Legal Document Search Engine

Project report by

Muhammad Suleman Tanveer

BSCS

Project Supervisor

Dr. Akmal Saeed Khattak

Department of Computer Sciences,

Quaid-i-Azam University,

Islamabad.

Session (2013-17)

DECLARATION

I hereby declare that I have developed this report entirely on the basis of my personal efforts under the sincere guidance of my supervisor. All the sources used in this report have been cited and the contents of this report have not been plagiarized. No portion of the work presented in this report has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Muhammad Suleman Tanveer

A report submitted to

Department of Computer Science

Quaid-i-Azam University, Islamabad

As a partial fulfillment of the requirements for the award of the degree of

BS in Computer Science.

Project Brief

Project Title:

Legal Document Search Engine

Developed By:

Muhammad Suleman Tanveer

Supervised By:

Dr. Akmal Saeed Khattak

Development Tools Used:

Eclipse Java Neon.3, JetBrains PyCharm 2016.3.1, Eclipse Java Mars 1.0

Operating System:

Windows 10

Document Design Tool:

MS Word 2013

Dedicated to

To

The Holiest Man Ever Born,

PROPHET MUHAMMAD (S.A.W.A.S)
&
To
My PARENT AND FAMILY

I am most indebted to my parents and family, whose affection has always been the source of encouragement for
Me, and whose prayers have always been a key to my
Success.

&

To

THOSE LOVED ONES AND FRIENDS

Who always worried and prayed for my success and gave me continuous moral support and encouragement.

&

To

MY HONORABLE TEACHERS

Who have been a beacon of knowledge and a constant source of inspiration for my whole life span.

ACKNOWLEDGEMENT

In the name of Allah, most Beneficent, most Merciful. First of all I want to thank to the Almighty Allah on the completion of my project, as I completed this task only by His favor and grace. At this moment this is due on me to than some personalities because without their cooperation and supervision, I may be unable to complete this work.

First of all my respected teachers and my project supervisor, **Dr. Akmal Saeed Khattak** whose door of kindness always remains open for all of his students. Completion of my task would not be possible without his help, encouragement, dynamic supervision and constructive criticism. I am highly indebted to express my gratitude to him for his entire collaboration. Specail thanks to **Dr. Rabbeh Ayaz Abbasi**, **Dr. Ghazanfar Farooq**, **Miss Ifrah Farrukh Khan**, **Dr. Khalid Saleem**, **Dr. Mubashar Mushtaq**, **Dr. Mudassir Azam Sindhu**, **Madam Memona Afsheen**, **Sir S. M. Naqi**, **Dr. Shuaib Karim**, **Mr. Umer Rahseed** and **Dr. Muhammad Usman** for their kind support and cooperation.

I am also very much thankful to my parents, family and friends whose prayers are treasure of my life. I have no words to pay gratitude to them whose affection, guidance and continuous encouragement did their best to shame my character.

In the end I would like to thank to all of my class fellows and my seniors. And especially to those who help me in completing my survey.

Thanks Everyone…!

Muhammad Suleman Tanveer
2013-2017

Abstract

Legal Document Search Engine is a web based application that enables user to search legal documents. Legal documents include court decisions, constitution and legislation. User will provide a query and against that a query a list of ranked documents will be displayed to the user. Two different ranked retrieval models are used to develop this search engine that are: tf-IDF and BM25. After implementing both ranked retrieval models the system is evaluated against a query set. Both models are evaluated against a same query set and then result are compared in order to find the best ranked retrieval model for this system. For persuasion of the user a dynamic word cloud has been made which is regenerated in response to every query along with auto-correction and suggestions for the given words.

Table of Contents

List of Figures

List of Tables

Chapter 1

Software Project Management Plan

1.1 Introduction

In this chapter SPMP (Software Project Management Plan), a plan is described which will be followed for our final year project for the degree of BS Computer Science. The project ‘Legal Document Search Engine’ is basically a Research and Development based project.

1.1.1 Project Overview

The name of project is ‘Legal Document Search Engine’. This will be a web based application for searching about legal information. This product will be available for free for common citizens so that they can access legal information. Legal information refers to court decision, contracts and legislation. To access legal information against a particular information need, user’s information need is expressed in the form of a query. In response to that query user is provided a list of ranked documents. Documents are shown to the user according to their relevancy. The most relevant document is shown at very top, the second most relevant document is placed at second positon and so on.

1.1.2 Project Deliverables

Possible project deliverables are given below.

- Survey report.
- Architecture.
- Evaluation Mechanism which will be used to evaluate the search engine.
- Document collection and classification.
- Product design.
- Design implementation.
- Implementation of algorithms for indexing of documents.
- Implementation of ranked retrieval models

1.2 Software Process Model

We will be using waterfall process model. Requirements are pretty clear that’s why we are using waterfall software process model.

1.2.1 Roles and Responsibilities

Since I am the only one who is dealing with this project so my responsibilities are to complete all tasks define in time table before time. My roles are:

- Do analysis.
- Propose comprehensive solution.
- Design interface.
- Implementation of algorithms.
- Testing.

1.2.2 Tools and Techniques

Tools that I will be using are Eclipse JAVA Mars for implementation of algorithms, and for inverted index, and MS-Word for documentation. First documents will be indexed and different ranked retrieval models will be applied. Ranked retrieval models will be used for ranking of result set.

1.3 Project Management Plan

Following are the tasks for our project.

1.3.1 Problem Analysis

Problem analysis consist of some of sub tasks. Estimated duration for this task is 11 days.

1.3.1.1 Problem Understanding

In this phase we will be dealing with problem understanding. Discussing it with supervisor and having complete understanding of the project. Estimated time for this subtask is 2 days.

1.3.1.2 Problem Survey

After the understanding of problem we will be doing survey of the problem. Collecting information about the similar problems and their solution, matching solution with our problem. Since this task involves lot of reading therefore estimated time for this subtask is 1 week (8 days).

1.3.1.3 Proposed Solution

After having understanding of the problem and having survey we then require to propose a solution for our problem. This phase will take almost 2 days.

1.3.2 Legal Document Collection

In this phase we will be collecting legal documents. Estimated time for this task is 18 days. This task consist of subtasks described below.

1.3.2.1 Finding Resources

In this task we will be finding resources from where we can collect legal documents. These resources can be either court or internet (from official website of Supreme Court of Pakistan). Estimated time for this task is 15 days.

1.3.2.2 Understanding of Legal Documents

The documents collected from previous task are need to be understood. For this understanding we will require almost 4 days. Reading out different court decision and understanding the structure of these documents.

1.3.3 Survey

In this phase we will be doing survey by reading different research papers and finding the similar systems and their working. This task also involve some sub tasks.

1.3.3.1 Collecting Related Research Papers

In this task we will be browsing internet, visiting libraries in order to get related research papers. Research papers related to information retrieval system and legal information retrieval system. Estimated time for this task is 3 days.

1.3.3.2 Reading Research Papers and Other Material

In this task I shall be reading different research papers and referred books recommended by the supervisor.

1.3.3.3 Propose a Solution

After reading different related research papers and article I would try to propose a better problem solution. This task will require some brainstorming. Estimated time for this task is 3 days.

1.3.4 Architecture

This phase involve the architecture design of our problem. This task involve some of sub task described below.

1.3.4.1 Diagrammatic View

In this task I shall be describing architecture of our system in the form of diagram. Estimated time for this task is 2 days.

1.3.4.2 Describing Architecture

In this task I shall be describing the architecture of our project. The diagram obtained from very previous task will be explained in detail. Estimated time for this task will be 10 days.

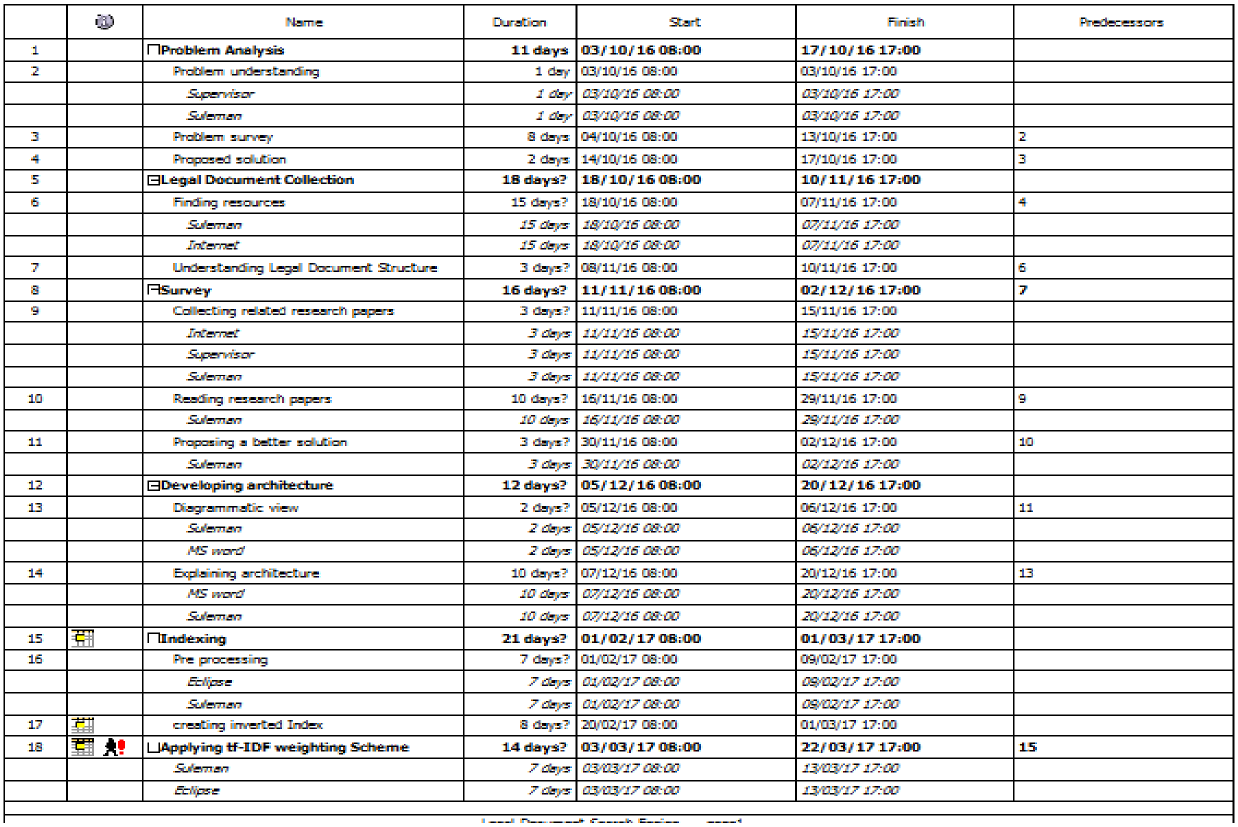
1.3.5 Implementation

In this phase I will be dealing with implementation of our product. Probably a period of 4 months will be given to this phase.

