vector Space Model which is type of Algebraic model is used. In addition, model without term-interdependencies is used. Vector Space Model gives very good result when model without term-interdependencies is used.

CHAPTER 2

SEARCH ENGINE

2.1 Search Engine (Information Retrieval Application):-

A **web search engine** is designed to search for information on the World Wide Web. The search results are usually presented in a list of results and are commonly called *hits*. The information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

2.2 Search Engine Techniques:-

Search engines have a short list of critical operations that allows them to provide relevant web results when searchers use their system to find information [Christian Platzer et. al.].

1. Crawling the Web
Search engines run automated programs, called "bots" or "spiders" that use the hyperlink structure of the web to "crawl" the pages and documents that make up the World Wide Web. Estimates are that of the approximately 20 billion existing pages, search engines have crawled between 8 and 10 billion.

2. Indexing Documents
Once a page has been crawled, it's contents can be "indexed" - stored in a giant database of documents that makes up a search engine's "index". This index needs to be tightly managed, so that requests which must search and sort billions of documents can be completed in fractions of a second.

3. Processing Queries
When a request for information comes into the search engine (hundreds of millions do each day), the engine retrieves from its index all the documents that match the query.

A match is determined if the terms or phrase is found on the page in the manner specified by the user. For example, a search for car and driver magazine at Google returns 8.25 million results, but a search for the same phrase in quotes ("car and driver magazine") returns only 166 thousand results. In the first system, commonly called "Find all" mode, Google returned all documents which had the terms "car" "driver" and "magazine" (they ignore the term "*and*" because it's not useful to narrowing the results), while in the second search, only those pages with the exact phrase "car and driver magazine" were returned. Other advanced operators (Google has a list of 11) can change which results a search engine will consider a match for a given query.

4. Ranking Results

Once the search engine has determined which results are a match for the query, the engine's algorithm (a mathematical equation commonly used for sorting) runs calculations on each of the results to determine which is most relevant to the given query. They sort these on the results pages in order from most relevant to least so that users can make a choice about which to select.

In this study, documents are ranked based on static ranking i.e. keyword search. First documents are indexed and then using cosine similarity and tf-idf these documents are ranked.

Search engines use automated software programs known as spiders or bots to survey the Web and build their databases. Web documents are retrieved by these programs and analyzed. Data collected from each web page are then added to the search engine index. When you enter a query at a search engine site, your input is checked against the search engine's index of all the web pages it has analyzed. The best documents are then returned to you as hits, ranked in order with the best results at the top. In this study the documents are first index in doc or txt format and then based on the query terms the index is matched and then results are ranked according to similarity score.

2.3 Keyword Search:-

This is the most common form of text search on the Web. Most search engines do their text query and retrieval using keywords.

What is a keyword, exactly? It can simply be any word on a webpage. For example, the word "simply" is used in the previous sentence, making it one of the keywords for this particular webpage in some search engine's index. However, since the word "simply" has nothing to do with the subject of this webpage (i.e., how search engines work), it is not a very useful keyword. Useful keywords and key phrases for this page would be "search," "search engines," "search engine methods," "how search engines work," "ranking" "relevancy," "search engine tutorials," etc. Those keywords would actually tell a user something about the subject and content of this page. TF/IDF (which is base algorithm for ranking in this study) is an example of keyword search. In TF/IDF terms are checked and then the word which has more discriminative power is given more weight [**Juan Ramos**].

Unless the author of the Web document specifies the keywords for his/her document (this is possible by using meta tags), it's up to the search engine to determine them. Essentially, this means that search engines pull out and index words that appear to be significant. Since engines are software programs, not rational human beings, they work according to rules established by their creators for what words are *usually* important in a broad range of documents. The title of a page, for example, usually gives useful information about the subject of the page (if it doesn't, it should!). Words that are mentioned towards the beginning of a document (think of the "topic sentence" in a high school essay, where you lay out the subject you intend to discuss) are given more weight by most search engines. The same goes for words that are repeated several times throughout the document.

Some search engines index every word on every page [Christian Platzer et. al.]. Others index only part of the document. Some of the search engines discriminate upper case from lower case; others store all words without reference to capitalization. In this study only the important words are indexed and stop words and words which are repeated many times are not indexed , upper case words are converted to lower case words for indexing and checking similarity with the query.

2.3.1 The problem with Keyword Search:-

Keyword searches have a tough time distinguishing between words that are spelled the same way, but mean something different (i.e. hard cider, a hard stone, a hard exam, and the hard drive on your computer). This often results in hits that are completely irrelevant to your query. Some search engines also have trouble with so-called stemming -- i.e., if you enter the word "big," should they return a hit on the word, "bigger?" What about singular and plural words? What about verb tenses that differ from the word you entered by only an "s," or an "ed"? [Sergey Brin et al.]. Search engines also cannot return hits on keywords that mean the same, but are not actually entered in your query. A query on heart disease would not return a document that used the word "cardiac" instead of "heart”. This problem can be removed by using an index where these words are similar.

Most sites offer two different types of searches--"basic" and "refined" or "advanced." In a "basic" search, you just enter a keyword without sifting through any pull down menus of additional options. Depending on the engine, though, "basic" searches can be quite complex.

Advanced search refining options differ from one search engine to another, but some of the possibilities include the ability to search on more than one word, to give more weight to one search term than you give to another, and to exclude words that might be likely to muddy the results. You might also be able to search on proper names, on phrases, and on words that are found within a certain proximity to other search terms.

Some search engines also allow you to specify what form you'd like your results to appear in, and whether you wish to restrict your search to certain fields on the internet (i.e., UseNet or the Web) or to specific parts of Web documents (i.e., the title or URL). In this study only basic search is used based on number of terms matching.

Many, but not all search engines allow you to use so-called Boolean operators to refine your search. These are the logical terms AND, OR, NOT, and the so-called proximal locators, NEAR and FOLLOWED BY.

Boolean AND means that all the terms you specify must appear in the documents, i.e., "heart" AND "attack." You might use this if you wanted to exclude common hits that would be irrelevant to your query.

Boolean OR means that at least one of the terms you specify must appear in the documents, i.e., bronchitis, acute OR chronic. You might use this if you didn't want to rule out too much.

Boolean NOT means that at least one of the terms you specify must not appear in the documents. You might use this if you anticipated results that would be totally off-base, i.e., nirvana AND Buddhism, NOT Cobain.

Not quite Boolean + and - Some search engines use the characters + and - instead of Boolean operators to include and exclude terms.

NEAR means that the terms you enter should be within a certain number of words of each other. **FOLLOWED BY** means that one term must directly follow the other. **ADJ**, for adjacent, serves the same function. A search engine that will allow you to search on phrases uses, essentially, the same method (i.e., determining adjacency of keywords).

Phrases: The ability to query on phrases is very important in a search engine. Those that allow it usually require that you enclose the phrase in quotation marks, i.e., "space the final frontier."

Capitalization: This is essential for searching on proper names of people, companies or products. Unfortunately, many words in English are used both as proper and common nouns--Bill, bill, Gates, gates, Oracle, oracle, Lotus, lotus, Digital, digital--the list is endless.

All the search engines have different methods of refining queries. [Sergey Brin et al.].

2.4 Relevancy Rankings:-

Most of the search engines return results with confidence or relevancy rankings. In other words, they list the hits according to how closely they think the results match the query. However often the results may seem completely irrelevant.

It's because search engine technology has not yet reached the point where humans and computers understand each other well enough to communicate clearly.

Most search engines use search term frequency as a primary way of determining whether a document is relevant. If you're researching diabetes and the word "diabetes" appears multiple times in a Web document, it's reasonable to assume that the document will contain useful information. Therefore, a document that repeats the word "diabetes" over and over is likely to turn up near the top of your list [Juan Ramos et al.].

If your keyword is a common one, or if it has multiple other meanings, you could end up with a lot of irrelevant hits. And if your keyword is a subject about which you desire information, you don't need to see it repeated over and over--it's the information *about* that word that you're interested in, not the word itself.

Some search engines consider both the frequency and the positioning of keywords to determine relevancy, reasoning that if the keywords appear early in the document, or in the headers, this increases the likelihood that the document is on target. For example, one method is to rank hits according to how many times your keywords appear and in which fields they appear (i.e., in headers, titles or plain text). Another method is to determine which documents are most frequently linked to other documents on the Web. In this study term frequency method is used and modified and then used it for classification.

If you use the advanced query form on AltaVista, you can assign relevance weights to your query terms before conducting a search. Although this takes some practice, it essentially allows you to have a stronger say in what results you will get back.

As far as the user is concerned, relevancy ranking is critical, and becomes more so as the sheer volume of information on the Web grows. Most of us don't have the time to sift through scores of hits to determine which hyperlinks we should actually explore.

The more clearly relevant the results are, the more we're likely to value the search engine.

2.5 Information on Meta Tags:-

Some search engines are now indexing Web documents by the meta tags in the documents' HTML (at the beginning of the document in the so-called "head" tag). What this means is that the Web page author can have some influence over which keywords are used to index the document, and even in the description of the document that appears when it comes up as a search engine hit [**Sergey Brin et al**.].

There is no perfect way to ensure that the document receive a high ranking. Even if the document gets a great ranking, there's no assurance that it can keep it for long. For example, at one period a document from the Spider's Apprentice was the number- one-ranked result on AltaVista for the phrase "how search engines work." And after some time this document’s relevance became less.

There is a lot of conflicting information out there on meta-tagging. Some search engines rely heavily on meta tags, others don't use them at all. The general opinion seems to be that meta tags are less useful than they were a few years ago, largely because of the high rate of spamdexing (web authors using false and misleading keywords in the meta tags).

It seems to be generally agreed that the "title" and the "description" meta tags are important to write effectively, since several major search engines use them in their indices. Use relevant keywords in your title, and vary the titles on the different pages that make up corpus of documents, in order to target as many keywords as possible. As for the "description" meta tag, some search engines will use it as their short summary of your url, so make sure the document’s description is one that will entice surfers to corpus of documents.

Many search engine algorithms score the words that appear towards the top of your document more highly than the words that appear towards the bottom. Words that appear in HTML header tags (H1, H2, H3, etc) are also given more weight by some search engines. It sometimes helps to give the document a file name that makes use of one of your prime keywords, and to include keywords in the "alt" image tags.

Different search engines have different policies for meta tagging. It is mentioned in help files about these policies.

2.6 Concept-based search:-

Excite used to be the best-known general-purpose search engine site on the Web that relies on concept-based searching. It is now effectively extinct.

Unlike keyword search systems, concept-based search systems try to determine what you mean, not just what you say. In the best circumstances, a concept-based search returns hits on documents that are "about" the subject/theme you're exploring, even if the words in the document don't precisely match the words you enter into the query.

There are various methods of building clustering systems, some of which are highly complex, relying on sophisticated linguistic and artificial intelligence theory. Excite used to a numerical approach. Excite's software determines meaning by calculating the frequency with which certain important words appear. When several words or phrases that are tagged to signal a particular concept appear close to each other in a text, the search engine concludes, by statistical analysis that the piece is "about" a certain subject.

2.7 Search Engine Ranking Algorithms:-

2.7.1 Link Based Algorithm:-

Search engine ranking algorithms are closely guarded secrets, for at least two reasons: search engine companies want to protect their methods from their competitors, and they also want to make it difficult for web site owners to manipulate their rankings.

Document relevance ranking for a specific query depends on three factors:

- Its relevance to the words and concepts in the query

- Its overall link popularity
- Whether or not it is being penalized for excessive search engine optimization (SEO).

Examples of SEO abuse would be a lot of sites linked to each other in a circular scam, or excessive and highly ungrammatical stuffing with keywords.

Factor #2 was innovated by Google with PageRank. Essentially, the more incoming links your page has, the better. But it is more complicated than that: indeed, PageRank is a tricky concept because it is circular, as follows: Every page on the Internet has a minimum PageRank score just for existing. 85% (at least, that's the best known estimate, based on an early paper) of this PageRank is passed along to the pages that page links to, divided more or less equally along its outgoing links. A page's PageRank is the sum of the minimum value plus all the PageRank passed to it via incoming links [Sergey Brin et al.].

Although this is circular, mathematical algorithms exist for calculating it iteratively. Google actually reports PageRank scores of 0 to 10 that are believed to be based on the logarithm of raw PageRank. And the base of that logarithm is believed to be approximately 6.

Anyhow, there are about 30 sites on the Web of PageRank10, including Yahoo, Google, Microsoft, Intel, and NASA. IBM, AOL, and CNN, by way of contrast, were only at PageRank 9 as of early in 2004.

Further refinements in link popularity rankings are under development. Notably, link popularity can be made specific to a subject or category; i.e., pages can have different PageRanks for health vs. sports vs. computers vs. whatever. Supposedly, AskJeeves/Teoma already works that way. In this study also the concept like this is analysed but in keyword search based ranking. The documents are classified and ranked individually for each category [Alok Ranjan et al.].More on document classification is explained in Chapter 5.

It is believed that Inktomi, AltaVista, et al. use link popularity in their ranking algorithms, but to a much lesser extent than Google. Yahoo, owner of Inktomi, AltaVista, Althaea, is rolling out a new search engine, which reportedly includes a feature called Web Rank.

2.7.2 Keyword Search Algorithm:-

Most search engines handle words and simple phrases. In its simplest form, text search looks for pages with lots of occurrences of each of the words in a query, stop words aside. The more common a word is on a page, compared with its frequency in the overall language, the more likely that page will appear among the search results. Hitting all the words in a query is a lot better than missing some [**G.Salton et al.**].

Search engines also make some efforts to "understand" what is meant by the query words. For example, most search engines now offer optional spelling correction. And increasingly they search not just on the words and phrases actually entered, but the also use stemming to search for alternate forms of the words (e.g., speak, speaker, speaking, spoke). Teoma-based engines are also offering refinement by category. However, Excite-like concept search has otherwise not made a comeback yet, since the concept categories are too unstable.

When ranking results, search engines give special weight to keywords that appear:

- High up on the page
- In headings
- In BOLDFACE (at least in Inktomi)
- In the URL
- In the title (important)
- In the description
- In the ALT tags for graphics.
- In the generic keywords meta tags (only for Inktomi, and only a little bit even for them)
- In the link text for inbound links.