1.3.6 Finding and Results

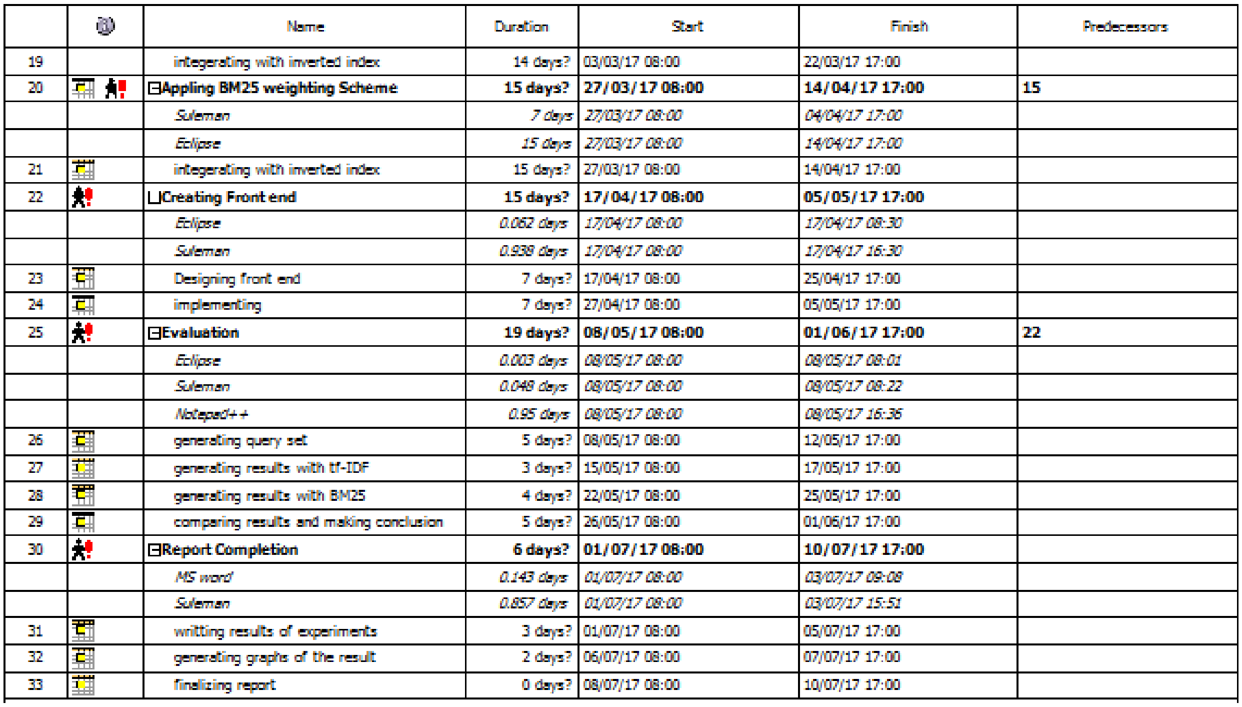
After implementation of product the last task will be its evaluation. Different experiments will be made to evaluate the system and at the end conclusion will be provided.

1.3.7 Timetable



	ⓘ	Name	Duration	Start	Finish	Predecessors
1		**Problem Analysis**	**11 days**	**03/10/16 08:00**	**17/10/16 17:00**	
2		Problem understanding	1 day	03/10/16 08:00	03/10/16 17:00	
		Supervisor	*1 day*	*03/10/16 08:00*	*03/10/16 17:00*	
		Suleman	*1 day*	*03/10/16 08:00*	*03/10/16 17:00*	
3		Problem survey	8 days	04/10/16 08:00	13/10/16 17:00	2
4		Proposed solution	2 days	14/10/16 08:00	17/10/16 17:00	3
5		**Legal Document Collection**	**18 days?**	**18/10/16 08:00**	**10/11/16 17:00**	
6		Finding resources	15 days?	18/10/16 08:00	07/11/16 17:00	4
		Suleman	*15 days*	*18/10/16 08:00*	*07/11/16 17:00*	
		Internet	*15 days*	*18/10/16 08:00*	*07/11/16 17:00*	
7		Understanding Legal Document Structure	3 days?	08/11/16 08:00	10/11/16 17:00	6
8		**Survey**	**16 days?**	**11/11/16 08:00**	**02/12/16 17:00**	**7**
9		Collecting related research papers	3 days?	11/11/16 08:00	15/11/16 17:00	
		Internet	*3 days*	*11/11/16 08:00*	*15/11/16 17:00*	
		Supervisor	*3 days*	*11/11/16 08:00*	*15/11/16 17:00*	
		Suleman	*3 days*	*11/11/16 08:00*	*15/11/16 17:00*	
10		Reading research papers	10 days?	16/11/16 08:00	29/11/16 17:00	9
		Suleman	*10 days*	*16/11/16 08:00*	*29/11/16 17:00*	
11		Proposing a better solution	3 days?	30/11/16 08:00	02/12/16 17:00	10
		Suleman	*3 days*	*30/11/16 08:00*	*02/12/16 17:00*	
12		**Developing architecture**	**12 days?**	**05/12/16 08:00**	**20/12/16 17:00**	
13		Diagrammatic view	2 days?	05/12/16 08:00	06/12/16 17:00	11
		Suleman	*2 days*	*05/12/16 08:00*	*06/12/16 17:00*	
		MS word	*2 days*	*05/12/16 08:00*	*06/12/16 17:00*	
14		Explaining architecture	10 days?	07/12/16 08:00	20/12/16 17:00	13
		MS word	*10 days*	*07/12/16 08:00*	*20/12/16 17:00*	
		Suleman	*10 days*	*07/12/16 08:00*	*20/12/16 17:00*	
15		**Indexing**	**21 days?**	**01/02/17 08:00**	**01/03/17 17:00**	
16		Pre processing	7 days?	01/02/17 08:00	09/02/17 17:00	
		Eclipse	*7 days*	*01/02/17 08:00*	*09/02/17 17:00*	
		Suleman	*7 days*	*01/02/17 08:00*	*09/02/17 17:00*	
17		creating inverted Index	8 days?	20/02/17 08:00	01/03/17 17:00	
18		**Applying tf-IDF weighting Scheme**	**14 days?**	**03/03/17 08:00**	**22/03/17 17:00**	**15**
		Suleman	*7 days*	*03/03/17 08:00*	*13/03/17 17:00*	
		Eclipse	*7 days*	*03/03/17 08:00*	*13/03/17 17:00*	

Figure 1. 1 *Project Planning Timetable (1)*



		Name	Duration	Start	Finish	Predecessors
19		integerating with inverted index	14 days?	03/03/17 08:00	22/03/17 17:00	
20		⊟Appling BM25 weighting Scheme	15 days?	27/03/17 08:00	14/04/17 17:00	15
		Suleman	*7 days*	*27/03/17 08:00*	*04/04/17 17:00*	
		Eclipse	*15 days*	*27/03/17 08:00*	*14/04/17 17:00*	
21		integerating with inverted index	15 days?	27/03/17 08:00	14/04/17 17:00	
22		⊟Creating Front end	15 days?	17/04/17 08:00	05/05/17 17:00	
		Eclipse	*0.062 days*	*17/04/17 08:00*	*17/04/17 08:30*	
		Suleman	*0.938 days*	*17/04/17 08:00*	*17/04/17 16:30*	
23		Designing front end	7 days?	17/04/17 08:00	25/04/17 17:00	
24		implementing	7 days?	27/04/17 08:00	05/05/17 17:00	
25		⊟Evaluation	19 days?	08/05/17 08:00	01/06/17 17:00	22
		Eclipse	*0.003 days*	*08/05/17 08:00*	*08/05/17 08:01*	
		Suleman	*0.048 days*	*08/05/17 08:00*	*08/05/17 08:22*	
		Notepad++	*0.95 days*	*08/05/17 08:00*	*08/05/17 16:36*	
26		generating query set	5 days?	08/05/17 08:00	12/05/17 17:00	
27		generating results with tf-IDF	3 days?	15/05/17 08:00	17/05/17 17:00	
28		generating results with BM25	4 days?	22/05/17 08:00	25/05/17 17:00	
29		comparing results and making conclusion	5 days?	26/05/17 08:00	01/06/17 17:00	
30		⊟Report Completion	6 days?	01/07/17 08:00	10/07/17 17:00	
		MS word	*0.143 days*	*01/07/17 08:00*	*03/07/17 09:08*	
		Suleman	*0.857 days*	*01/07/17 08:00*	*03/07/17 15:51*	
31		writting results of experiments	3 days?	01/07/17 08:00	05/07/17 17:00	
32		generating graphs of the result	2 days?	06/07/17 08:00	07/07/17 17:00	
33		finalizing report	0 days?	08/07/17 08:00	10/07/17 17:00	

Figure 1. 2 *Project Planning Timetable (2)*

1.3.8 Gant chart

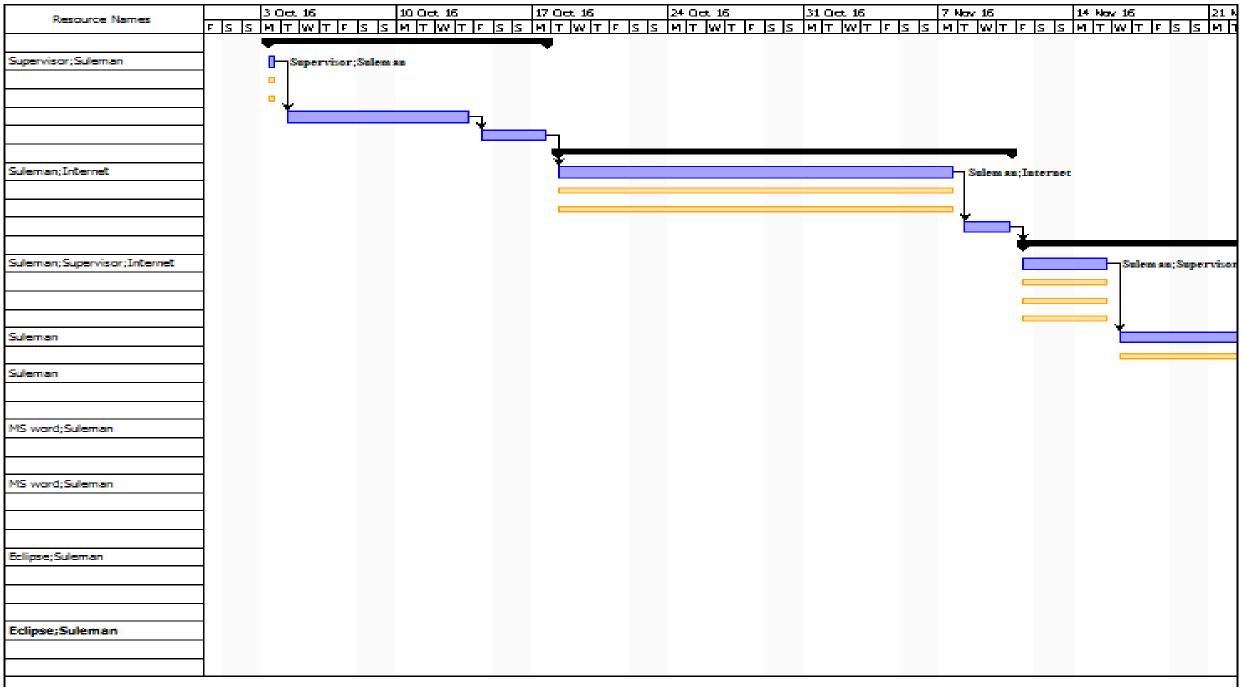


Figure 1. 3 *Gantt Chart (1)*

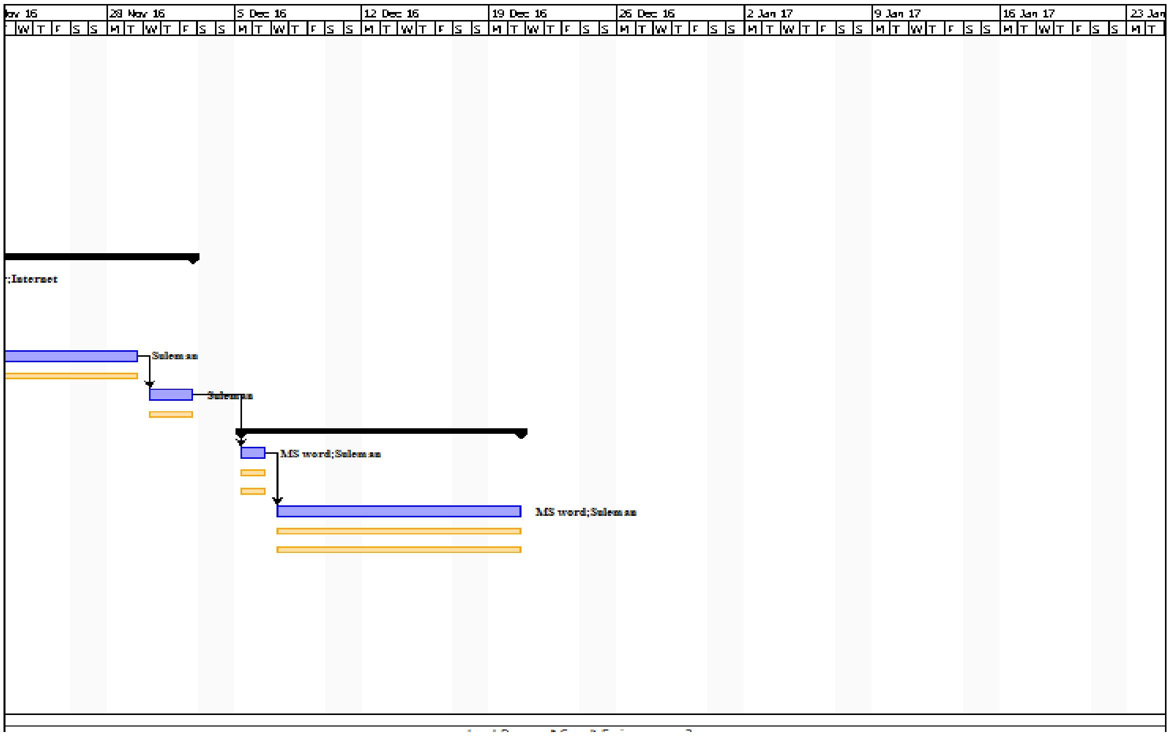


Figure 1. 4 Gant Chart (2)

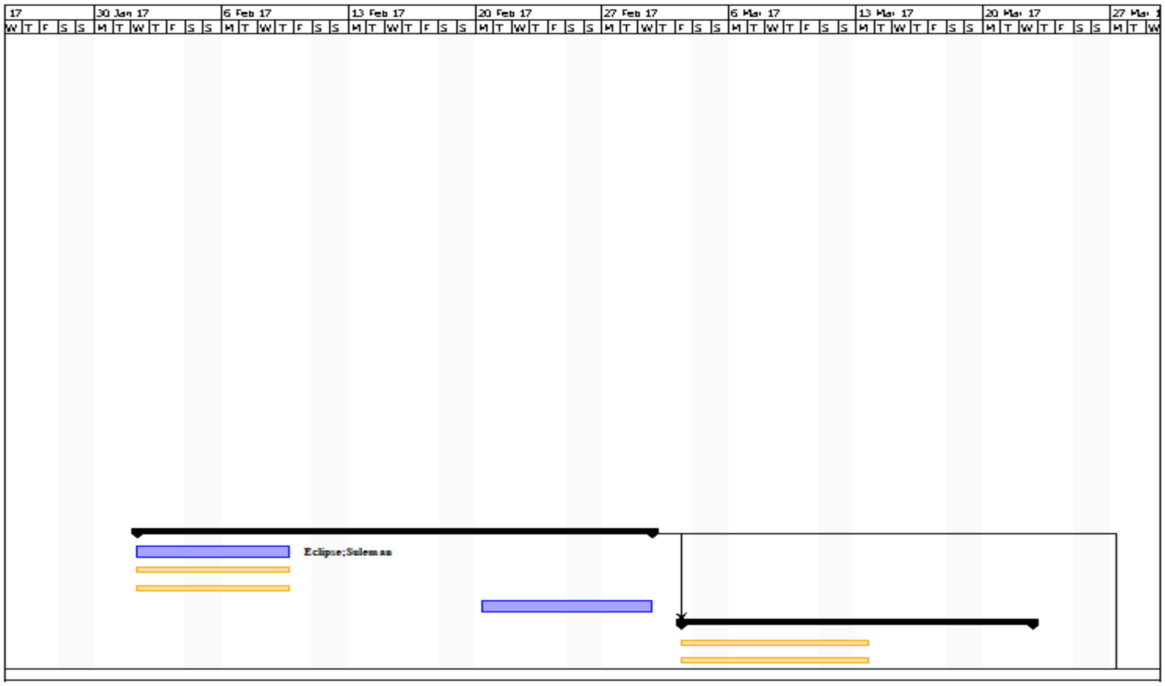


Figure 1. 5 Gant Chart (3)

Chapter 2

Legal Document Search Engine Introduction

2.1 Introduction

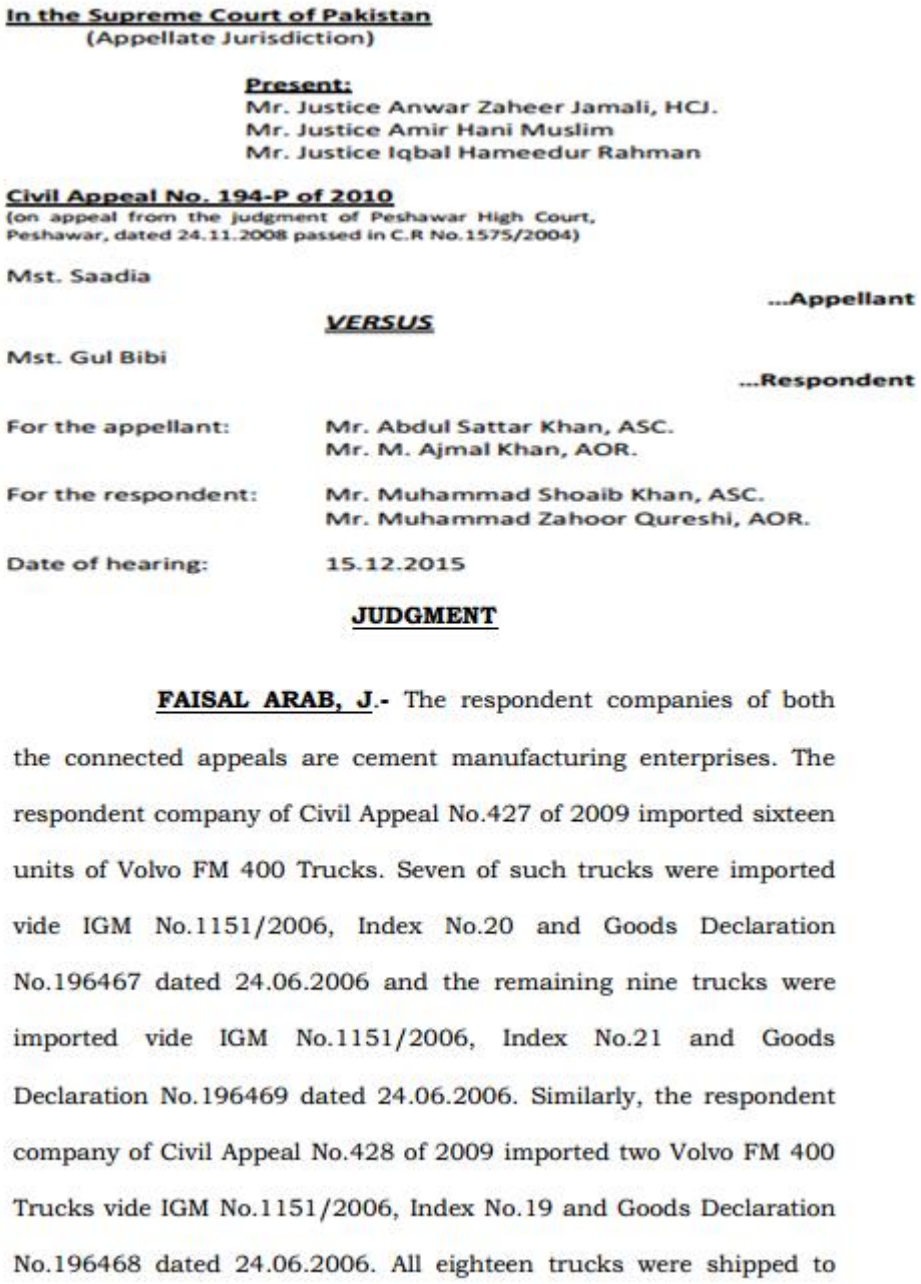
Information retrieval refers to the retrieval of information from a collection of documents with respect to some information need expressed in the form of user's query. User enters some keywords against which a list of documents is displayed as a result. There are a number of information retrieval systems which allows user to express their query with a set of key terms and the result of the query is a list of documents, some of IR systems are Google, Yahoo, and Bing etc. These IR systems are general, not very specific. What we are going to develop is a dedicated information retrieval system that will work for a specific documents (legal documents). Also there are numerous dedicated information retrieval system that work for a specific field like biology, mathematics, and for computer science etc. We are going to develop a search engine for Legal documents i.e., for court decisions of Supreme Court of Pakistan and constitution of Pakistan. This may help common citizen for getting information about legal issues and can make decisions based on related issues.