More weight is put on the factors that can not be faked for misuse or some website’s benefit, such as inbound link text, page title (which shows up on the SERP -- Search Engine Results Page), and description.

CHAPTER 3

TERM FREQUENCY / INVERSE DOCUMENT FREQUENCY

3.1 Basics of Model:-

Term Frequency/Inverse Document Frequency model is based on the principle that if a term occurs more within a document(TF) and rare within the corpus of documents(IDF), then that term would be having high discriminative power to distinguish between relevant and non-relevant documents[**Juan Ramos et al**.]. TF/IDF is a type of keyword search based algorithm as explained in Chapter 2.The document having high TF/IDF value is having strong relationship with the query and would be more relevant for the user.

Inverse Document Frequency (IDF) is based on the fact that a term which occurs in many documents is not a good discriminator and should be given less weight than one which occurs in few documents [**Stephen Robertson**].

Let there are N documents in the collection, and that term t_i occurs in n_i of them. IDF is calculated as:-

$$\text{idf}\ (t_i) = \log \frac{N}{\text{ni}}$$

Term Frequency/Inverse Document Frequency model incorporates local and global information. Encoding TF/IDF is simple. Term Weight is calculates as:-

$$w_i = tf_i * log\left(\frac{D}{df_i}\right)$$

where

- tf_i = *term frequency (term counts) or number of times a term i occurs in a document. This accounts for local information.*
- df_i = *document frequency or number of documents containing term i*
- D = *number of documents in a database.*

the df_i / D ratio is the probability of selecting a document containing a queried term from a collection of documents. This can be viewed as a global probability over the entire collection. Thus, the $log(D/df_i)$ term is the *inverse document frequency,* IDF_i and accounts for global information.

The **tf–idf** weight (term frequency–inverse document frequency) is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model. Many search engines use combination of keyword search based algorithm e.g. tf-idf based ranking and link based algorithm e.g. PageRank based ranking [Sergey Brin et al.].

3.2 Term Weight calculation Example:-

Suppose there is a set of English text documents and user wants to determine which document is most relevant to the query "the brown cow." A simple way to start out is by eliminating documents that do not contain all three words "the," "brown," and "cow," but this still leaves many documents. To further distinguish them total number of terms in a document is counted and summed together; the number of times a term occurs in a document is called its *term frequency*. However, because the term "the" is so common, this will tend to incorrectly emphasize documents which happen to use the word "the" more, without giving enough weight to the more meaningful terms "brown" and "cow". Also the term "the" is not a good keyword to distinguish relevant and non-relevant documents and terms like "brown" and "cow" that occur rarely are good keywords to distinguish relevant documents from the non-relevant documents. Hence an *inverse document frequency* factor is incorporated which diminishes the weight of terms that occur very frequently in the collection and increases the weight of terms that occur rarely.

Consider a document containing 100 words wherein the word cow appears 3 times. Following the previously defined formulas, the term frequency (TF) for cow is then 0.03 (3 / 100). Now, assume we have 10 million documents and cow appears in one thousand of these. Then, the inverse document frequency is calculated as ln (10000 / 1 000) = 9.21. The TF-IDF score is the product of these quantities: 0.03 * 9.21 = 0.28

The results of applying Term Frequency-Inverse Document Frequency (TF-IDF) to determine what words in a corpus of documents might be more favourable to use in a query can be calculated with this model. As the term implies, TF-IDF calculates values for each word in a document through an inverse proportion of the frequency of the word in a particular document to the percentage of documents the word appears in. Words with high TF-IDF numbers imply a strong relationship with the document they appear in, suggesting that if that word were to appear in a query, the document could be of interest to the user. This algorithm efficiently categorizes relevant words that can enhance query retrieval.

In this study, tf-idf is used as the base technique in document ranking and classification. In Chapter 4 use of tf-idf with cosine similarity will be explained in detail. In document classification (explained in Chapter 5), there is requirement of distance measure between a document and predefined class label. Cosine Similarity is used as the distance measure and then documents are classified based on Nearest Neighbor Algorithm [**Gang Qian et al**.].

CHAPTER 4

VECTOR SPACE MODEL

4.1 Basics of Model:-

Vector space model (or term vector model) is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings. Its first use was in the SMART Information Retrieval System [David Grossman].

4.2 Document Representation:-

In order to reduce the complexity of the documents and make them easier to handle, the document have to be transformed from the full text version to a document vector which describes the contents of the document. A definition of a document is that it is made of a joint membership of terms which have various patterns of occurrence. There can be many representations for a document but in this thesis tf-idf is used as indexing/ranking and tf-idf works best with vector space model.

Term frequency is content descriptive features for documents .Documents and queries are represented as vectors.

$$d_j = (w_{1,j},w_{2,j},...,w_{t,j})$$
$$q = (w_{1,q},w_{2,q},...,w_{t,q})$$

Each dimension corresponds to a separate term. If a term occurs in the document, its value in the vector is non-zero. Several different ways of computing these values, also known as (term) weights, have been developed. In this thesis term weights are calculated as tf-idf weights [**Juan Ramos et al**.].

The definition of term depends on the application. Typically terms are single words, keywords, or longer phrases. If the words are chosen to be the terms, the dimensionality of the vector is the number of words in the vocabulary (the number of distinct words occurring in the corpus).In this thesis the words are chosen to be single words but not stopwords or frequently occurring words. Vector operations like dot product, determinant i.e. length of vector, angle between vectors (in this thesis cosine angle is used) can be used to compare documents with queries.

4.3 Use of Tf-idf weight in Vector Space Model:-

In vector space model the term specific weights in the document vectors are products of local and global parameters [G. Salton et al.]. The model is known as term frequency-inverse document frequency model. The weight vector for document *d* is $\mathbf{v}_d = [w_{1,d}, w_{2,d}, \ldots, w_{N,d}]^T$, where

$$w_{t,d} = \mathrm{tf}_t \cdot \log \frac{|D|}{|\{t \in d\}|}$$

and

- tf_t is term frequency of term *t* in document *d* (a local parameter)
- $\log \frac{|D|}{|\{t \in d\}|}$ is inverse document frequency (a global parameter). $|D|$ is the total number of documents in the document set; $|\{t \in d\}|$ is the number of documents containing the term *t*.

Using the cosine the similarity between document d_j and query *q* can be calculated as:

$$sim(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{t} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} * \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

In a simpler Term Count Model the term specific weights do not include the global parameter. Instead the weights are just the counts of term occurrences: $w_{t,d} = \mathrm{tf}_t$.

4.3.1 Document Indexing

It is obvious that many of the words in a document do not describe the content, words like *the, is*. By using automatic document indexing those non significant words (function words) are removed from the document vector, so the document will only be represented by content bearing words. This indexing can be based on term frequency, where terms that have both high and low frequency within a document are considered to be function words. In practice, term frequency has been difficult to implement in automatic indexing. Instead the use of a stop list which holds common words to remove high frequency words (stop words), which makes the indexing method language dependent. In general, 40-50% of the total numbers of words in a document are removed with the help of a stop list. In this thesis this stopwords list are indexed in a file and then terms of original document are matched with these stopwords using java.

Non linguistic methods for indexing have also been implemented. Probabilistic indexing is based on the assumption that there is some statistical difference in the distribution of content bearing words, and function words. Probabilistic indexing ranks the terms in the collection w.r.t. the term frequency in the whole collection. The function words are modeled by a Poisson distribution over all documents, as content bearing terms cannot be modeled. The use of Poisson model is expanded to Bernoulli model. Recently, an automatic indexing method which uses serial clustering of words in text has been introduced. The value of such clustering is an indicator if the word is content bearing [**Bader Aljaber et al**.].

4.3.2 Term Weighting

Term weighting has been explained by controlling the exhaustivity and specificity of the search, where the exhaustivity is related to recall and specificity to precision. The term weighting for the vector space model has entirely been based on single term statistics. There are three main factors term weighting: term frequency factor, collection frequency factor and length normalization factor. These three factor are multiplied together to make the resulting term weight.

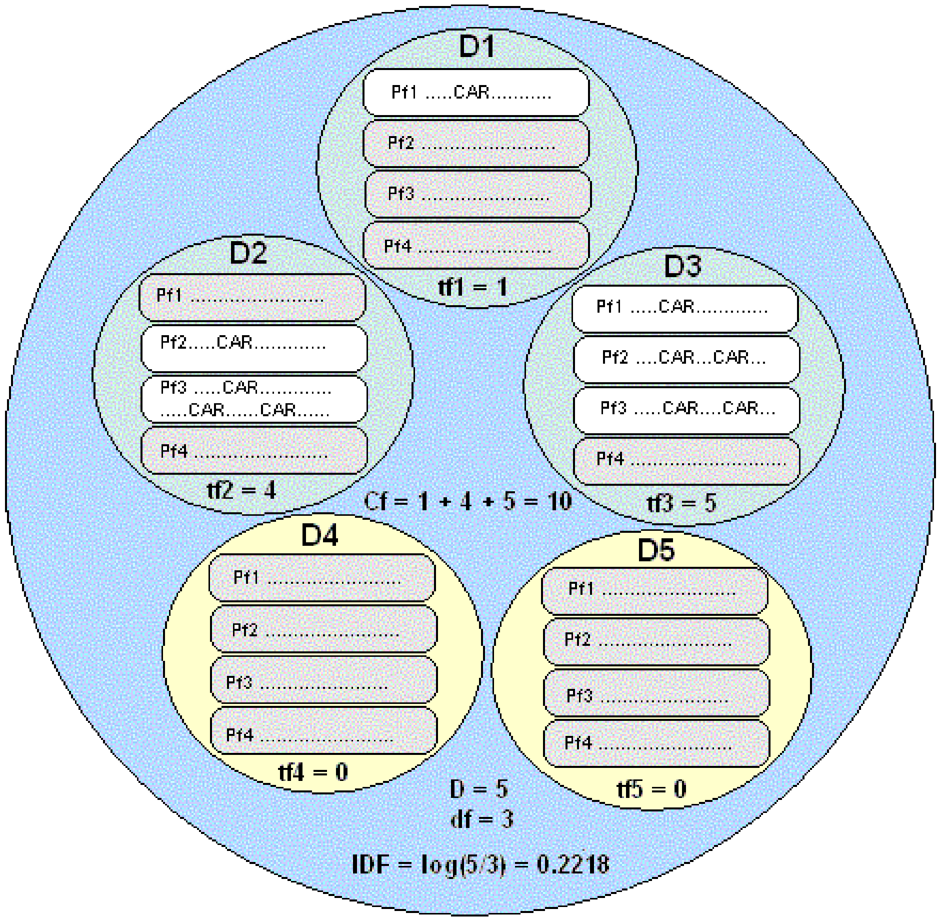
A common weighting scheme for terms within a document is to use the frequency of occurrence. The term frequency is somewhat content descriptive for the documents and is generally used as the basis of a weighted document vector. It is also possible to use binary document vector, but the results have not been as good compared to term frequency when using the vector space model [**David Grossman**].

There are used various weighting schemes to discriminate one document from the other. In general this factor is called collection frequency document. Most of them, e.g. the inverse document frequency, assume that the importance of a term is proportional with the number of document the term appears in. Experimentally it has been shown that these document discrimination factors lead to a more effective retrieval, i.e., an improvement in precision and recall.

The third possible weighting factor is a document length normalization factor. Long documents have usually a much larger term set than short documents, which makes long documents more likely to be retrieved than short documents. This problem is removed in this thesis by normalizing the vectors. So, longer documents do not get unfair advantage than shorter documents in classification and ranking [**Christian Platzer et al.**]

4.3.3 Similarity Coefficients

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


D1
Pf1 .....CAR..........
Pf2 ....
Pf3 ....
Pf4 ....
tf1 = 1
D2
Pf1 ....
Pf2.....CAR............
Pf3 .....CAR............ .....CAR......CAR......
Pf4 ....
tf2 = 4
D3
Pf1 .....CAR............
Pf2 ....CAR...CAR...
Pf3 .....CAR....CAR...
Pf4 ....
tf3 = 5
Cf = 1 + 4 + 5 = 10
D4
Pf1 ....
Pf2 ....
Pf3 ....
Pf4 ....
tf4 = 0
D5
Pf1 ....
Pf2 ....
Pf3 ....
Pf4 ....
tf5 = 0
D = 5
df = 3
IDF = log(5/3) = 0.2218

Distribution of Term "CAR" across Collection, Documents, Passages and Sentences

white = Passages with term
gray = Passages without term
green = Documents with term
yellow = Documents without term
blue = Documents Collection

Copyright 2004 Dr. E. Garcia

4.4 Vector Space Example

4.4.1 Assumptions:-

To understand term weight calculation, a trivial example is used. In this thesis following assumptions are used:-

1. do not take into account WHERE the terms occur in documents.
2. use all terms, including very common terms and stop words.
3. do not reduce terms to root terms (stemming).

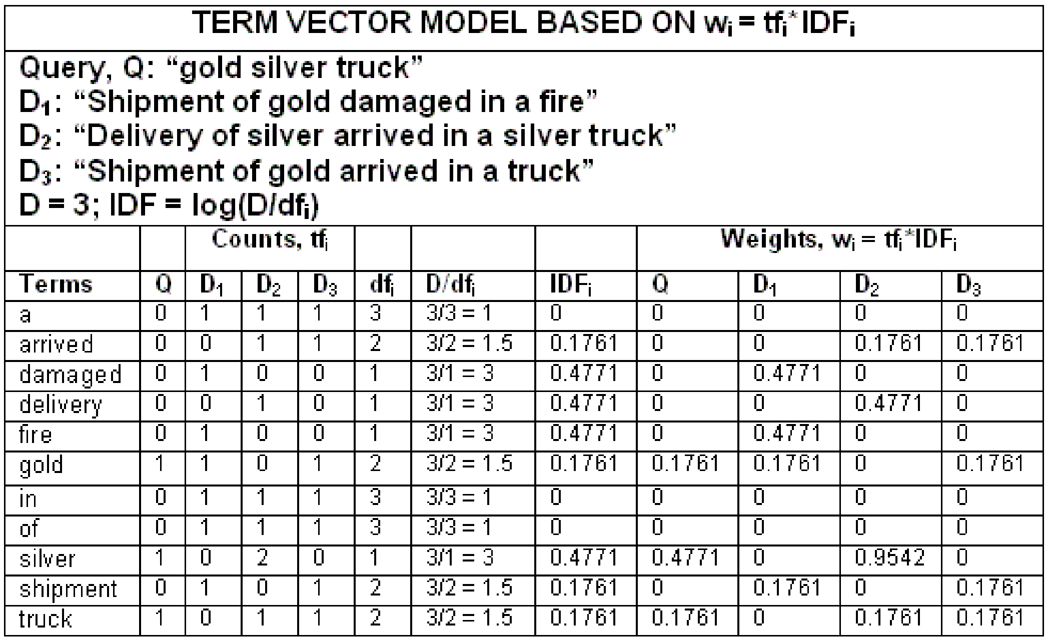
4. use raw frequencies for terms and queries (unnormalized data).