2.2 Legal Documents

Legal document is a document that grants some rights and states some contract based relationship. These documents involve some text. Some of these texts may be a part of legislation, some of them consist of contracts, police statements, official pleadings, warrants and court decisions. According to research [12] the most commonly legal document used in information retrieval systems are legislation and court decisions. Legal documents are in natural language text.

Decision of judges could also be counted in legal documents and are a source of law. Lawyers and even civilians can build arguments for their current case based on the previous cases that will support them and make their case strong. Usually decisions are written in natural language. Court decisions have two basic components, *opinions* and *facts*. In opinion, judge give reason about the case, explain concepts or interprets legislation. The other component i.e. *facts* of a case are very important. The *facts* are the basis on which the judge has made his conclusion.

The law uses common words found in ordinary language in combination with specific interpretations of legal terms. Characteristics of a court decision is shown in figure below.



In the Supreme Court of Pakistan
(Appellate Jurisdiction)

Present:
Mr. Justice Anwar Zaheer Jamali, HCJ.
Mr. Justice Amir Hani Muslim
Mr. Justice Iqbal Hameedur Rahman

Civil Appeal No. 194-P of 2010
(on appeal from the judgment of Peshawar High Court, Peshawar, dated 24.11.2008 passed in C.R No.1575/2004)

Mst. Saadia
...Appellant

VERSUS

Mst. Gul Bibi
...Respondent

For the appellant:	Mr. Abdul Sattar Khan, ASC. Mr. M. Ajmal Khan, AOR.
For the respondent:	Mr. Muhammad Shoaib Khan, ASC. Mr. Muhammad Zahoor Qureshi, AOR.
Date of hearing:	15.12.2015

JUDGMENT

FAISAL ARAB, J.- The respondent companies of both the connected appeals are cement manufacturing enterprises. The respondent company of Civil Appeal No.427 of 2009 imported sixteen units of Volvo FM 400 Trucks. Seven of such trucks were imported vide IGM No.1151/2006, Index No.20 and Goods Declaration No.196467 dated 24.06.2006 and the remaining nine trucks were imported vide IGM No.1151/2006, Index No.21 and Goods Declaration No.196469 dated 24.06.2006. Similarly, the respondent company of Civil Appeal No.428 of 2009 imported two Volvo FM 400 Trucks vide IGM No.1151/2006, Index No.19 and Goods Declaration No.196468 dated 24.06.2006. All eighteen trucks were shipped to

Figure 2. 1 Court Decision Example

2.3 Scope

Legal document search engine is a web based application for searching legal information. Legal information refers to court decision, contracts and legislation. To access legal information against a particular information need, user's information need is expressed in the form of a query. There are two modes of retrieving information against an information need, one is push mode and other is pull mode. In push mode, system recommends user for relevant information whereas in pull mode user initiates request for an information need in the form of query. We will be dealing with pull mode. Browsing and querying are included in pull mode. In browsing user navigates into relevant information by following a path enabled by the structures on the documents, useful when user don't know what keywords to use. Whereas in querying mode user provide keywords for his/her information need. It is useful when if user knows which keywords to use. A query will be provided by the user. The query terms are than matched with the documents. On the basis of matching, a number of documents are retrieved. These retrieved documents are than ranked according to their relevancy. Different ranked retrieval models are used for ranking which includes TF-IDF, vector space modelling, BM25 and SMART modelling.

2.4 Motivation

Information is knowledge and having access to more knowledge gives you a valuable advantage over you competitor especially in case of legal case. If you have more knowledge about your case you can make your case more strong by referring to the previous cases. Court decisions are also a source of law, and right to get legal information is one of the basic right of any individual. Currently in Pakistan there is no search engine available where a common citizen can access legal information. Only lawyer community have access to the legal information and they pay some amount per year for getting legal information. With the help of this search engine every person can access legal information and take their decisions accordingly. Legal information includes previous court decision and legislation. Case belongs to some category like murder case, missing person, robbery, property case etc. The motivation is to develop a legal document search engine which provide access to legal information to common citizen to Pakistan.

Moreover Pakistan bar council have a legal document search engine, but it is follows Boolean retrieval model. This search is not publically available. Boolean retrieval model works in a way that either we are in or out. Disadvantage of this model is *feast* or *famine* i.e., either we will have too few hits or too many hits. Our focus is to use different term weighting schemes in ranked retrieval model to overcome the short comes of Boolean retrieval model. Legal documents will be retrieved using free text queries in English, against a query a list of legal documents is displayed which are shown to the user according to their relevancy. On backend of search engine the documents are indexed, in this way we can improve our searching and it will be better than Boolean retrieval model (indexing and how documents are matched with query terms is described below in detail). The reference to the matched documents against the query are pulled from the collection of documents and shown to the user on the basis of ranking. Ranking refers to the relevancy of the document with the query, most relevant documents are shown to the user first. Different ranking models are used for ranking of the documents described below. This type of legal document search engine isn't available in Pakistan. So it is a great step towards betterment for information retrieval in legal case.

Chapter 3

Literature Survey

3.1 Introduction

In this chapter we will be explaining some survey made on existing legal documents search engines and will try to explain the mechanism they are using for indexing their documents. Moreover we will be describing what technique they are using for text processing also will suggest what techniques we are going to use for the implementation of our search engine.

3.2 Westlaw and LexisNexis

There are some legal document search engines available like WESTLAW and LexisNexis. WESTLAW is a legal research service for lawyers and legal professionals. WELSTLAW is used in over 60 countries [2]. It include cases and statues of Australia, Hong Kong, Canada, European Union, United Kingdom and United States. WESTLAW supports Boolean as well as natural language searches. Documents on WESTLAW are indexed according to West Key Number System. In 19th Century John B. West described the classification of legal documents. He divided law into major categories which he called topics (such as Murder Case, Contracts etc.)[1]. He created hundreds of subcategories and assigned them a number (key). This key number is used to identify (refer) a case. The list of those key numbers can be seen on this link (http://static.legalsolutions.thomsonreuters.com/product_files/westlaw/wlawdoc/wlres/keynmb06.pdf). WESLAW include a feature named “keyCite”. It is a citation checking service which allow customers to check whether case or statues are still good law. Good law means verification of citation, because lawyers must determine whether a case has been overruled, modified or reversed before making reference of that case.

WESTLAW introduced WestlawNext on February 08, 2010. It works in a way that suppose you want to search “Can a municipality be held liable for civil rights violation by its employees?” WestlawNext return some results, key cases ranked by relevance to your topic (your query). It also search across content type like Cases, Statues and Pending and Proposed legislation. If you want to search deeper it suggest related cases as well. When you enter a query it matches the related documents by applying different algorithms and rank them by relevancy. It uses *federated search* across multiple content types [3] (Federated search is an information retrieval technology that allows simultaneous search of multiple searchable resources). When a user makes a query, it is distributed to different search engines or other query engines that are participating in the federation. The federal search then aggregates the results received from participating engines and present it to the user). When they index them they assign them a key number means they are putting them in a category. This is helpful because if you don't make (describe) your query well even it can bring some good (related) documents on the bases of the selected category [4].Filters can also be applied further on the result set. It also provides a facility to retrieve documents by references like citation, keyCite etc. It also provide folder for storing portion of the result selected by user [4]. WestlawNext has been renamed Thomson Reuters Westlaw, effective from February 2016 [5].

The LexisNexis is cooperation that provides legal document search. Its collection of documents contains laws and statutes of United States. It also have case judgments and opinions for jurisdiction such as Australia, Canada, France, Hong Kong, South Africa and United Kingdom.

These legal document search engine are based on NLP (Natural Language Processing) technique. In NLP we are interested in semantics. It consist of five staged, tokenization, lexical analysis, syntactic analysis, semantic analysis and pragmatic analysis [21]. But what we are going to build is an IR system that uses a fraction of NLP and core mathematical model of information retrieval system. The parts of NLP that we will be using are tokenization, stop word removal, stemming and lemmatization and normalization and mathematical models are vector space model, tf-idf, BM25, SMART.

Chapter 4

Architecture and Methodology

4.1 Introduction

In this chapter architecture and methodology of a search engine will be discussed. Inverted index will be described along with different term weighting schemes like tf-IDF, BM25 and SMART. These schemes are used for ranking. Ranking is very important in every search engine therefore different schemes will be discussed along with their pros and cons. After discussing the architecture of search engine different evaluation measures will discussed that will be taken after the implementation of this product. These evaluation measure includes precision, recall and F-measures. In the process of information retrieval user will provide its information need, this information will be in the form of query. There will be a collection of document, these documents must will be indexed. Before indexing the documents, there must be some pre-processing on these them. Pre-processing is a process in which data (documents) are prepared for another process (for indexing). Pre-processing includes tokenization, removal of stop words, normalization and stemming. Detailed pre-processing techniques are described below in section 4.2. After pre-processing on the collection, it is ready for indexing. Indexing holds the vocabulary (words used in the document) and it's posting list. Posting list is a list which shows that in which documents a particular word is appearing, it holds document id of it along with its frequency in the corpus. Each document is different from the other on the basis of document id. Detail about indexing is described in section 4.3. Once documents are indexed then user enters his/her information need in the form of a query, his/her query is processed and on the basis of relevancy, a list of relevant documents are retrieved. These retrieved documents are then ranked according to their relevancy. Their relevancy is found by using different type of term weighting schemes like TF-IDF, vector space model and BM25. The document which is most relevant is displayed on the top and the second most is displayed on the second number and so on. First k (let say k=10) documents are shown to the user. These k documents are most relevant to the provided query. Detail about ranking is described in section 4.4. Figure 4.1 shows the diagrammatic view of architecture of a search engine.

Below (in Fig 4.1) is shown the architecture of a typical Information System.

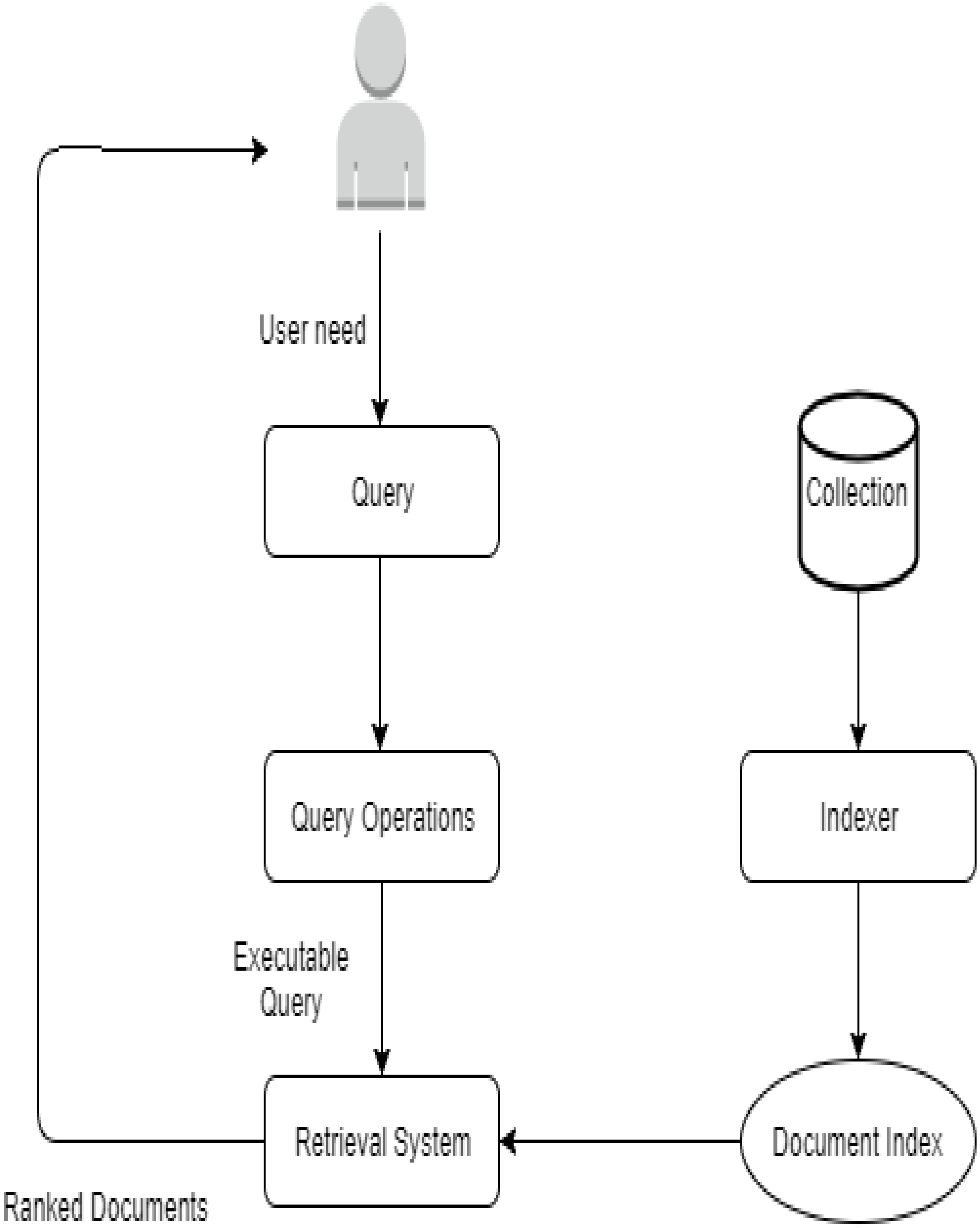


Figure 4. 1 *Information Retrieval System Architecture*

4.2 Preprocessing

Processing performed on a raw data for preparing it for another process is called preprocessing. The first step towards text mining is preprocessing. Figure 4.2 shows the hierarchy of pre-processing techniques.

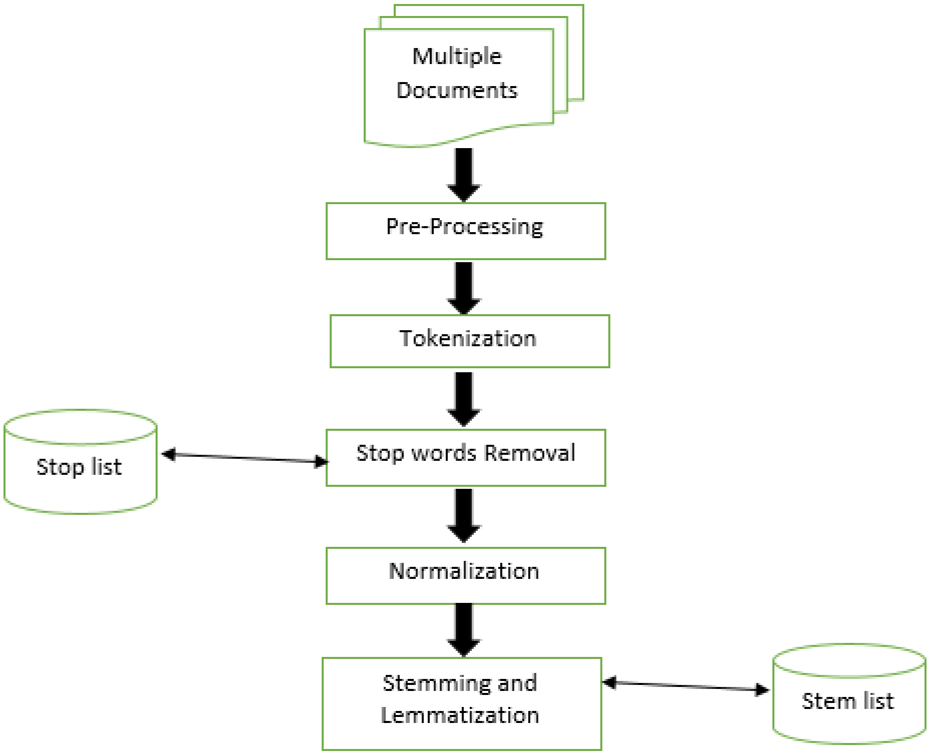


Figure 4. 2 Pre-processing

Following are techniques of preprocessing.

4.2.1 Tokenization

Chopping a sequence of characters into collection of words is referred as tokenization. The use of tokenization is to get or identify meaningful keywords.

Input: The court decides that, this hearing is done now.

Output: The court decides that this hearing is done now

Tokens of same *types* are stored (indexed) once [11]. (The term *type* referrers to a class of all tokens containing the same character sequence).

4.2.2 Stop Words Removal

Some frequent words that are used in documents which may have little value but they will appear most of the times, these words are referred as *stop words*. The most common words in a document is articles, pro-nouns and prepositions which do not give meaning about document and their occurrence is high although they do not represent a document. So the removal of these words (*stop words*) is necessary. Examples of *stop words* are a, an, is, the, He etc.

4.2.3 Normalization

Now we have tokens but still there are some problems with these tokens. For example if there is a token like U.S.A and if someone query like USA, it will not match. For these related problems we have to normalize our tokens.

Token normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequence of the tokens [7]. For normalization of tokens *equivalence classes* can be made. For example if the token *anti-corruption* and *anticorruption* are both mapped on a term anticorruption, in both document and text query then search against one term will retrieve the documents that contain either *anti-corruption* or *anticorruption*.

4.2.4 Stemming and Lemmatization

In documents, usually different form of a word is used like implementing, implemented, implements. These words are derived from some root word. Like in this case, all of the words belong to root word which is ‘implement’. Stemming is used to replace derived terms from their root (word stem). *Stemming* is the process that simply chops the ends of word and hope for achieving it correctly whereas, *Lemmatization* do these thing in a proper way. It uses vocabulary and morphological analysis of words (morphological analysis is a method for identifying the total set of possible relationship). Lemmatization not only chops the end of a derived word but also give the root (word stem) of word known as *lemma* [9].

4.3 Inverted index

An inverted index is an index which is used for storing mapping from content to its location in collection of documents or files. Content can be of any type like numbers and free text etc. An inverted index consist of posting lists associated with each term that is in the documents. The structure of posting list is shown in figure 4.3.