Suppose the query is "gold silver truck". The database collection consists of three documents (D = 3) with the following content

D1: "Shipment of gold damaged in a fire"
D2: "Delivery of silver arrived in a silver truck"
D3: "Shipment of gold arrived in a truck"

4.4.2 Results:-

Retrieval results are summarized in the following table.



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

The tabular data is based on Dr. Grossman's example [**David Grossman**]. The last four columns are added to illustrate all term weight calculations. Let's analyze the raw data, column by column.

1. Columns 1 - 5: First, we construct an index of terms from the documents and determine the term counts tf_i for the query and each document D_j.

2. Columns 6 - 8: Second, we compute the document frequency d_i for each document. Since $IDF_i = \log(D/df_i)$ and D = 3, this calculation is straightforward.
3. Columns 9 - 12: Third, we take the tf*IDF products and compute the term weights. These columns can be viewed as a sparse matrix in which most entries are zero.

Now weights are treated as coordinates in the vector space, effectively representing documents and the query as vectors. Similarity analysis is done to check which document is closer to query.

4.4.3 Similarity Analysis:-

First for each document and query, vector lengths are computed (zero terms ignored)

$$|D_1| = \sqrt{0.4771^2 + 0.4771^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.5173} = 0.7192$$

$$|D_2| = \sqrt{0.1761^2 + 0.4771^2 + 0.9542^2 + 0.1761^2} = \sqrt{1.2001} = 1.0955$$

$$|D_3| = \sqrt{0.1761^2 + 0.1761^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.1240} = 0.3522$$

$$\therefore\ |D_i| = \sqrt{\sum_i w_{i,j}^2}$$

$$|Q| = \sqrt{0.1761^2 + 0.4771^2 + 0.1761^2} = \sqrt{0.2896} = 0.5382$$

$$\therefore\ |Q| = \sqrt{\sum_i w_{Q,j}^2}$$

Next, all dot products are computed (zero products ignored)

$$Q \bullet D_1 = 0.1761 * 0.1761 = 0.0310$$
$$Q \bullet D_2 = 0.4771 * 0.9542 + 0.1761 * 0.1761 = 0.4862$$
$$Q \bullet D_3 = 0.1761 * 0.1761 + 0.1761 * 0.1761 = 0.0620$$

$$\therefore\ Q \bullet D_i = \sum_i w_{Q,j} w_{i,j}$$

Now similarity values are calculated. This is tf*idf normalization, so that longer documents are not unfairly given more weight. Normalization makes the range from 0 to 1.

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

$$\therefore\ \text{Cosine } \theta_{D_i} = \text{Sim}(Q, D_i)$$

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

Finally we sort and rank the documents in descending order according to the similarity values

Rank 1: Doc 2 = 0.8246

Rank 2: Doc 3 = 0.3271

Rank 3: Doc 1 = 0.0801

4.4.4 Observations:-

This example illustrates several facts. Frequent terms such as "a", "in", and "of" tend to receive a low weight -a value of zero in this case. Thus, the model correctly predicts that very common terms, occurring in many documents in a collection are not good discriminators of relevancy. Note that this reasoning is based on global information; i.e., the IDF term. Precisely, this is why this model is better than the term count model discussed in Chapter 1. Instead of calculating individual vector lengths and dot products, computational time can be saved by applying directly the similarity function:- -

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

Individual values of tf and idf have to be calculated. By using TF-IDF Model with the use of Vector Space Model, the rank of documents can be calculated more accurately than Probabilistic Model. IDF (Inverted Document Frequency) value i.e. global information is added to TF (Term Frequency) i.e. local information, because in simple tf value, longer documents get unfair higher rank than shorter documents and the word which comes more times in corpus of documents get higher weights due to term frequency weights, but with the use of idf value, the term which occurs in many documents has less discriminative power of relevant and non- relevant document. With the use of tf*idf normalization, the longer documents can not get unfair higher rank than shorter documents [**M.S.Xiaoying Tai et al**.].

CHAPTER 5

DOCUMENT CLASSIFICATION

5.1 Document Classification:-

Documents Classification can be defined as categorizing the documents in different classes. The web documents available on World Wide Web are not in structured pattern and the documents which are in companies' intranet are also unstructured. The documents have also grown very much in size. So to retrieve the information from these documents effectively and efficiently there is requirement of categorizing these documents. There can be different categories like Education, Business, Sports, Health, Entertainment, Technology, Politics and International [**S.Suseela**].

Document Classification (also called Text Categorization) can be stated as:-
Given a set of document class C and example documents for each class, construct a classifier which, given a document d, finds the class to which d is most similar. There are two variants of Text Classification- Text clustering and Text Categorization. The former is concerned with finding a latent group structure in the set of documents, while the latter (also known as text classification) can be seen as the task of structuring the repository of documents according to a group structure that is known in advance.

Document Classification is distinguished into binary, multi-class and multi-label settings. In the binary setting there are exactly two classes- relevant and non relevant or spam and non spam. In multi-class setting there can be more than two classes, for example in e-mail routing the mail should be routed to the appropriate class or department in case of intranet within an organization. Document can be labelled with exactly one class out of possible k classes. In multi-label setting, one document can belong to many, exactly one or no category at all [**Davide Magatti et al**.]. General Text Parser (GTP) is used mostly for document classification in Vector Space Model. GTP supports local and global weighting and document frequencies, document normalization schemes, keyword lengths and vector space decomposition schemes [**Barry Britt**].

5.2 Applications of Document Classification:-

By using classification search space becomes very small. For example if a user has to search in business domain then the user can directly search in that domain. There is no need to search the entire document corpus. Classifying the documents helps the users find the information quicker and their search is in appropriate direction.
The following are the applications of Document Classification:-

(1) E-mail filtering: - E-mails can be filtered to spam or to a different category. E-mails to weed out spam, or to categorize them into different classes, are just become available now. E-mail systems which typically provide rule-based methods for routing messages to specific mailboxes. The e- mails are routed based either on the sender's name or the occurrence of specific words in the subject line. For example, the occurrence of the word "remove" would cause the sender's name to be dropped from an e-mail list.

(2) Mail routing:-Large enterprises are currently automating their document processing by means of Work-Flow Management systems, allowing an image of the document to circulate through the company rather than it’s original. In particular, they aim for a uniform treatment of incoming mail, whether it is electronic or in paper form. A bottleneck in this approach is the entering of documents into the right work flow. This process involves a superficial human interpretation of the contents of the document, which is time consuming and error prone.

(3) News Monitoring:-In knowledge-based companies like the stock exchanges, a number of persons are concerned with the scanning of newspapers and other information sources for items which are concerned with the national or international economy, or with individual companies on the stock market. The results are sent to the person who should be informed. Can this alert service be automated?

(4) Narrowcasting:-Press agencies strive to give more and more individual service, where each client obtains out of the large stream of outgoing news items only those that are relevant to him, according to his profile.

(5) Content classification: - Large information brokers (such as the European Patent Office) have traditionally used pre-classification of documents as an aid in document disclosure. Documents are manually given a place within a large semantical hierarchy, or index terms according to a given thesaurus. This process is costly and error prone and changes in the thesaurus are hard to accommodate. Modern search machines on the Web use an automatic pre-classification of web pages.

5.3 Classification Algorithms:-

Document Classification Algorithms can be classified into agglomerative or partitioning algorithms and hierarchical or non-hierarchical algorithms [**Michael Steinbach et al.**].

The following are the main algorithms used for classification:-

(1) **Nearest Neighbor Algorithm**: - Nearest Neighbor Algorithm is based on the distance function between the predefined classes and testing document. Distance can be measured in form of Euclidean distance or cosine distance. In Text Mining, Cosine similarity is calculated as it gives good distance measurement of documents.

(2) **Naive Bayesian Algorithm**: - NB algorithm has been widely used for document classification, and shown to produce very good performance. The basic idea is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. NB algorithm computes the posterior probability that the document belongs to different classes and assigns it to the class with the highest posterior probability. The posterior probability of class is computed using Bayes rule and the testing sample is assigned to the class with the highest posterior probability. The naive part of NB algorithm is the assumption of word independence that the conditional probability of a word given a category is assumed to be independent from the conditional probabilities of other words given that category. There are two

versions of NB algorithm. One is the multi-variate Bernoulli event model that only takes into account the presence or absence of a particular term, so it doesn't capture the number of occurrence of each word. The other model is the multinomial model that captures the word frequency information in documents.

5.4 Distance Measures:-

Distance measure is an important part in Document Classification using Vector Model. The reason for turning documents into vectors is that documents that are close together in the vector space, talk about the same thing and then the nearest neighbor of a vector should be having the same class. In Nearest Neighbor Algorithm, two distance measures are used – Euclidean Distance (EUD) and Cosine Angle Distance (CAD). These two distance measures have their own advantages and disadvantages. In text mining, Cosine Angle Distance (CAD) performs better because in calculation of CAD, text documents can be represented in the form of vectors and distance is naturally normalized. In Content Based Image Retrieval (CBIR), Euclidean Distance performs better [Gang Qian et al.].

In high dimensional data spaces, the NN query results by EUD and CAD are similar. For clustered data the results by both distance measures are also same.

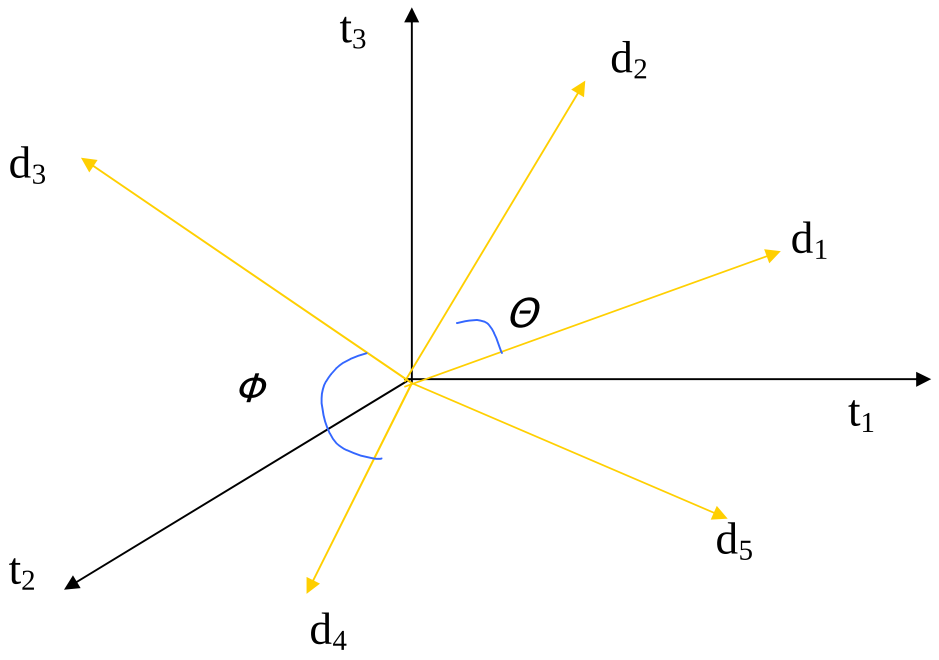


Fig. Documents in Vector Space Model

In Cosine Angle Distance (CAD), calculation in Nearest Neighbor algorithm of Document Classification, nearest neighbor are the documents which are having some angle which is less than predefined threshold. If the value of angle is less, the similarity between documents is high. For Example in the above figure of Vector Space Model document pair- d1 and d2 are having more similarity than document pair – d3 and d4.Euclidean Distance (EUD) is calculated as $|D1 - D2|$.But the problem in calculation of EUD is that there is no normalization used in EUD. So, longer documents get higher similarity score, but in CAD, normalization is used and similarity score is calculated based on cosine similarity score i.e. how many terms are matching between documents and not on the basis of length of documents. CAD calculates similarity and EUD calculates distance [S.Suseela].

5.5 Performance Measures:-

Entropy:-

Entropy determines the quality of classification of a document within a particular class label. If the documents which are classified in same class are most similar then the entropy is 0. So, if the value of entropy is less it means that the quality of classification is good and if the value of entropy is high it means the quality of classification is not good.