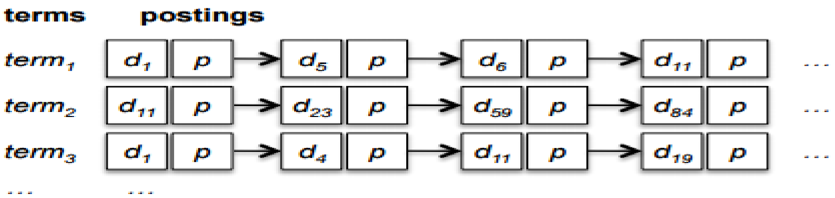


Figure 4. 3 Posting list example

A posting list is composed of different posting that contain the document id and payload (p), payload is the information about the occurrence of word in a document which we can say *term frequency* (*tf*). Term frequency (tf) tells us that how many time a word occurred in a particular document d1, d2, d5 etc., are basically document ids. Documents can be identified by a unique number that can have range from 0 to n.

An inverted index is composed of two components, *vocabulary* and *occurrence.* The term vocabulary refers to the list of words used in the collection (all the documents). For each word in the vocabulary there is a reference of the list of documents in which the particular word occurred, this list represents the occurrence of word in the documents.

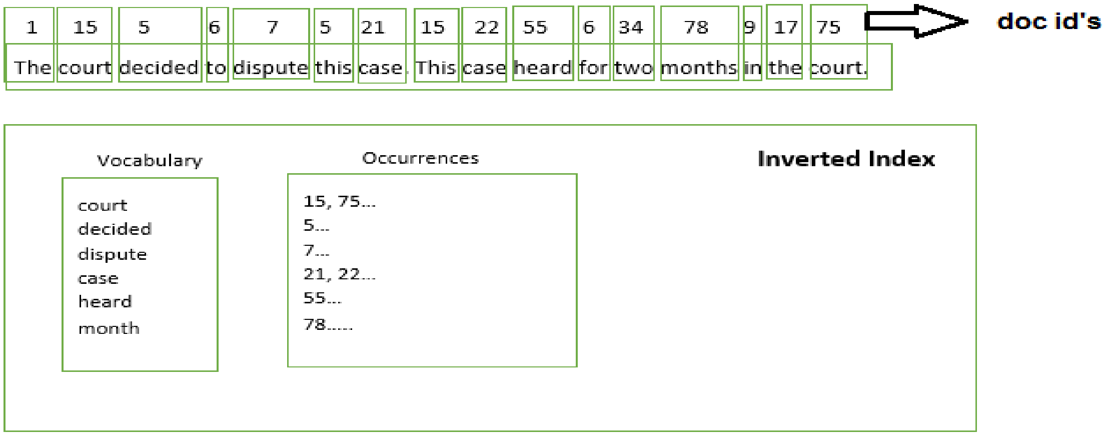
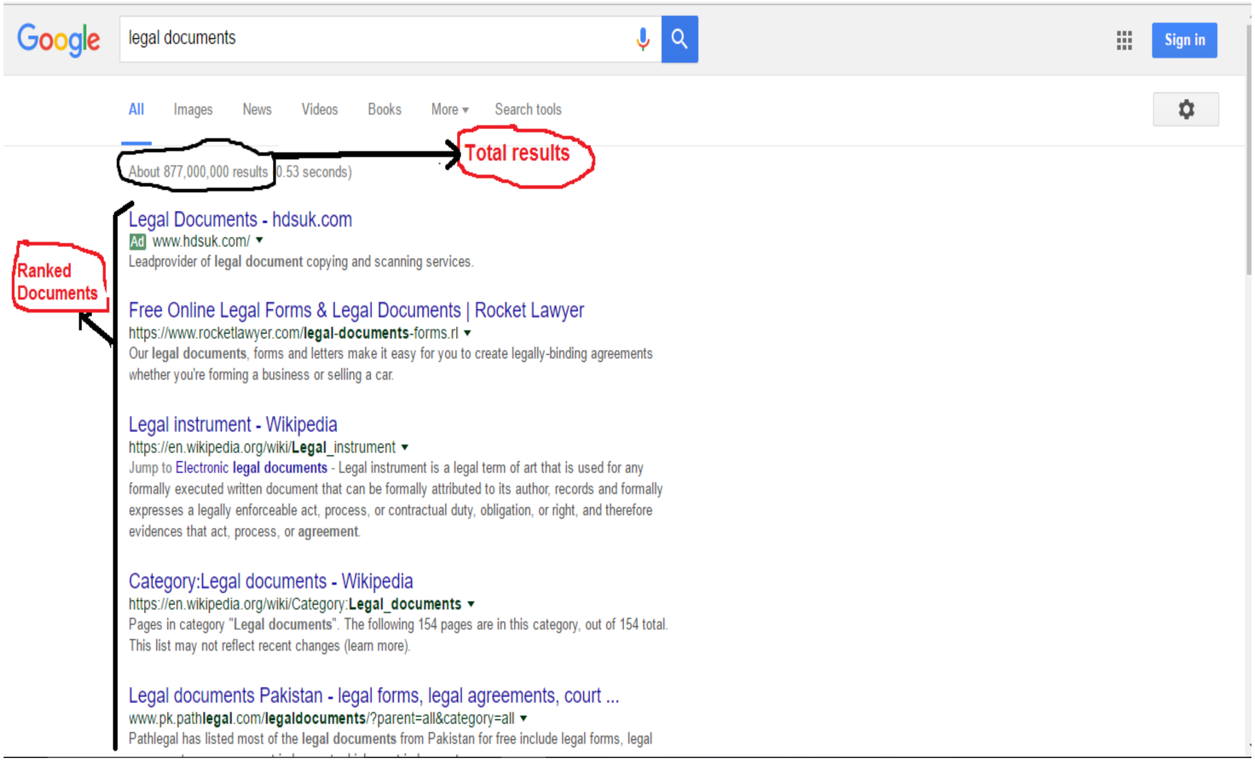


***Figure 4. 4** Inverted Index Example*

4.4 Ranking

Ranking is a key step in information retrieval system. When a user enters some query, this query is processed and related documents against that particular query are retrieved. The number of relevant documents can be very large. Ranking is very important because it reduces a large result set to a smaller one. Most of the user don't wade through 1000's of results, instead of it user want to look at a few of results usually first two or three of it. For this, documents must be ranked accordingly i.e., most relevant document should appear at the top and second most important document should be displayed on second number and so on. Documents are usually ranked on the basis of their relevancy to the query i.e., documents which are related to given query should be retrieved. First k (let say k = 10) documents are displayed to the user among all the retrieved documents, and these first k documents are shown on the basis of ranking as described above. Like in google if we query something it gets millions of result but displays only some of them in first list, the documents which are displayed in top k (let say k =10) list are on the basis of their ranking. (Fig 4.5). Different ranked retrieval models are used for ranking the result set which includes TF-IDF and vector space modelling etc. We will be using TF-IDF, BM25 and SMART technique along with vector space model for ranking of documents (result).



About 877,000,000 results (0.53 seconds)

Legal Documents - hdsuk.com
Ad www.hdsuk.com/ ▾
Leadprovider of **legal document** copying and scanning services.

Free Online Legal Forms & Legal Documents | Rocket Lawyer
https://www.rocketlawyer.com/**legal-documents**-forms.rl ▾
Our **legal documents**, forms and letters make it easy for you to create legally-binding agreements whether you're forming a business or selling a car.

Legal instrument - Wikipedia
https://en.wikipedia.org/wiki/**Legal**_instrument ▾
Jump to Electronic **legal documents** - Legal instrument is a legal term of art that is used for any formally executed written document that can be formally attributed to its author, records and formally expresses a legally enforceable act, process, or contractual duty, obligation, or right, and therefore evidences that act, process, or **agreement**.

Category:Legal documents - Wikipedia
https://en.wikipedia.org/wiki/Category:**Legal_documents** ▾
Pages in category "**Legal documents**". The following 154 pages are in this category, out of 154 total. This list may not reflect recent changes (learn more).

Legal documents Pakistan - legal forms, legal agreements, court ...
www.pk.path**legal**.com/**legaldocuments**/?parent=all&category=all ▾
Pathlegal has listed most of the **legal documents** from Pakistan for free include legal forms, legal

Figure 4. 5 Ranked retrieval example (Google)

4.4.1 Term Frequency and Weighting

Now we will assign weight to terms in documents. This weight will depend upon the occurrence of a term in a particular document. This weight will be used for computing score between query term *t* and document *d*. This weighting scheme is known as *term frequency*.

Term frequency $\mathrm{tf}_{t,d}$ of term *t* in a document *d* is defined as the number of times that *t* occurs in d [12]. It is possible that a term would appear much more times in long documents than in short ones. Therefore *tf* is often divided by document length as a way of *normalization.*

TF(t) = (No. of times term t appears in a document) / (Total number of terms in a document). The log frequency weight of a term *t* in document *d* is given by:

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Term frequency only retain the information of occurrence of a term in a document, this may cause some problem. Let's take an example: Suppose we have two documents, d1 = "John is quicker than Marry" and d2 = "Marry is quicker than John". In view of term frequency weighting, these documents are identical because *tf* of both documents is same but actually they are not. We will be handling this problem in IDF (Inverse Document Frequency).

Rare terms are more informative than frequent terms. Since rare terms do not get enough weight so they can be neglected. We want a high rate for rare terms. For this we will be using *document frequency*. Document frequency is defined to be number of documents in the collection that contains a term t.

It is obvious that frequent terms like 'that', 'from' etc. will have high weight and may have less importance but terms like 'murder', 'robbery' may have less weight but are more informative. Thus we need to scale down the frequent terms while scale up the rare terms by computing following [13]:

$$\mathrm{IDF}(t) = \log_{10} (N/\mathrm{df}_t).$$

N is total number of documents in collection and df_t is document frequency of a term *t*. One thing is to be noted that *idf* doesn't effect on ranking of query involving only one term. It will affect the ranking of documents for queries with at least two terms. Like if user enters a query like "murder", it will not affect the ranking of documents but if user enters query like "murder case", idf makes occurrence of 'murder' for much more in the final document ranking than occurrence of 'case'.

4.4.1.1 TF-IDF

Now we will combine the both definition, *term frequency* and *inverse document frequency* to get a composite weight for each term in a document.

The *tf-idf* weighting scheme assigns term *t* a weight in document *d* is given by

$$Tf\text{-}idf(t,d) = log(1 + tf(t,d)) \; * \; idf(t)$$

Tf-idf is best known weighting scheme in information retrieval [6]. It covers the biasness factor created from *tf(t)* that frequent terms will get more weight than rare terms. It increases with rarity of term in the collection. Also it increases with the number of occurrence within a document.

For normalization purpose we can also use log-frequency weighting. The log-frequency weight of a term t in a document d is given by:

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Inverse document frequency (*idf*) measures how important a term is. In *idf* all terms are considered equally important. In *idf* we give high weight to the rare terms [10].

4.4.1.2 SMART

SMART is a weighting scheme for query vs document. SMART stand for System for Mechanical Analysis and Retrieval of text. It is a combination of different weights. The mnemonic for representing a combination of weights takes the form *ddd.qqq* where the first triplet gives the term weighting of the document vector, while the second triplet gives the weighting in the query vector. The first letter in each triplet specifies the term frequency component of the weighting, the second gives the document frequency component, and the third gives the form of normalization used. It is quite usual to apply different normalization function to the query vector and the document vector. For example a weighting scheme is *lnc.ltc*

Document vector (*ddd*):

- *I represents that the document vector has log-term frequency.*
- *n represents no idf (inverse document frequency).*
- *c represents cosine similarity.*

Query vector (*qqq*):

- *l represents log-weighted term frequency.*
- *t represents idf weighting.*
- *c represents cosine similarity.*

4.4.1.3 BM25

Most of the weighting models use document term frequency (*tf*), the number of occurrences of the given query term in the given document, into consideration as a basic factor for weighting documents. The classical *tf.idf* weighting scheme formula is given by:

$$Tf\text{-}idf(t,d) = log(\ 1 + tf(t,d))\ \ * idf(t)$$

Where *Tf-idf (t,d)* is the weight of the document d for the term t. The above *tf-idf* formula is based on two basic principles of weighting:

- For a given term, the higher its frequency in the collection the less likely it is that it reflects much content [16].
- For a given term in a given document, if the term frequency (*tf*) is higher within a document, it means term carries more information within the document [16].

The term frequency (*tf*) is dependent on the document length. It needs to be normalized by using a technique called *term frequency normalization*. The reasons for the need of tf normalization are given below:

- The same term usually occurs repeatedly in long documents.
- A long document has usually a large size of vocabulary. Therefore it has a greater chance of mapping any query on it.

Without normalization, *tf-idf* can produce biased weights with respect to the document length. Since long document have a greater chance of mapping a query term on it, therefore long documents will be preferred. To avoid this biasness we will use *tf* normalization. A classical method of the *tf* normalization tuning is the *pivoted normalization* approach proposed by Singhal et.al [17]. In pivoted normalization we penalize a long document with a document length normalizer. As described earlier that a long document has a better chance to match any query so we need to normalize the document using pivot normalization. The formula for pivot length normalization is given below:

$$normalizer = 1 - b + b(\frac{|d|}{avg\_dl})$$

The reason why we are calling it pivot normalizer is that it normalizes documents around some pivot, and its pivot point is average document length. Average document length refers to finding length of all the document in the corpus and then calculating its average. *d* is document length, b is a free parameter between 0 and 1 i.e., b Є [0, 1]. The documents which are of greater length than average document length are penalized, i.e. it will be given less importance and the documents which are less than average document length are given some reward, reward in the sense that it will be given high priority.

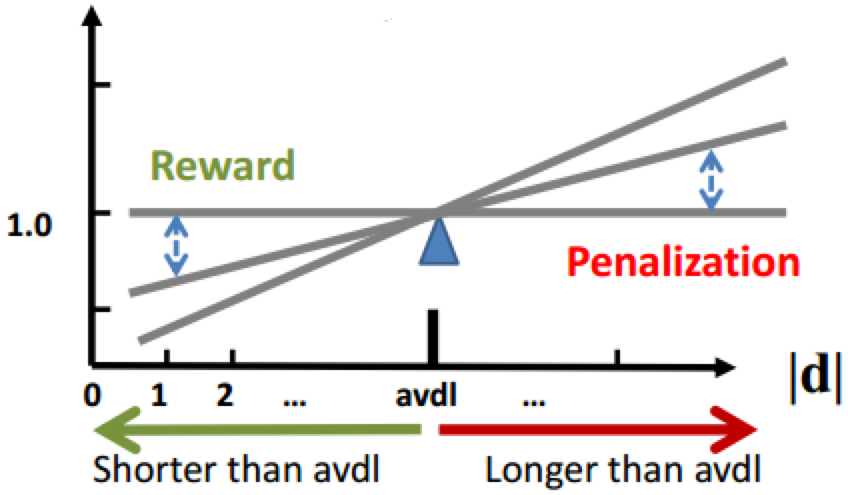


Figure 4. 6 *Pivot Normalization*

The formula for pivoted length normalization in vector space model is given by:

$$f(q,d) = \sum_{w\epsilon q^\frown d} c(w,q)\frac{\ln[1+\ln[1\ +c(w,d)]]}{1-b+b\frac{|d|}{avg_{dl}}}\ log\frac{N+1}{df(w)}$$

Where N is total number of documents in the collection, df (w) is total number of documents containing word *w*, c(w,d) is word count of *w* in document *d*, c(w,q) is word count of *w* in query q. In the denominator, there is pivot length normalizer discussed above.

BM25 is a retrieval function that ranks the documents against a query. It also uses *idf* as its base, removing the flaws of tf-idf stated above. For a given query q, the score of a document d is given by:

$$f(q,d) = \sum_{w\epsilon q^{\wedge}d} c(w,q)\frac{(\mathrm{k}+1)\mathrm{c}(\mathrm{w},\mathrm{d})}{c(w,d)+k(1-b+b\frac{|d|}{avg_{dl}})}\ log\frac{N+1}{df(w)}$$

In above stated formula *f(q,d)* is the score of a document *d* against query *q*, *c(w,q)* is word count of *w* in query *q*, *c(w,d)* is word count of *w* in document *d*, *avg_dl* is average document length, *d* is document length, N is total number of documents in the collection, *df(w)* is document frequency containing word *w*, k and b are free parameters usually chosen, in absence of an advanced optimization, k ϵ [0, +∞) and b ϵ[0,1].

4.4.2 Vector Space Model

One of the most commonly used model for information retrieval is vector space model. In the vector space model text is represented by a vector of terms [15].

4.4.2.1 Documents as Vectors

Suppose we have ha |V| dimensional vector space where V is the number of words i.e. vocabulary. The terms (words) are the axes of the space. The documents you can think of a vector from the origin pointing out some point in the space. So we now have a very high dimensional space, tens of millions of dimensions in a real system when you apply this to a web search engine. The crucial property of these vectors is that they are very sparse vectors means most of the entries are zero. Because each individual document only typically has a few hundred or thousand words in it.

4.4.2.2 Queries as Vector

So then if we have vector space of documents the question may arise, how do we handle query when a query comes in? The key idea is that we treat queries exactly the same way: that also will be vectors in the same space and then if we do that we can rank documents according to their proximity to the query in the space. Proximity corresponds to the similarity of vectors or we can say that reverse of distance between the vectors. We are doing this because we want to get away from the Boolean model which results either you are in or out and have a relative score, depending upon how well a document matches a query. We are going to rank more relevant documents higher than less relevant documents. Let's try all of this more precise.

So how can we formalize proximity in the space? The first attempt is to simply take the distance between the two points i.e. the distance between the end points of the vectors. And the standard way to do that in a vector space is by calculating the Euclidean distance between the points. But using Euclidean distance isn't a good idea because Euclidean distance is large for the vectors of different lengths [14].

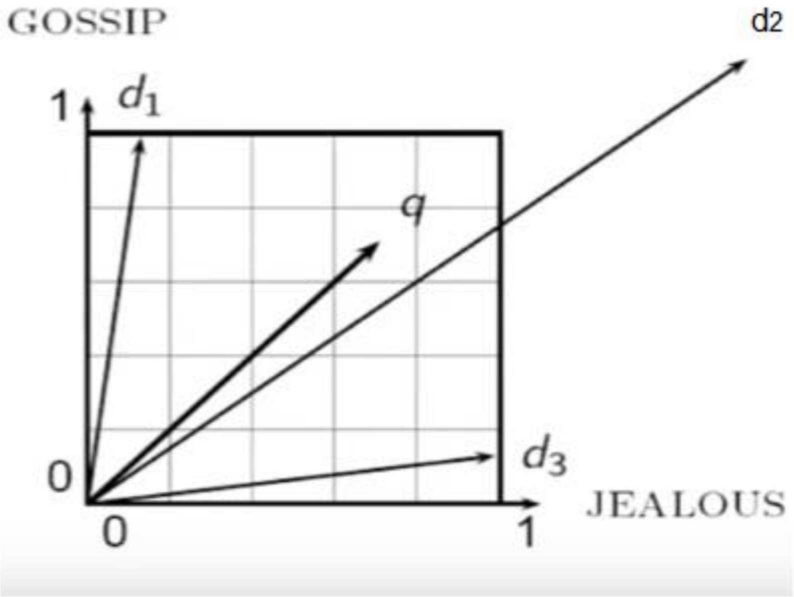


Figure 4. 7 Euclidean distance

Consider vector space shown in figure 4.7. So what we found is the distance between **q** and **d2** is large and particular is larger than the distance between **q** - **d3** and **q** - **d1**. But if we actually think of this in terms of information retrieval problem and try to look what is in our space that tells us that something is wrong there. In this small example, two word axes shown (as described earlier axes are term). On y-axis term is Gossip and on x-axis is Jealous and our query **q** is somewhat "gossip and jealous". If we look with our documents, what we find is **d1** seems have a lot to do with gossip and probably nothing to do with jealous and **d3** have a lot to do with jealous and nothing to do with gossip whereas **d2** seems just a kind of document that we want to get, one that have a lot to do with both gossip and jealous. So the term in the document **d2** are very similar to the term in **q** so we want to be saying that is actually the most similar document. So the solution to the problem is rather than talking about distance what we want to start looking at angle in the vector space. So the idea is to consider angle between the vectors rather than distance. Let's take and experiment:

Suppose we take a document ***d*** and appended to itself taking as a new document ***d'***. Semantically ***d*** and ***d'*** have the same content, they have the same information. But if we are working with a regular vector space with Euclidean distance, distance between the two documents will be quite large.