Given a cluster S_r of size $N_{r,}$ the entropy of this cluster is defined to be N

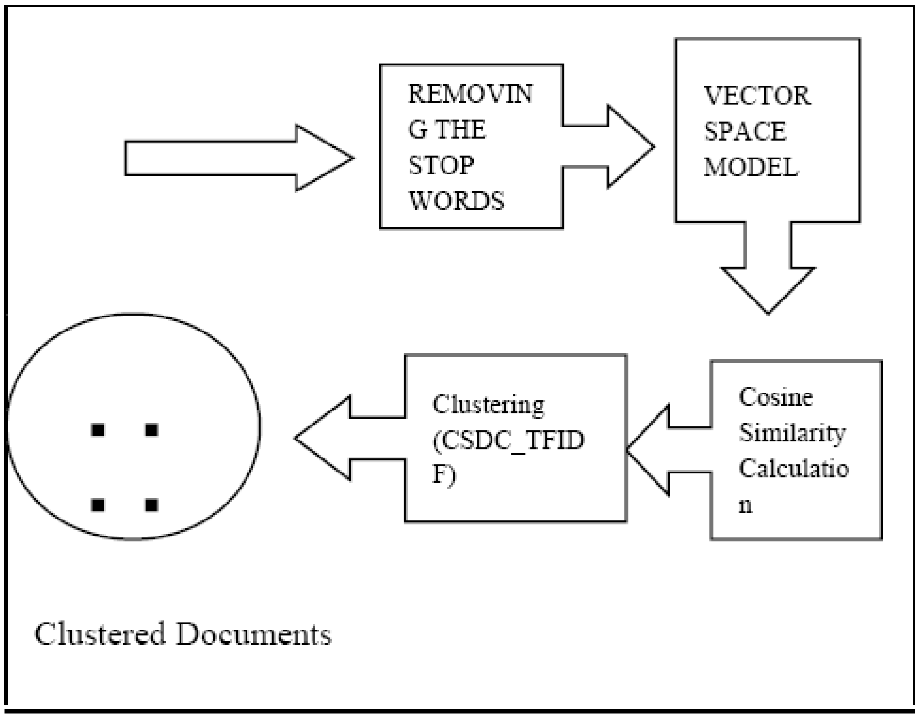
$$E(S_r) = -\frac{1}{\log q}\sum_{i}^{q}\frac{\mathrm{Nri}}{\mathrm{Nr}}\log\left(\frac{Nri}{Nr}\right)$$

where q is the number of classes and Nri is the number of documents belonging to the i_{th} class that were assigned to the r_{th} cluster [Alok Ranjan et al.]. The total entropy is given by

$$\text{Entropy} = \sum_{r=1}^{k} \frac{Nr}{N} E(Sr)$$

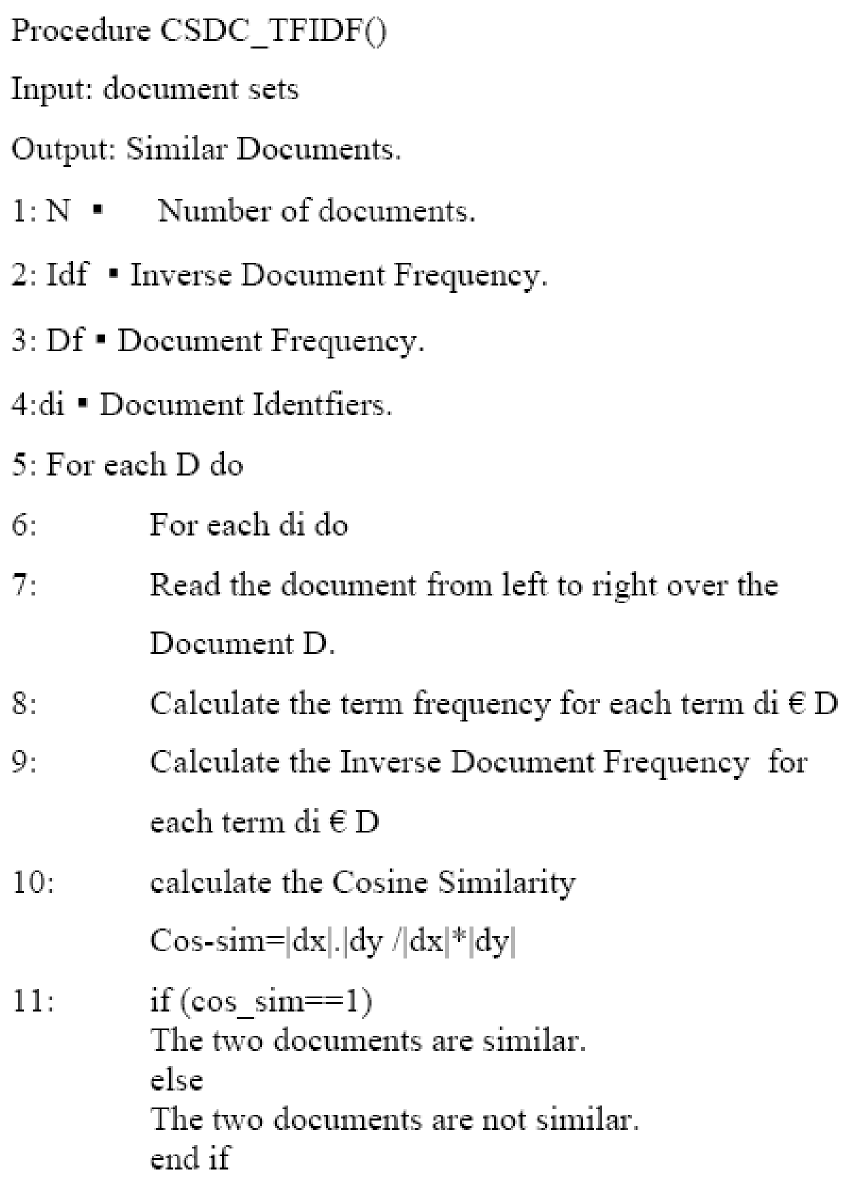
5.6 Threshold Calculation:-

Documents are clustered or classified based on cosine similarity distance measure and using Term Frequency and Inverse Document Frequency using Vector Space Model. This algorithm is a type of HAC (Hierarchical Agglomerative Clustering) Algorithm. A clustering approach CS_TFIDF is used and implemented.



Before Document clustering all non word tokens are removed and then the text is parsed into words and then the stop words are removed. Now using the Vector Space Model documents are represented in the form of vectors. Term frequency (TF) and Inverse Document Frequency of all documents are calculated. Classes are predefined like sports, business, politics and education etc. Cosine Similarity between test document and class labels is calculated. The document and class label combination having high similarity is the right most combination and then that test document is assigned to that class.

In text mining, cosine similarity for calculating the distance between document and class label is very much efficient. Other distance measures such as Euclidean distance can also be used for distance measure. But Euclidean distance is used in other information retrieval techniques and cosine similarity can be used effectively and efficiently in text mining [S.Suseela].



```
Procedure CSDC_TFIDF()
Input: document sets
Output: Similar Documents.
1: N ▪     Number of documents.
2: Idf ▪ Inverse Document Frequency.
3: Df ▪ Document Frequency.
4:di ▪ Document Identfiers.
5: For each D do
6:         For each di do
7:         Read the document from left to right over the
           Document D.
8:         Calculate the term frequency for each term di € D
9:         Calculate the Inverse Document Frequency  for
           each term di € D
10:        calculate the Cosine Similarity
           $Cos\text{-}sim=|dx|.|dy /|dx|*|dy|$
11:        if (cos_sim==1)
           The two documents are similar.
           else
           The two documents are not similar.
           end if
```

Vector Space Model is very important model in information retrieval tasks. Using TF/IDF and cosine similarity, the class of a document can be easily determined. Score is given to document-class label pair after calculating the similarity. Threshold value is defined on score for effective classification.

Three threshold techniques are used in Nearest Neighbor Algorithm [Yiming Yang]:-

(a) R-Cut (Rank Based Threshold):-

Let m be the number of classes and n be the number of test documents. In R-Cut (Rank Based Threshold) algorithm, Classes are sorted based on the document-class label score. YES is assigned to top t-classes where t can be between 1 and m.

(b) P-Cut (Proportion Based Assignments):-

For each category (c_j), sort the test documents by score and assign a YES decision to each of the k_j top-ranking documents where $k_j = P(c_j) * x *m$ is the number of documents assigned to category c_j and $P(c_j)$ is the prior probability for an arbitrary document to be an member of category c_j. x can be any real-valued number specified by the user.

(c) S-Cut (Score Based Local Optimization):-

In this threshold algorithm, per-category threshold is applied .S-Cut optimizes the performance of the classifier on individual categories without guaranteeing a global optimum. This algorithm is applied to various classifiers like kNN, LLSF and Rocchio.

The choice of threshold technique can significantly influence the performance of Nearest Neighbor Algorithm. S-Cut algorithm is used mostly in Nearest Neighbor clustering. S-Cut defines some threshold on the document-class label pair scores. A document d may be classified into category c only if its degree of acceptance (DOA) with respect to c is higher than the threshold of c. But the problem with this threshold value is that sometimes it becomes too high that many results are rejected in document-class label pair and sometimes it becomes too low that false alarm exists in the document-class label pair. So finding an accurate threshold value is difficult and is an active area of research [**Rey-Long Liu et al**.].

CHAPTER 6

PROPOSED TECHNIQUE

6.1 Previous Work:-

In previous Document classification studies the main focus was on improving the classifier rather than on thresholding process. The studies were on improving the classifier to distinguish between relevant document and non-relevant document. More work is being done for pre processing i.e. checking the documents for relevant or non-relevant. Calculating appropriate threshold value is post processing task and is good deciding factor for classification purpose [**James G.Shanahan**]. These methods are not suitable for thresholding since the document which can not be classified properly often can not have reliable DOA (Degree of Acceptance). Calculation of Degree of Acceptance is a challenging task. It should be parameter free and adaptive [**Rey-Long Liu et al**.].R-Cut, P-Cut and S-Cut algorithms were used in previous studies, but none of these algorithms could be able to give parameter free and adaptive threshold value [**Yiming Yang**].

6.2 Proposed Technique:-

In proposed technique, the dependency of threshold calculation on numeric value is removed. Threshold value is not based on numeric scores .Numeric value leads to wrong calculation of threshold value. So if there is knowledge that how many terms are to be matched in document-class label pair then it can be deciding factor for setting the threshold value. Threshold value is decided by the matching of number of terms.

Number of terms to be matched can be decided based on the requirement. For example if there is requirement that five terms should be matched between testing document and class label document then these numbers of terms are threshold value for similarity score. But if five terms to be matched then even two documents which are having 5 terms matched with predefined document class pair, would have different similarity score. So in this technique it is multiplied by that testing document and class label so that uniform threshold value is calculated. So from using uniform threshold the problem of varying length of documents is removed.

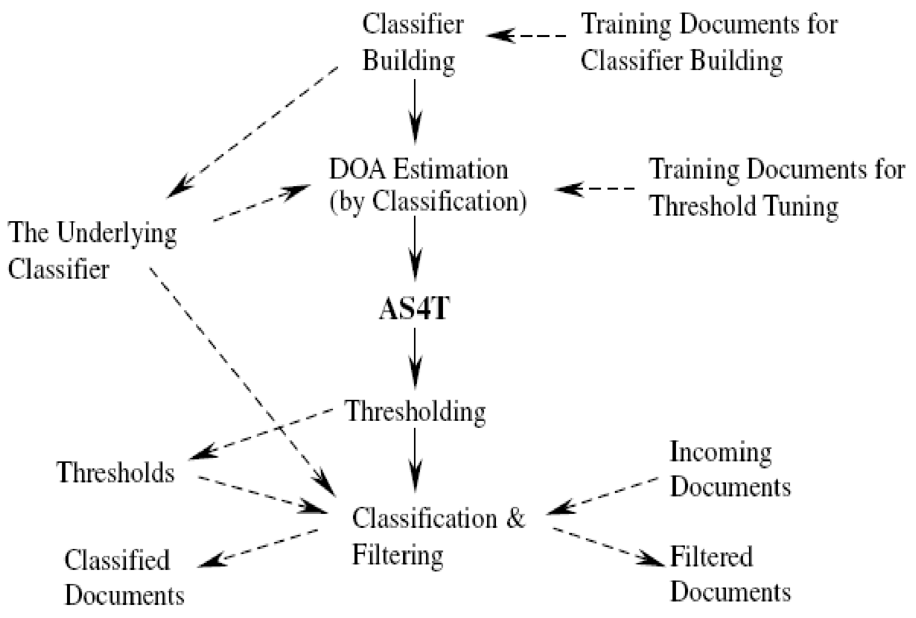
Cosine Similarity is calculated as:-

$$\text{Sim}_{i,\,j} = \frac{Di.Dj}{|Di| * |Dj|}$$

After calculating document-class label pair similarity score, even if in two different documents the same numbers of terms are matching with the class label ,then also their similarity score would be different due to different lengths of documents. To avoid this, score is multiplied by the length of that document and length of class label. After this normalization, score would be calculated based on only similarity of the number of terms not depending upon the length of documents.

Now threshold value can be calculated given the number of terms to be matched within a category. This is better approach than used in simple S-Cut which is based only on numeric score [**Yiming Yang**]. After threshold value is known of each category based on number of terms, rank of documents based on score can be calculated effectively and efficiently within each category. Finally user will have the experience of effective and efficient extraction of information i.e. documents with the use of document classification as well as the relevance of documents with the use of ranking or sorting based on similarity score.

Threshold calculation is divided into two parts:-first is document classifier and second is threshold tuning. The first part is used to build the classifier and second part is used to tune threshold for each category.


Classifier Building
Training Documents for Classifier Building
DOA Estimation (by Classification)
Training Documents for Threshold Tuning
The Underlying Classifier
AS4T
Thresholding
Thresholds
Incoming Documents
Classification & Filtering
Classified Documents
Filtered Documents

6.3 Algorithm:-

Procedure CSDCTH_TFIDF

Input: Document Sets

Output: Similar class Documents

```
1.  N  -   Number of Documents
2.  IDF-  Inverse Document Frequency
3.  DF -  Document Frequency
4.  $d_i$   -   Document Identifier
5.  For each D do
6.  For each $d_i$ do
7.  Read the document from left to right over the document D
8.  Calculate the Term Frequency for each term $d_i \in D$
9.  Calculate the Inverse Document Frequency for each term $d_i \in D$
10. Calculate the Cosine Similarity
11. Calculate Threshold $T_c$ based on number of matched terms for each $d \in$
    D and for each class label
12.  If Cosine Similarity  $> T_c$

            Document in that particular class is accepted i.e. DOA is true

    Else

            Document in that particular class is rejected i.e. DOA is false
    END If

13. END For
14. END For
15. END
```

In this algorithm TF, IDF and cosine similarity is calculated as in any base classification algorithm. But in previous studies, there was problem of calculation of Threshold value (T_c) as numeric value had to be calculated each time and this value changed if there was some modification in class label. But in this algorithm the Threshold calculation (T_c) is parameter-free. Even if there is modification in class label, this algorithm can adapt to changes and adjust T_c accordingly. As clear from Steps 11 and 12 of this algorithm, T_c is calculated based on required number of matched terms.

CHAPTER 7

IMPLEMENTATION

7.1 IntegerSet:-

```
public class IntegerSet {
private int [] a;  // holds a set of numbers from 0 - 100

  public IntegerSet () {
    // an empty set, all a[i] are set to 0
    a = new int [101];
  }

  // A constructor that copies from an existing set.
  public IntegerSet (IntegerSet existingSet) {
    a = new int [101];
    for(int i=0; i<a.length; i++)
      a[i] = existingSet.a[i];
  }

  public void deleteElement(int i) {
    if ((i >= 0) && (i < a.length))
      a[i] = 0;  // set to 0
  }

  public void insertElement(int i) {
    if ((i >= 0) && (i < a.length))
      a[i] = 1;  // set to 1
  }

  public boolean isSet(int i) {
    return (a[i] == 1);
  }

  // The union of this set and another set
  public IntegerSet unionOfIntegerSets(IntegerSet otherSet) {
    IntegerSet newSet = new IntegerSet(this);

    // newSet is now a copy of the current set,  Next we
    // want to union with set a

    for(int i=0; i<a.length; i++) {
      if (otherSet.isSet(i))
        newSet.insertElement(i);
    }
    return newSet;
  }

  // The intersection of this set and another set
  public IntegerSet intersectionOfIntegerSets(IntegerSet otherSet) {
    IntegerSet newSet = new IntegerSet(this);

    // newSet is now a copy of the current set,  Next we
    // want to intersect with set a

    for(int i=0; i<a.length; i++) {
      if (!otherSet.isSet(i))
        newSet.deleteElement(i);
    }

    return newSet;
  }

  // return true if the set has no elements
  public boolean isEmpty() {
    for (int i=0; i<a.length; i++)
      if (isSet(i)) return false;
    return true;
  }

  // return the 'length' of a set
  public int cardinality() {
    int count = 0;
    for (int i=0; i<a.length; i++)
      if (isSet(i))
        count++;
    return count;
  }

  // Print a set to System.out
  public void setPrint() {
    System.out.print("[Set:");

    if (isEmpty())
      System.out.print("---");

    for (int i=0; i<a.length; i++) {
      if (isSet(i))
        System.out.print(" " + i);
    }

    System.out.print("\n");
  }
```

```
  // return true if two sets are equal
  public boolean isEqualTo(IntegerSet otherSet) {
    for(int i=0; i<a.length; i++) {
      if (otherSet.isSet(i) != isSet(i))
        return false;
    }
    return true;
  }

  // ----------------------------------------------------------
  // Small test program to verify that IntegerSet works!
  public static void main (String [] args) {
    IntegerSet smallEvens = new IntegerSet();
    IntegerSet smallOdds = new IntegerSet();
    for (int i=0; i<10; i++)
      if ((i % 2) == 0)
        smallEvens.insertElement(i);
      else
        smallOdds.insertElement(i);

    System.out.print("smallEvens: ");
    smallEvens.setPrint();

    System.out.print("smallOdds: ");
    smallOdds.setPrint();

    IntegerSet union = smallEvens.unionOfIntegerSets(smallOdds);
    System.out.print("union: ");
    union.setPrint();

    IntegerSet intersection = smallEvens.intersectionOfIntegerSets(smallOdds);
    System.out.print("intersection: ");
    intersection.setPrint();

  }
}
```

7.2 SearchEngine:-

```
import java.util.*;
import java.io.*;
import java.io.File;
import java.io.FileReader;

class Document {
  public IntegerSet set;
  static int count=0;
  //static int doc_count=0;
  public Document (String s,int flag){

 /*   char buffer[] = new char[s.length()];
s.getChars(0,s.length(),buffer,0);

try
{
FileWriter f0 = new FileWriter("file1.doc",true);
for (int i=0;i<buffer.length;i++) {
f0.write(buffer[i]);
}
f0.write(' ');
f0.close();
}
catch(Exception e)
{
        System.out.println("Exception caught");
}*/
if(flag==1)
{

        StringTokenizer st1 = new StringTokenizer(s, " \t\n");

                                try
                                {
                                        FileWriter f0 = new
FileWriter("file1.doc",true);
                                        while (st1.hasMoreTokens()) {



f0.write(st1.nextToken());

count++;

f0.write(' ');

                                                            }
```

```
                                                    //System.out.println("The
numbers of words in file are :"+count);
                                                    f0.close();
                                            }
                                            catch(Exception e)
                                            {
                                                    System.out.println("Exception
caught");
                                            }

        }

                set = new IntegerSet();
            StringTokenizer st = new StringTokenizer(s, " \t\n");
            while (st.hasMoreTokens()) {

                 int index =
SearchEngine.wordToNumber(st.nextToken().toLowerCase(),flag);
              if (index > 0)
                set.insertElement(index);
                    }

          }
        }

        public class SearchEngine {
        static String [] dictionary; /*= {
            "books", "library", "java", "computer", "science", "exam",
            "phone", "dog", "cat", "money", "vacation", "hawaii", "acadia",
            "national", "park", "driving", "car", "the", "algorithm"};*/

                static int count[];
                static int qcount[];
                static int doc[][];

        public void setString(String sbuf[])
        {
                dictionary=sbuf;

        }
        public void setArray(int count1[])
        {
                count=count1;
        }
        public void setqArray(int qcount1[])
        {
                qcount=qcount1;
        }
```

```
            // ----------------------------------------------------------
              // compare two documents (higher numbers are more similar)
                        public static int similarity (Document a, Document b) {
                // number of words in common
                        return a.set.intersectionOfIntegerSets(b.set).cardinality();
              }
                    public static int wordToNumber (String s,int flag){



                        for(int i=0; i<dictionary.length; i++) {
                                                                        if
(s.equalsIgnoreCase(dictionary[i]))
                                                                        {
                                                                        if(flag==1)
                                                                                    {

            (count[i])++;

            //System.out.println(doc[0][0]);

            //(doc[doc_count][i])++;
                                                                                    }
                                                                                    if(flag==0)
                                                                                    {

            qcount[i]++;
                                                                                    }
                                                                                    return i;

                                                                        }
                                                            }
                        return -1;  // word not found
                                                      }
            public static void scoreDocsForQuery(Document [] pages, String s){
                        int flag=0;
                        Document query = new Document(s,flag);
                // Search all web pages (documents) and find the 'closest' one
                 System.out.println();
                 System.out.println("For Query: [" + s + "]");
                 for (int i=0; i<pages.length; i++) {
                  /*System.out.println("\tPage " + i +
                                    " is scored: " + similarity(query, pages[i]));*/
                                     }
                                   }
                public static void main(String[] args) throws Exception {
                    File f= new File("MyFile.doc");
                        FileReader fr = new FileReader(f);
                        BufferedReader br = new BufferedReader(fr);
                        int
length,length1,length2,length3,length4,length5,length6,length7,length8,length9,length
10,i=0,j=0,k=0,k1=0,k2=0,k3=0,flag=1,len,test=0,doc_count=0;
                        char cbuf[]=new char[(int)f.length()];
                        char cbufnew[][]=new char[100][(int)f.length()];
                        length=(int)f.length();
                        br.read(cbuf,0,length);
                        String s=new String(cbuf);
                        //System.out.println(s);
                        StringTokenizer st = new StringTokenizer(s, " \t\n");
                        while (st.hasMoreTokens()) {
                                        st.nextToken().toLowerCase();
                                        k++;
                        }
                        String sbuf[] =new String[k];
                        StringTokenizer st1 = new StringTokenizer(s, " \t\n");
                        while (st1.hasMoreTokens()) {


                                sbuf[k2]=st1.nextToken().toLowerCase();
                                k2++;

                         }
                        k2--;
                        /*for(k3=0;k3<k2;k3++)
                        {
                                System.out.println(sbuf[k3]);
                        }*/
                        fr.close();
                        int counter[]=new int[k2];
                        for(k3=0;k3<k2;k3++)
                        {
                                counter[k3]=0;
                        }
                        int qcounter[]=new int[k2];
                        for(k3=0;k3<k2;k3++)
                        {
                                qcounter[k3]=0;
                        }

                        Document [] webPages = new Document[10];
                        SearchEngine search1=new SearchEngine();
                        search1.setString(sbuf);
                        search1.setArray(counter);
                        search1.setqArray(qcounter);
                        int doc[][]=new int[10][k2];
                     double wdoc[][]=new double[10][k2];
                        for(i=0;i<10;i++)
                        {
```

```
            for(j=0;j<count.length;j++)
            {
                  doc[i][j]=0;

            }
    }
File f1= new File("Document1.txt");
    FileReader fr1 = new FileReader(f1);
    BufferedReader br1 = new BufferedReader(fr1);
    char cbuf1[]=new char[(int)f1.length()];
    char cbufnew1[][]=new char[100][(int)f1.length()];
    length1=(int)f1.length();
    br1.read(cbuf1,0,length1);
    String s1=new String(cbuf1);
System.out.println(s1);
webPages[0] = new Document(s1,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[0].set.isSet(i))
            {
                  doc[doc_count][i]++;
            }
    }
    doc_count++;
File f2= new File("Document2.txt");
    FileReader fr2 = new FileReader(f2);
    BufferedReader br2 = new BufferedReader(fr2);
    char cbuf2[]=new char[(int)f2.length()];
    char cbufnew2[][]=new char[100][(int)f2.length()];
    length2=(int)f2.length();
    br2.read(cbuf2,0,length2);
    String s2=new String(cbuf2);
System.out.println(s2);
    webPages[1] = new Document(s2,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[1].set.isSet(i))
            {
                  doc[doc_count][i]++;
            }
    }
    doc_count++;
File f3= new File("Document3.txt");
    FileReader fr3 = new FileReader(f3);
    BufferedReader br3 = new BufferedReader(fr3);
    char cbuf3[]=new char[(int)f3.length()];
    char cbufnew3[][]=new char[100][(int)f3.length()];
    length3=(int)f3.length();
    br3.read(cbuf3,0,length3);
    String s3=new String(cbuf3);
System.out.println(s3);
    webPages[2] = new Document(s3,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[2].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
    }
doc_count++;
File f4= new File("Document4.txt");
    FileReader fr4 = new FileReader(f4);
    BufferedReader br4 = new BufferedReader(fr4);
    char cbuf4[]=new char[(int)f4.length()];
    char cbufnew4[][]=new char[100][(int)f4.length()];
    length4=(int)f4.length();
    br4.read(cbuf4,0,length4);
    String s4=new String(cbuf4);
System.out.println(s4);
webPages[3] = new Document(s4,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[3].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
    }
    doc_count++;
File f5= new File("Document5.txt");
    FileReader fr5 = new FileReader(f5);
    BufferedReader br5 = new BufferedReader(fr5);
    char cbuf5[]=new char[(int)f5.length()];
    char cbufnew5[][]=new char[100][(int)f5.length()];
    length5=(int)f5.length();
    br5.read(cbuf5,0,length5);
    String s5=new String(cbuf5);
System.out.println(s5);
webPages[4] = new Document(s5,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[4].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
    }
    doc_count++;
File f6= new File("Document6.txt");
    FileReader fr6 = new FileReader(f6);
    BufferedReader br6 = new BufferedReader(fr6);
    char cbuf6[]=new char[(int)f6.length()];
    char cbufnew6[][]=new char[100][(int)f6.length()];
    length6=(int)f6.length();
    br6.read(cbuf6,0,length6);
    String s6=new String(cbuf6);
System.out.println(s6);
webPages[5] = new Document(s6,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[5].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
    }
    doc_count++;
File f7= new File("Document7.txt");
    FileReader fr7 = new FileReader(f7);
    BufferedReader br7 = new BufferedReader(fr7);
    char cbuf7[]=new char[(int)f7.length()];
    char cbufnew7[][]=new char[100][(int)f7.length()];
    length7=(int)f7.length();
    br7.read(cbuf7,0,length7);
    String s7=new String(cbuf7);
System.out.println(s7);
webPages[6] = new Document(s7,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[6].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
}
doc_count++;
File f8= new File("education.txt");
    FileReader fr8 = new FileReader(f8);
    BufferedReader br8 = new BufferedReader(fr8);
    char cbuf8[]=new char[(int)f8.length()];
    char cbufnew8[][]=new char[100][(int)f8.length()];
    length8=(int)f8.length();
    br8.read(cbuf8,0,length8);
    String s8=new String(cbuf8);
System.out.println(s8);
webPages[7] = new Document(s8,flag);
    for(i=0;i<count.length;i++)
    {
            if(webPages[7].set.isSet(i))
            {
                    doc[doc_count][i]++;
            }
}
doc_count++;
File f9= new File("business.txt");
   FileReader fr9 = new FileReader(f9);
   BufferedReader br9 = new BufferedReader(fr9);
   char cbuf9[]=new char[(int)f9.length()];
   char cbufnew9[][]=new char[100][(int)f9.length()];
   length9=(int)f9.length();
   br9.read(cbuf9,0,length9);
   String s9=new String(cbuf9);
System.out.println(s9);
webPages[8] = new Document(s9,flag);
   for(i=0;i<count.length;i++)
   {
            if(webPages[8].set.isSet(i))
            {
                     doc[doc_count][i]++;
            }
}
doc_count++;
File f10= new File("sports.txt");
   FileReader fr10 = new FileReader(f10);
   BufferedReader br10 = new BufferedReader(fr10);
   char cbuf10[]=new char[(int)f10.length()];
   char cbufnew10[][]=new char[100][(int)f10.length()];
   length10=(int)f10.length();
   br10.read(cbuf10,0,length10);
   String s10=new String(cbuf10);
System.out.println(s10);
webPages[9] = new Document(s10,flag);
   for(i=0;i<count.length;i++)
   {
            if(webPages[9].set.isSet(i))
            {
                     doc[doc_count][i]++;
            }
}
System.out.println();
   System.out.println("\t\t\t---------");
   System.out.println("\t\t\tCOUNT tfi");
   System.out.println("\t\t\t---------");
   System.out.println();
   for(j=1;j<8;j++)
            {

                     System.out.print("doc"+j);
                     System.out.print("    ");
            }
```

```
System.out.println();
for(i=0;i<count.length;i++)
{
        for(j=0;j<7;j++)
        {

                System.out.print(doc[j][i]);
                System.out.print("       ");
        }
        System.out.println();
}
//doc[1][8]=2;
//count[8]=1;
scoreDocsForQuery(webPages, "engineering college");
System.out.println();
System.out.println("Q");
for(i=0;i<count.length;i++)
{
        System.out.println(qcount[i]);
}
System.out.println();
double test1,test2;
len=webPages.length;
double idf[]=new double[k2];
double wq[]=new double[k2];
for(i=0;i<count.length;i++)
{
        test1=(double)len;
        test2=(double)count[i];
        idf[i]=(Math.log(test1/test2))/(Math.log(10));
}
for(i=0;i<count.length;i++)
{
        wq[i]=(qcount[i])*(idf[i]);

}
double lengthq,sum=0,sum1=0,sum2=0,sum3=0,sum4=0,sum5=0;
for(i=0;i<count.length;i++)
{
        sum=sum+(wq[i]*wq[i]);
}
lengthq=Math.sqrt(sum);
for(i=0;i<10;i++)
{
        for(j=0;j<count.length;j++)
        {
                wdoc[i][j]=(doc[i][j])*(idf[j]);

        }
```

```
    }
    double lengthdoc[]=new double[len];
    for(i=0;i<len;i++)
    {
            sum=0;
            for(j=0;j<count.length;j++)
            {
                    sum=sum+(wdoc[i][j]*wdoc[i][j]);
            }
            if(i==0)
                    sum=sum+(0.4771*0.4771);
            lengthdoc[i]=Math.sqrt(sum);
    }
    double sim[]=new double[len];
double sim2[]=new double[len-1];
double sim2new[]=new double[len-1];
double sim3new[]=new double[len-1];
double sim3[]=new double[len-1];
double sim4[]=new double[len-1];
double sim1[][]=new double[len-1][len];
double simq[]=new double[3];
    for(i=0;i<len-3;i++)
    {
            sum=0;
            for(j=0;j<count.length;j++)
            {
                    sum=sum+(wdoc[i][j]*wq[j]);
            }
            sim[i]=((sum)/(lengthdoc[i]*lengthq));
    }
for(i=0;i<len-4;i++)
    {
            for(k=i+1;k<len-3;k++)
        {
        sum1=0;
            for(j=0;j<count.length;j++)
            {
                    sum1=sum1+(wdoc[i][j]*wdoc[k][j]);
            }
            sim1[i][k]=((sum1)/(lengthdoc[i]*lengthdoc[k]));
        }
    }
for(i=0;i<len-3;i++)
    {
        sum2=0;
            for(j=0;j<count.length;j++)
            {
                    sum2=sum2+(wdoc[i][j]*wdoc[7][j]);
            }
```

```
                        sim2[i]=((sum2)/(lengthdoc[i]*lengthdoc[7]));

                }
            for(i=0;i<len-3;i++)
                {
                    sum3=0;
                        for(j=0;j<count.length;j++)
                        {
                                sum3=sum3+(wdoc[i][j]*wdoc[8][j]);
                        }
                        sim3[i]=((sum3)/(lengthdoc[i]*lengthdoc[8]));

                }
            for(i=0;i<len-3;i++)
                {
                    sum4=0;
                        for(j=0;j<count.length;j++)
                        {
                                sum4=sum4+(wdoc[i][j]*wdoc[9][j]);
                        }
                        sim4[i]=((sum4)/(lengthdoc[i]*lengthdoc[9]));

                }
            k=0;
            for(i=7;i<10;i++)
            {
                sum5=0;
                for(j=0;j<count.length;j++)
                {
                        sum5=sum5+(wdoc[i][j]*wq[j]);
                }
                simq[k]=((sum5)/(lengthdoc[i]*lengthq));
                k++;
            }
                System.out.println();
                System.out.println("\t\t-------------------------------------------");
                System.out.println("\t\tTERM FREQUENCY/INVERTED
DOCUMENT FREQUENCY");
                System.out.println("\t\t-------------------------------------------");
                System.out.println();
                System.out.print("Terms");
                System.out.print("\t\t\t");
                System.out.print("dfi");
                System.out.print("\t\t\t");
                System.out.print("IDFi");
                System.out.println();
                for(i=0;i<count.length;i++)
                {
                        System.out.print(dictionary[i]);
                if(dictionary[i].length()<8)
                        System.out.print("\t\t\t");
                else
                        System.out.print("\t\t");
                System.out.print(count[i]);
                System.out.print("\t\t\t");
                System.out.print(Math.round(idf[i]*10000.0)/10000.0);
                System.out.println();
        }
        System.out.println();
        System.out.println();
        System.out.println("\t\t\t-------------------");
        System.out.println("\t\t\tWEIGHTS wi=tfi*IDFi");
        System.out.println("\t\t\t-------------------");
        System.out.println();
        for(j=1;j<8;j++)
                {

                        System.out.print("doc"+j);
                        System.out.print("\t\t\t");
                }
        System.out.println();
        for(i=0;i<count.length;i++)
        {
                for(j=0;j<7;j++)
                {


System.out.print(Math.round(wdoc[j][i]*10000.0)/10000.0);
                        System.out.print("\t\t\t");
                }
                System.out.println();
        }
        System.out.println();
        System.out.println("Q");
        for(i=0;i<count.length;i++)
        {
                System.out.println(Math.round(wq[i]*10000.0)/10000.0);
        }
        System.out.println();
        System.out.println("\t\t\t--------------");
        System.out.println("\t\t\tVECTOR LENGHTS");
        System.out.println("\t\t\t--------------");
        System.out.println();
        System.out.print("| D1 |");
        System.out.print("\t\t\t");
        System.out.print("| D2 |");
        System.out.print("\t\t\t");
        System.out.print("| D3 |");
System.out.print("\t\t\t");
   System.out.print("| D4 |");
System.out.print("\t\t\t");
   System.out.print("| D5 |");
System.out.print("\t\t\t");
   System.out.print("| D6 |");
System.out.print("\t\t\t");
   System.out.print("| D7 |");
   System.out.println();
   for(i=0;i<len-3;i++)
   {
            System.out.print(Math.round(lengthdoc[i]*10000.0)/10000.0);
            System.out.print("\t\t\t");
   }
   System.out.print("\n\n");
   System.out.println("| Q |");
   System.out.println(Math.round(lengthq*10000.0)/10000.0);
   System.out.println();
   System.out.println("\t\t\t-----------------");
   System.out.println("\t\t\tSimilarity Values");
   System.out.println("\t\t\t-----------------");
   System.out.println();
   System.out.print("D1");
   System.out.print("\t\t\t");
   System.out.print("D2");
   System.out.print("\t\t\t");
   System.out.print("D3");
System.out.print("\t\t\t");
   System.out.print("D4");
System.out.print("\t\t\t");
   System.out.print("D5");
System.out.print("\t\t\t");
   System.out.print("D6");
System.out.print("\t\t\t");
   System.out.print("D7");
   System.out.println();
   for(i=0;i<len-3;i++)
   {
            System.out.print(Math.round(sim[i]*10000.0)/10000.0);
            System.out.print("\t\t\t");
   }
System.out.println();
System.out.println();
for(i=0;i<len-3;i++)
   {
            for(k=i+1;k<len-2;k++)
      {
      System.out.print("The similarity between document "+(i+1));
      System.out.print(" and document "+(k+1));
```

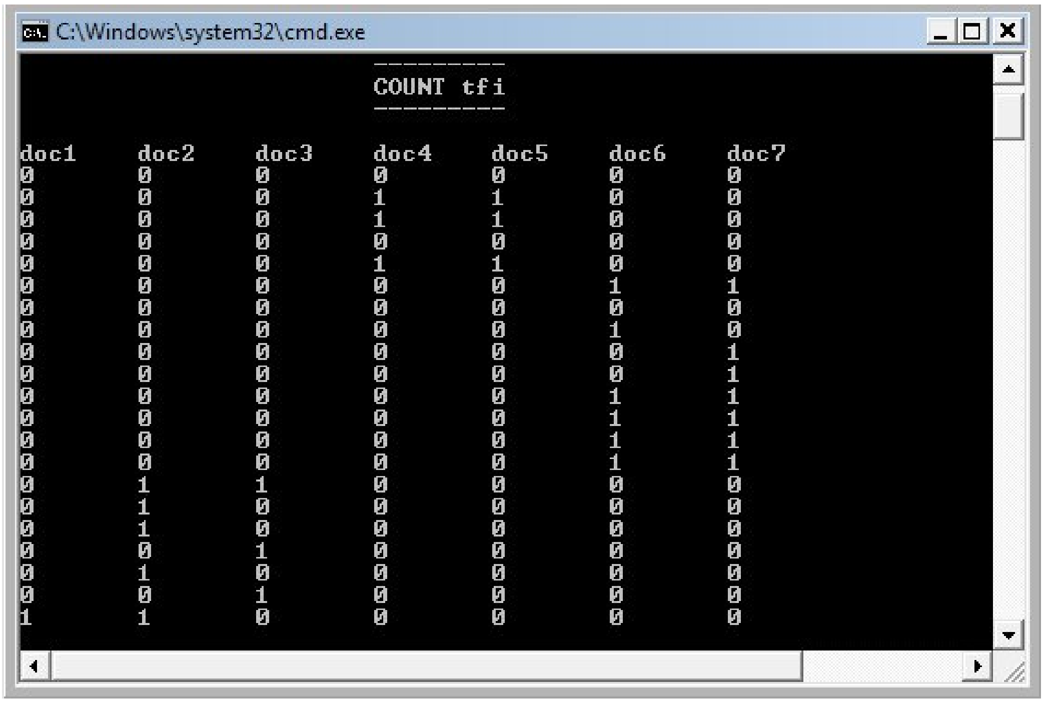
```
        System.out.print(" is ");
        System.out.print(Math.round(sim1[i][k]*10000.0)/10000.0);
            System.out.print("\n");
        }
    }
System.out.print("\n");
for(i=0;i<len-3;i++)
    {
            for(k=i+1;k<len-2;k++)
        {
        if((Math.round(sim1[i][k]*10000.0)/10000.0)>0.4000)
        {
          System.out.print("Document "+(i+1));
          System.out.print(" and Document "+(k+1));
          System.out.print(" are in same cluster");
          System.out.print("\n");
        }
        }
}
System.out.println();
for(i=0;i<len-3;i++)
    {
        System.out.print("The similarity between document "+(i+1));
        System.out.print(" and education ");
        System.out.print(" is ");
        System.out.print(Math.round(sim2[i]*10000.0)/10000.0);
            System.out.print("\n");
        sim2new[i]=(sim2[i]*lengthdoc[i]*lengthdoc[7]);

    }
System.out.print("\n");
System.out.println();
for(i=0;i<len-3;i++)
    {

        System.out.print(Math.round(sim2new[i]*10000.0)/10000.0);
            System.out.print("\n");

    }
System.out.print("\n");
for(i=0;i<len-3;i++)
{
    if((Math.round(sim2[i]*10000.0)/10000.0)>0.2)
    {
        System.out.print("Document "+(i+1));
        System.out.print(" is in 'Education' cluster");
        System.out.print("\n");
    }
}
```

```
System.out.print("\n");

System.out.println();
for(i=0;i<len-3;i++)
   {
      System.out.print("The similarity between document "+(i+1));
      System.out.print(" and business ");
      System.out.print(" is ");
      System.out.print(Math.round(sim3[i]*10000.0)/10000.0);
            System.out.print("\n");
      sim3new[i]=(sim3[i]*lengthdoc[i]*lengthdoc[8]);

   }
System.out.print("\n");
System.out.println();
for(i=0;i<len-3;i++)
   {

      System.out.print(Math.round(sim3new[i]*10000.0)/10000.0);
            System.out.print("\n");

   }
System.out.print("\n");
System.out.print("\n");
for(i=0;i<len-3;i++)
{
   if((Math.round(sim3[i]*10000.0)/10000.0)>0.2)
   {
      System.out.print("Document "+(i+1));
      System.out.print(" is in 'Business' cluster");
      System.out.print("\n");
   }
}
System.out.print("\n");
System.out.println();
for(i=0;i<len-3;i++)
   {
      System.out.print("The similarity between document "+(i+1));
      System.out.print(" and sports ");
      System.out.print(" is ");
      System.out.print(Math.round(sim4[i]*10000.0)/10000.0);
            System.out.print("\n");

   }
System.out.print("\n");
for(i=0;i<len-3;i++)
{
   if((Math.round(sim4[i]*10000.0)/10000.0)>0.2)
   {
        System.out.print("Document "+(i+1));
        System.out.print(" is in 'Sports' cluster");
        System.out.print("\n");
    }
}
System.out.print("\n");
System.out.print("\n");
System.out.println();
for(i=0;i<3;i++)
    {
        System.out.print("The similarity between document");
        System.out.print(" and query ");
        System.out.print(" is ");
        System.out.print(Math.round(simq[i]*10000.0)/10000.0);
                System.out.print("\n");

    }
System.out.print("\n");
if(simq[0]>0.0)
{
    System.out.print("Query is in Education Class");
    System.out.print("\n");
    for(i=0;i<len-3;i++)
    {
        if(sim2[i]>0.0)
        {
            System.out.println(Math.round(sim[i]*10000.0)/10000.0);
        }
    }
}
if(simq[1]>0.0)
{
    System.out.print("Query is in Business Class");
    System.out.print("\n");
    for(i=0;i<len-3;i++)
    {
        if(sim3[i]>0.0)
        {
            System.out.println(Math.round(sim[i]*10000.0)/10000.0);
        }
    }
}
if(simq[2]>0.0)
{
    System.out.print ("Query is in Sports Class");
    System.out.print ("\n");
    for(i=0;i<len-3;i++)
    {
        if(sim4[i]>0.0)
```

```
                {
                    System.out.println(Math.round(sim[i]*10000.0)/10000.0);
                }
            }
        }
    }
}
```

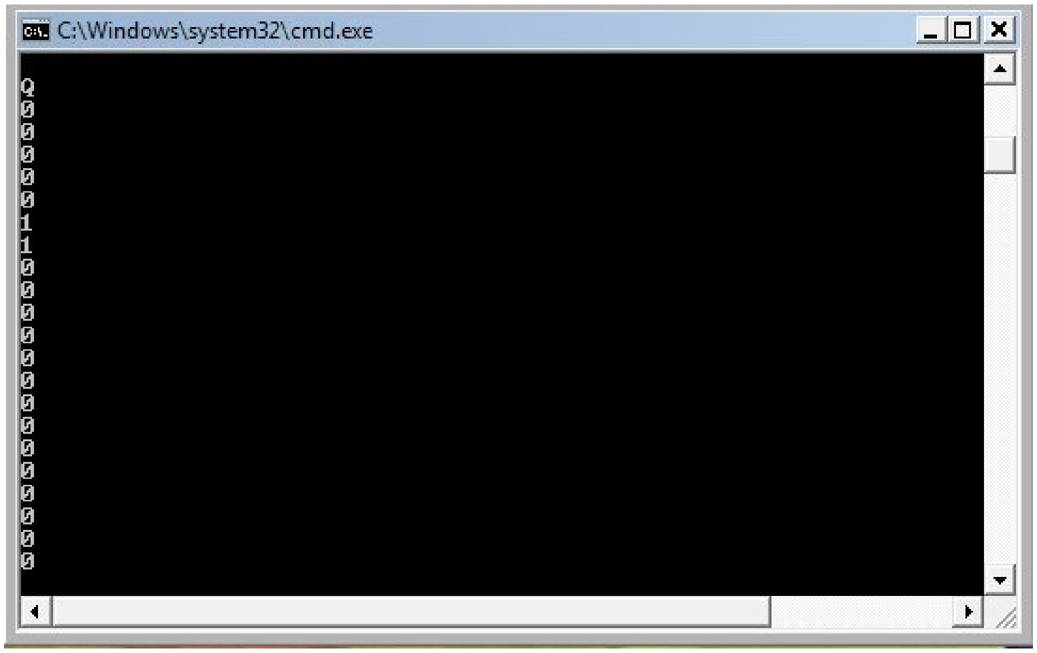
RESULTS

Terms Frequency (TF) Count: - Term frequency is calculated within each document. In start only simple term frequency (TF) is calculated. There are 7 documents and for a query the terms are calculated within each document. In simple TF model only documents with higher term frequency gets higher rank because TF is an example of local information.



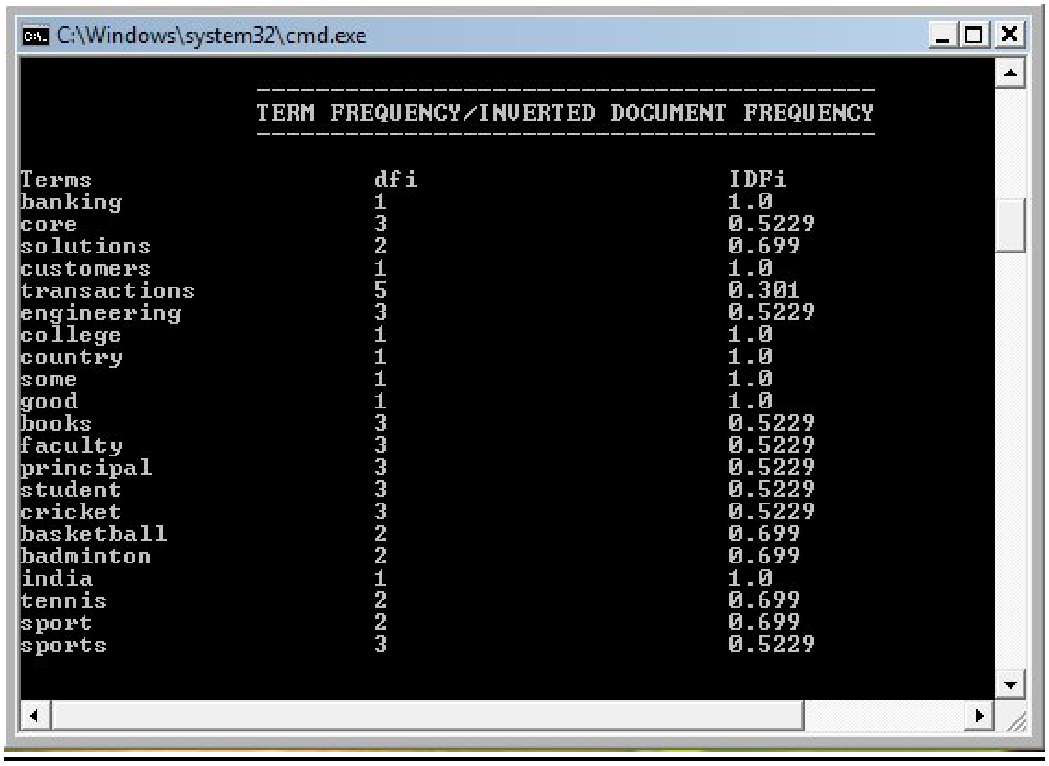
```
                              ---------
                              COUNT tfi
                              ---------

doc1    doc2    doc3    doc4    doc5    doc6    doc7
0       0       0       0       0       0       0
0       0       0       1       1       0       0
0       0       0       1       1       0       0
0       0       0       0       0       0       0
0       0       0       1       1       0       0
0       0       0       0       0       1       1
0       0       0       0       0       0       0
0       0       0       0       0       1       0
0       0       0       0       0       0       1
0       0       0       0       0       0       1
0       0       0       0       0       1       1
0       0       0       0       0       1       1
0       0       0       0       0       1       1
0       0       0       0       0       1       1
0       1       1       0       0       0       0
0       1       0       0       0       0       0
0       1       0       0       0       0       0
0       0       1       0       0       0       0
0       1       0       0       0       0       0
0       0       1       0       0       0       0
1       1       0       0       0       0       0
```

Query Count: - Terms in query are counted. For each term there will be count in query. This query count is used to calculate product with each document count and then it is used in calculating similarity score.



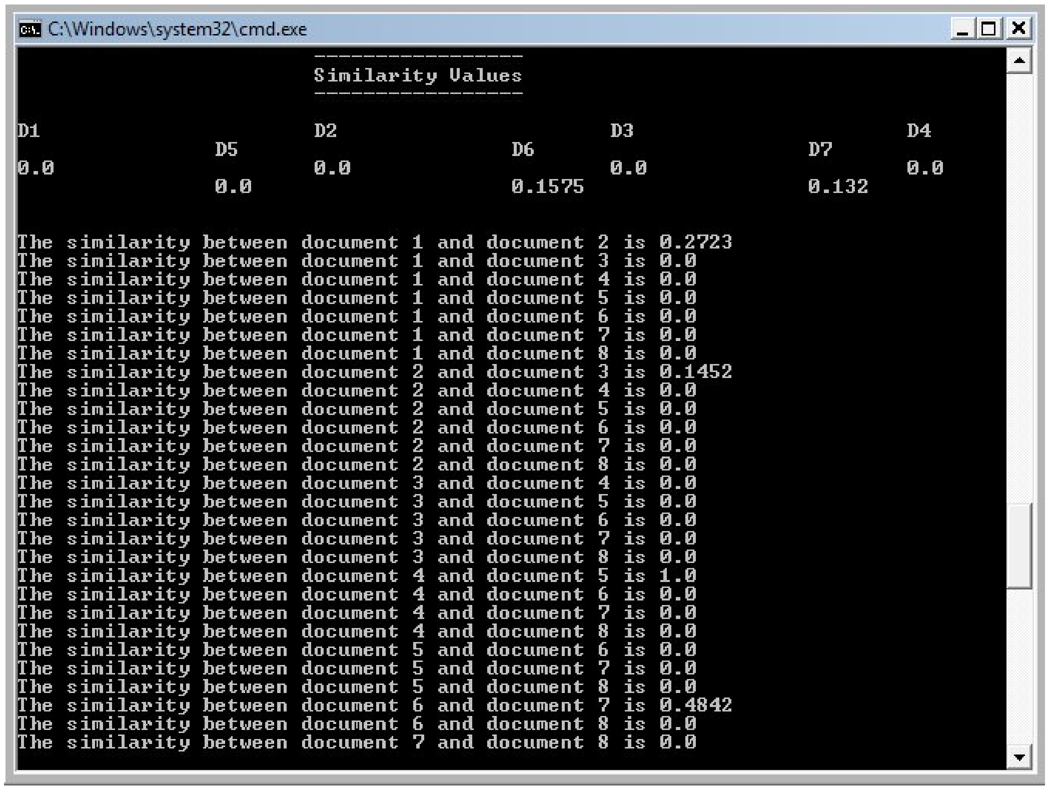
```
Q
0
0
0
0
0
1
1
0
0
0
0
0
0
0
0
0
0
0
0
0
0
```

Inverse Document Frequency (IDF) Count: - Term Frequency gives only local information; Inverse Document Frequency (IDF) is calculated to have global information. IDF is calculated for each term and every document.



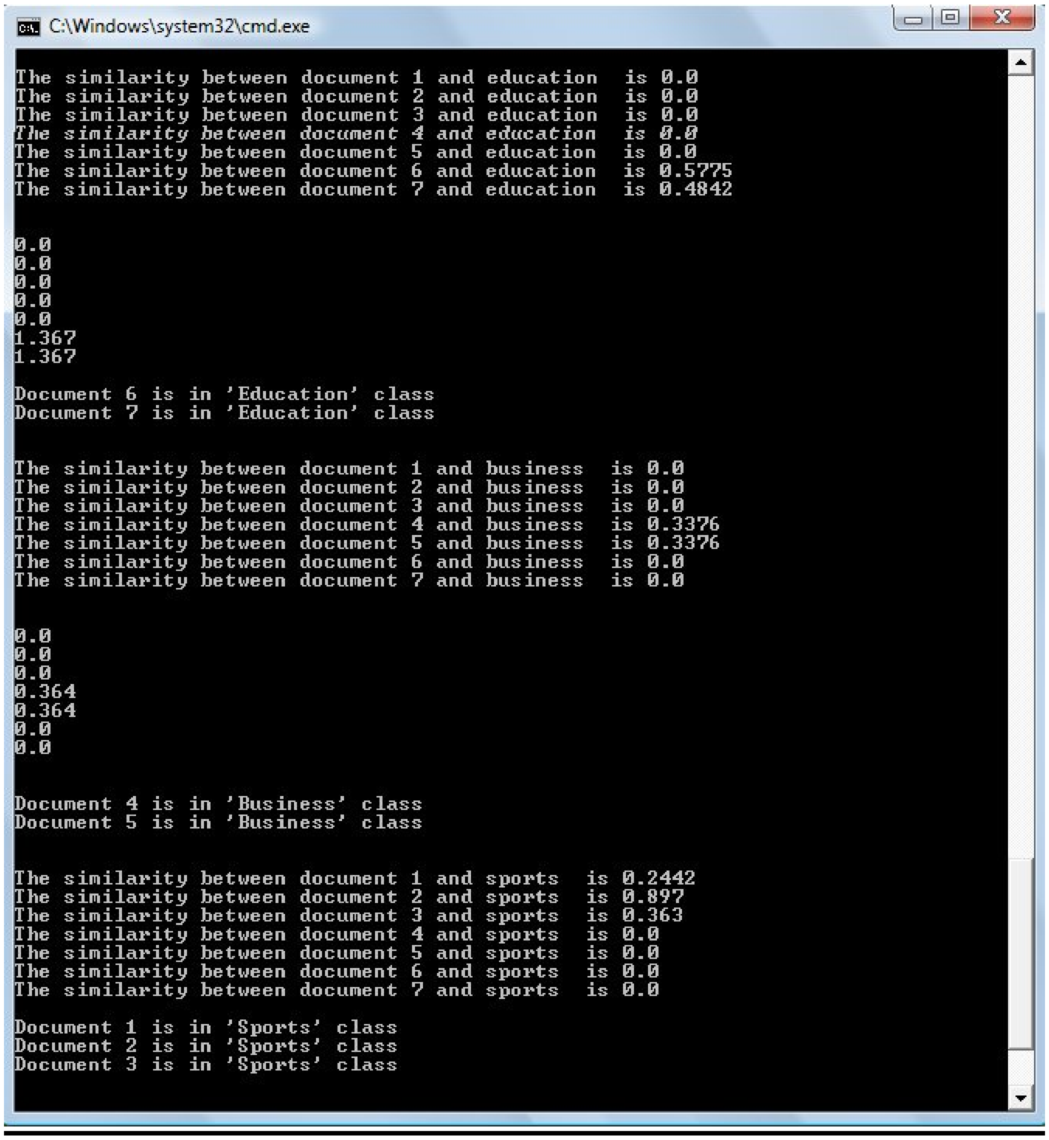
```
C:\Windows\system32\cmd.exe

                 -----------------------------------------
                 TERM FREQUENCY/INVERTED DOCUMENT FREQUENCY
                 -----------------------------------------

Terms                    dfi                      IDFi
banking                  1                        1.0
core                     3                        0.5229
solutions                2                        0.699
customers                1                        1.0
transactions             5                        0.301
engineering              3                        0.5229
college                  1                        1.0
country                  1                        1.0
some                     1                        1.0
good                     1                        1.0
books                    3                        0.5229
faculty                  3                        0.5229
principal                3                        0.5229
student                  3                        0.5229
cricket                  3                        0.5229
basketball               2                        0.699
badminton                2                        0.699
india                    1                        1.0
tennis                   2                        0.699
sport                    2                        0.699
sports                   3                        0.5229
```

Similarity Values: - Similarity values are calculated between each document and query. Similarity score is used for document ranking. Similarity score between query and document results are used for document ranking, but for document classification similarity between documents have to be calculated. So cosine similarity as distance measure is calculated between each document. Documents are assigned to same class if similarity score is above threshold value.



```
                        -----------------
                        Similarity Values
                        -----------------

D1                      D2                      D3                      D4
                D5                      D6                      D7
0.0                     0.0                     0.0                     0.0
                0.0                     0.1575                  0.132


The similarity between document 1 and document 2 is 0.2723
The similarity between document 1 and document 3 is 0.0
The similarity between document 1 and document 4 is 0.0
The similarity between document 1 and document 5 is 0.0
The similarity between document 1 and document 6 is 0.0
The similarity between document 1 and document 7 is 0.0
The similarity between document 1 and document 8 is 0.0
The similarity between document 2 and document 3 is 0.1452
The similarity between document 2 and document 4 is 0.0
The similarity between document 2 and document 5 is 0.0
The similarity between document 2 and document 6 is 0.0
The similarity between document 2 and document 7 is 0.0
The similarity between document 2 and document 8 is 0.0
The similarity between document 3 and document 4 is 0.0
The similarity between document 3 and document 5 is 0.0
The similarity between document 3 and document 6 is 0.0
The similarity between document 3 and document 7 is 0.0
The similarity between document 3 and document 8 is 0.0
The similarity between document 4 and document 5 is 1.0
The similarity between document 4 and document 6 is 0.0
The similarity between document 4 and document 7 is 0.0
The similarity between document 4 and document 8 is 0.0
The similarity between document 5 and document 6 is 0.0
The similarity between document 5 and document 7 is 0.0
The similarity between document 5 and document 8 is 0.0
The similarity between document 6 and document 7 is 0.4842
The similarity between document 6 and document 8 is 0.0
The similarity between document 7 and document 8 is 0.0
```

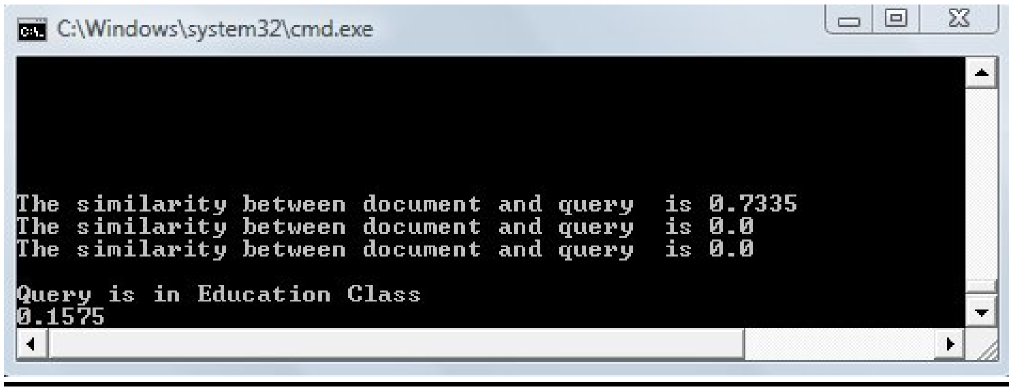
Classification: - Documents are classified after calculating similarity with predefined class labels. There are different class labels used like education, business and sports. After this implementation module documents are classified in their appropriate class.



C:\Windows\system32\cmd.exe

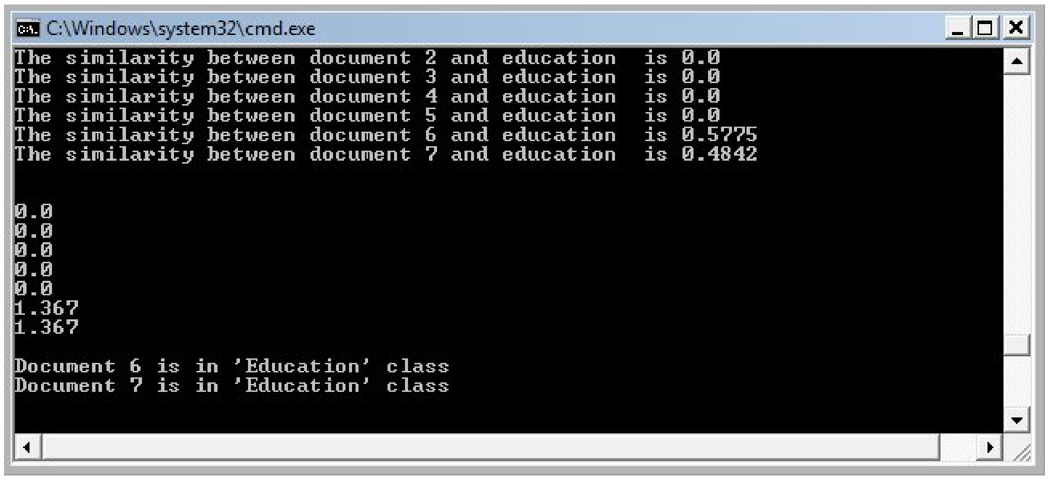
```
The similarity between document 1 and education  is 0.0
The similarity between document 2 and education  is 0.0
The similarity between document 3 and education  is 0.0
The similarity between document 4 and education  is 0.0
The similarity between document 5 and education  is 0.0
The similarity between document 6 and education  is 0.5775
The similarity between document 7 and education  is 0.4842


0.0
0.0
0.0
0.0
0.0
1.367
1.367

Document 6 is in 'Education' class
Document 7 is in 'Education' class


The similarity between document 1 and business  is 0.0
The similarity between document 2 and business  is 0.0
The similarity between document 3 and business  is 0.0
The similarity between document 4 and business  is 0.3376
The similarity between document 5 and business  is 0.3376
The similarity between document 6 and business  is 0.0
The similarity between document 7 and business  is 0.0


0.0
0.0
0.0
0.364
0.364
0.0
0.0


Document 4 is in 'Business' class
Document 5 is in 'Business' class


The similarity between document 1 and sports  is 0.2442
The similarity between document 2 and sports  is 0.897
The similarity between document 3 and sports  is 0.363
The similarity between document 4 and sports  is 0.0
The similarity between document 5 and sports  is 0.0
The similarity between document 6 and sports  is 0.0
The similarity between document 7 and sports  is 0.0

Document 1 is in 'Sports' class
Document 2 is in 'Sports' class
Document 3 is in 'Sports' class
```

Document Ranking and classification: - In this module documents are ranked for every class .Query is checked that for which class user wants information and then only for that class query results are shown and documents are classified and ranked.



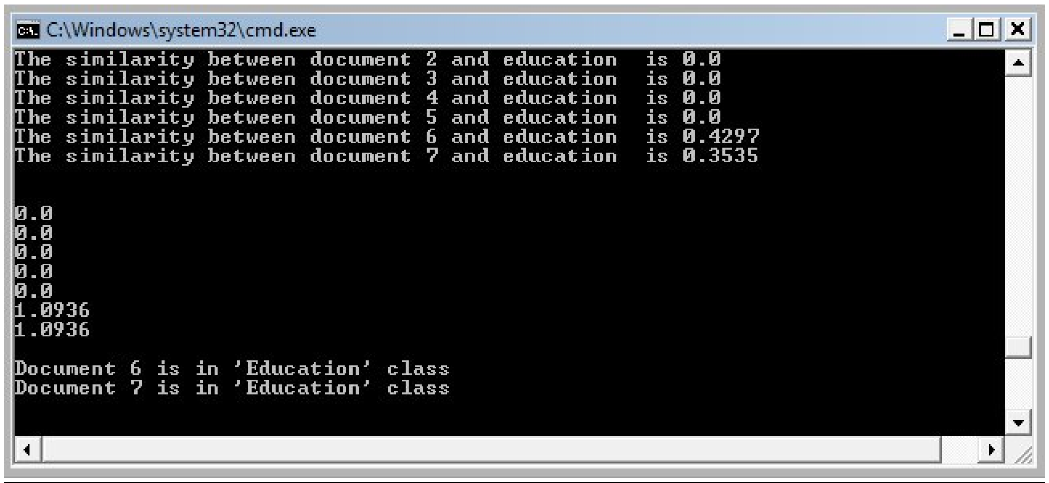
```
C:\Windows\system32\cmd.exe

The similarity between document and query  is 0.7335
The similarity between document and query  is 0.0
The similarity between document and query  is 0.0

Query is in Education Class
0.1575
```

Threshold for 5 terms matching: - In this module 5 terms threshold is taken and then it is multiplied by that testing document and predefined class label document to get uniform threshold.



```
The similarity between document 2 and education  is 0.0
The similarity between document 3 and education  is 0.0
The similarity between document 4 and education  is 0.0
The similarity between document 5 and education  is 0.0
The similarity between document 6 and education  is 0.5775
The similarity between document 7 and education  is 0.4842


0.0
0.0
0.0
0.0
0.0
1.367
1.367

Document 6 is in 'Education' class
Document 7 is in 'Education' class
```

Threshold for 4 terms matching: - In this module 4 terms threshold is taken and then it is multiplied by that testing document and predefined class label document to get uniform threshold.



```
The similarity between document 2 and education  is 0.0
The similarity between document 3 and education  is 0.0
The similarity between document 4 and education  is 0.0
The similarity between document 5 and education  is 0.0
The similarity between document 6 and education  is 0.4297
The similarity between document 7 and education  is 0.3535


0.0
0.0
0.0
0.0
0.0
1.0936
1.0936

Document 6 is in 'Education' class
Document 7 is in 'Education' class
```

CONCLUSION

TF-IDF is an efficient and simple algorithm for matching words in query to documents that are relevant to that query. TF-IDF returns documents that are highly relevant to a particular query. If a user were to input a query for a particular topic, TF-IDF can find documents that contain relevant information on the query. Furthermore, encoding TF-IDF is straightforward, making it ideal for forming the basis for more complicated algorithms and query retrieval systems. Despite its strength, TF-IDF has its limitations. In terms of synonyms, notice that TF-IDF does not make the jump to the relationship between words. If the user wanted to find information about, say, the word “priest”, TF-IDF would not consider documents that might be relevant to the query but instead use the word “reverend”.

By using Vector Space Model with TF-IDF Model, the documents can be ranked more accurately than probabilistic model. Cosine similarity and vector length calculation of Vector Space Model combined with tf and idf value and similarity based on tf*idf normalization of TF-IDF Model provides a better model for distinguishing between relevant documents from non-relevant documents.

Nearest Neighbor Algorithm gives very good result when cosine similarity is used as distance measure. Documents are classified very accurately by using this algorithm. Threshold value should be set properly so that false positives and false alarms are not there in classified output.

FUTURE WORK

TF-IDF is used as core technique for most of the search engines which are currently being used in data mining and information retrieval community. This technique when used with Vector Space Model gives very efficient and effective results according to the requirements of the user. The further improvements in this algorithm can be better removal of stop words and better technique for stemming. Further research can be on minimizing the sparse matrix which is generated while representing documents as vectors.

Other similarity measures can be applied for calculating scores between query and document in case of document ranking or document and predefined class label in case of document classification. Dice Coefficient, Manhattan Distance and Euclidean Distance can be applied for further research.

Further research can be on removing the problem of local optimum in Document Classification. Some global optimum value should be found out to remove local optimum problem. In Nearest Neighbor algorithm, further research can be on calculation of setting threshold. S-Cut Threshold technique can be modified for giving scores which can be used in any user environment.

REFERENCES

(1) Juan Ramos Department of Computer Science, Rutgers University "Using TF-IDF to Determine Word Relevance in Document Queries"

(2) Berger, A & Lafferty, J. Information Retrieval as Statistical Translation. In *Proceedings of the 22nd ACM Conference on Research and Development in Information Retrieval* (SIGIR), 222-229.

(3) G. Salton, A.Wong, C.S.Yang, "A Vector Space Model for Automatic Indexing", IRLC, ACM

(4) Christian Platzer and Schahram Dustdar Vienna University of Technology, Distributed Systems Group, Information Systems Institute, "A Vector Space Search Engine for Web Services"

(5) M. S. Xiaoying Tai, Y. Tanaka, "Improvement of vector space information retrieval model based on supervised learning", ACM.

(6) Information Retrieval - Wikipedia

(7) Stephen Robertson- Microsoft Research
"Understanding Inverse Document Frequency: On theoretical arguments for IDF"

(8) Aizawa, A., "An information-theoretic perspective of tf-idf measures", Information Processing and Management, Vol. 39, pp. 45–65.

(9) David Grossman, Ophir Frieder, "Vector Space Implementation", Illinois Institute of Technology

(10) Robertson, S. E., "The probability ranking principle in information retrieval", Journal of Documentation, Vol. 33, pp. 294–304.

(11) Davide Magatti, Fabio Stella and Marco Faini, "A Software System for Topic Extraction and Document Classification"

(12) Rey-Long Liu, Wan-Jung Lin, "Adaptive sampling for thresholding in document filtering and classification"

(13) Barry Britt, University of Tennessee, "Document Classification Techniques using LSI"

(14) S.Suseela, Periyar Maniammai University, "Document Clustering Based on Term Frequency and Inverse Document Frequency"

(15) Bader Aljaber, Nicola Stokes, James Bailey, Jian Pei, "Document Clustering of scientific texts using citation context" , LLC, Springer 2009

(16) Wei Xu, Xin Liu, Yihong Gong, "Document Clustering Based on Non-negative Matrix Factorization", SIGIR, ACM 2001

(17) T.W.Fox, "Document Vector Compression and Its Application in Document Clustering", CCECE/CCGEI, IEEE 2005

(18) Gang Qian, Shamik Sural, Yuelong Gu, Sakti Pramanik,"Similarity between Euclidean and cosine angle distance for nearest neighbor queries", SAC, ACM 2004

(19) Alok Ranjan, Eatesh Kandpal, Harish Verma, Joydip Dhar , “An Analytical Approach to Document Clustering Based on Internal Criterion Function”, IJCSIS Vol.7 ,No. 2, 2010

(20) Yiming Yang, “A Study on Thresholding Strategies for Text Categorization”, SIGIR, ACM 2001

(21) Sergey Brin, Lawrence Page, “The Anatomy of a Large-Scale Hypertextual Web Search Engine”, Stanford University

(22) John Zakos, Brijesh Verma, “A Novel Context Matching Based Technique for Web Document Retrieval”, ICDAR, IEEE 2005

(23) Michael Steinbach, George Karypis, Vipin Kumar, “A Comparison of Document Clustering Techniques”, University of Minnesota

(24) T.W.Fox, “Document Vector Compression and Its Application in Document Clustering”, IEEE 2005

(25) Dawei Song, Raymond Y.K.Lau, Peter D. Bruza , Kam-Fai Wong, Ding-Yi Chen, “An intelligent information agent for document title classification and filtering in document-intensive domains”, ScienceDirect 2007

(26) James G. Shanahan, Norbert Roma, “Boosting Support Vector Machines for Text Classification through Parameter-free Threshold Relaxation”, ACM 2003