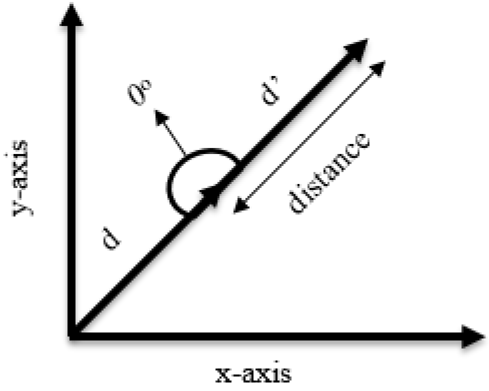


Figure 4. 8 *Euclidean Flaw Example*

So we don't want to do that instead what we want to notice the angle between the two vectors is zero corresponding to maximal similarity. So the idea is we got to rank documents according to angle between the document and the query. So following two notions are equivalent.

- Rank documents in *decreasing* order of the angle between query and document.
- Rank documents in *increasing* order of the cosine (query, document).

Cosine is a monotonically decreasing function for the interval [0^o, 180^o]. Therefore cosine serve as a kind of inverse of angle. Well that might still make it seem a strange thing to use instead we could've just taken the reciprocal of angle and negative of angle and that can serve our purpose too but it turns out that cosine measure is actually standard because it's actually very efficient way for calculating the similarly between query and document, any transcendental function other than cosine may take long time to compute the result.

4.4.2.3 Length Normalization

The starting point towards cosine similarity is getting idea about length of a vector and how to normalize length of a vector. A vector can be normalized by dividing each of its components by its length. For this we use the L_2 norm:

$$\|\boldsymbol{x}\|^2 = \sqrt{\sum_i x_i^2}$$

Dividing a vector by its L_2 norm makes it a unit vector on the surface of unit hyperspace around the origin. If we go back to example we had earlier of two documents **d** and **d'** (d appended to itself) one can see that these documents, if they are both length normalized will go back to exactly the same position and because of that once you length normalize vectors long and short documents will have comparable weights. So in cosine measure we do length normalization. For calculation cosine similarity we have a formula given below:

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\overset{\text{Dot product}}{\bullet}\vec{d}}{|\vec{q}||\vec{d}|} = \overbrace{\frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|}}^{\text{Unit vectors}} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

The function cos (q,d) is the cosine of the angle between the two documents. The way we do that, in the numerator we calculate dot product of query vector and document vector. The denominator is considering the lengths of vectors. So the way of calculating cosine similarity is to normalize the query and document vector separately and the finding the dot product of the normalized vectors. Where q_i is the tf-idf weight of term i in the query and d_i is the tf-idf weight of term i in the document.

For length normalized vectors, cosine similarity is simply the dot product of query and document vector.

$$cos(q,d) = \sum_{i=1}^{|V|}(q_i \; d_i)$$

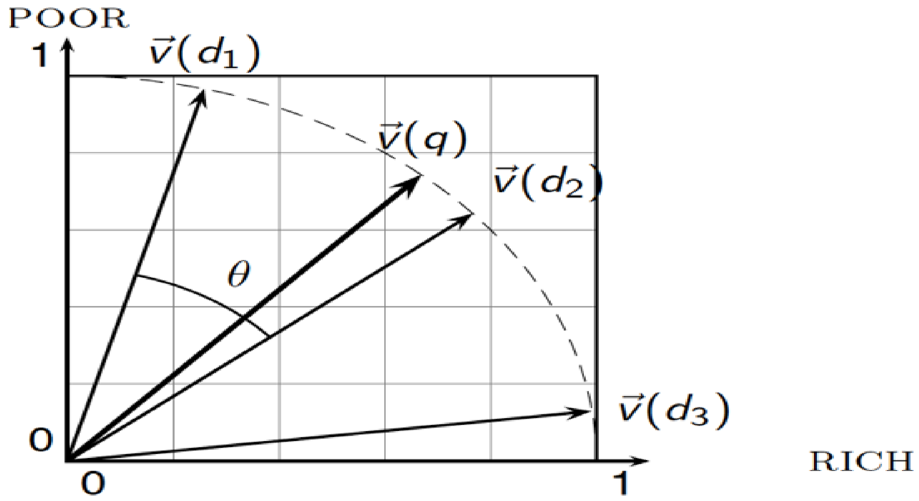


Figure 4. 9 *Cosine Similarity Hyperspace [7]*

In above vector space we can take any vector and we can map it down to the hyperspace by doing the length normalization. After doing that we will have all the vectors touching the surface of the hyperspace (the dotted line in figure 4.8). So then when we want to order documents by similarity to the query, we simply compute the cosine of the angle between the document and the query. Cosine will be highest for small angles since we described earlier that cosine is monotonically decreasing function. So if we order these documents in terms of cosine of the angle, the document that will be ranked first will be *d2* because it is making smallest angle with the query, the document which will be ranked second will be *d1* and then *d3* will be at last. Let's take an example for computing cosine similarity among some text.

For calculating the cosine similarity between two documents *d1* and *d2*, they are transformed in vectors as shown in the table below.

Each word in document defines a dimension in Euclidean space and the frequency of each word represents the dimension value. Now cosine similarity can be computed for two documents: "Murder person case" and "Missing person case person"

$$\frac{1.2 + 1.0 + 0.1 + 1.1}{\sqrt{1^2 + 1^2 + 0^2 + 1^2}\ \sqrt{2^2 + 0^2 + 1^2 + 1^2}} \cong 0.72$$

Table 4. 1 Cosine Similarity Example

	Person	Murder	missing	Case
d1	1	1	0	1
d2	2	0	1	1

4.5 Retrieval Performance Evaluation

For retrieval performance evaluation, one should consider the retrieval task that is to be evaluated. Retrieval task can be of two types, one is named as batch mode. In batch mode the user provides an information need in the form of query and receives the answer back, the other consists of a whole interaction session. In this session user specifies his information need through a series of interactive steps with the system. Further, the retrieval task could also comprise a combination of both of the two strategies. Since both batch and interactive query tasks are quite different processes therefore their evaluation is also different. In this chapter we will be discussing the evaluation of systems that processes batch mode only.

4.5.1 Recall and Precision

Let’s take an example query q from a test collection and its set R of relevant documents. Let |R| be the number of documents in this set. Assume that a given retrieval strategy process the information request q and generates a document answer set A. Let |A| be the number of documents in this set. Moreover, let |RA| be the number of documents in the intersection of the sets R and A. Figure 4.10 illustrates these sets.

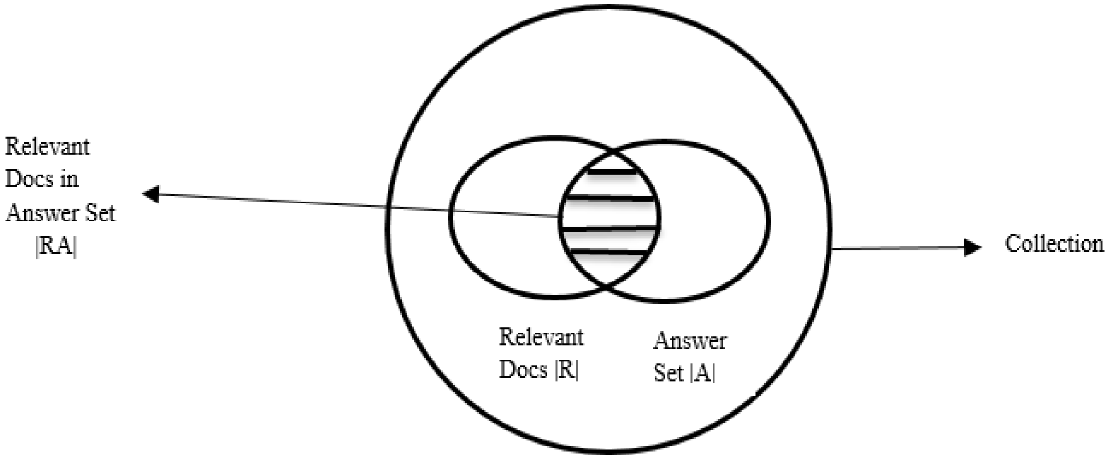


Figure 4. 10 Precision and Recall example illustration

4.5.1.1 Recall

Recall is the fraction of relevant documents (the set R) which have been retrieved.

Recall = |RA| / |R| OR Recall = Retrieved documents / Relevant documents

4.5.1.2 Precision

Precision is the fraction of retrieved documents (the set A) which are relevant.

Precision = $|RA|/|A|$ OR Recall = Relevant documents / Retrieved documents.

Let's take an example to explain the concept of recall and precision. Assume that a set R_q containing the relevant documents for a query q has been identified. Moreover assume that the set R_q is composed of the following documents.

$R_q = \{ d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123} \}$

There are ten documents which are relevant to the query q.

Assume an algorithm returns, for the query q, a ranking of the documents in the answer as follows.

Ranking for query q:

1. d_{123}
2. d_{84}
3. d_{56}
4. d_6
5. d_8
6. d_9
7. d_{511}
8. d_{129}
9. d_{187}
10. d_{25}
11. d_{38}
12. d_{48}
13. d_{250}
14. d_{113}
15. d_3

The documents that are relevant to the query q are underlined. If we examine this ranking, starting from the top documents, one can observe the following points.

- The document d_{123} which is ranked as number 1 is relevant. This document have 10% of all the relevant documents in the set R_q. Thus we can calculate precision by using formula:

  Precision = Relevant / Retrieved

  Precision = 1 / 1 *100 = 100%

Recall = Retrieved / Total Relevant Docs

Recall = 1 / 5 *100 = 20%

- The document d_{56} which is ranked 3rd is the next relevant document. At this point we have

Precision = 2/3 * 100 = 66.67%

Recall = 2 / 5 * 100 = 40% (means 2 of the 5 relevant doc seen)

Thus if we proceed with our examination of the ranking generated, we can plot a graph of precision vs recall shown in figure 4.11. The precision at levels of recall higher than 50% drops to 0 because not all relevant documents have been retrieved. This precision versus recall is usually base on 11 standard recall instead of 10 which are 0%, 10%, 20% …, 100%.

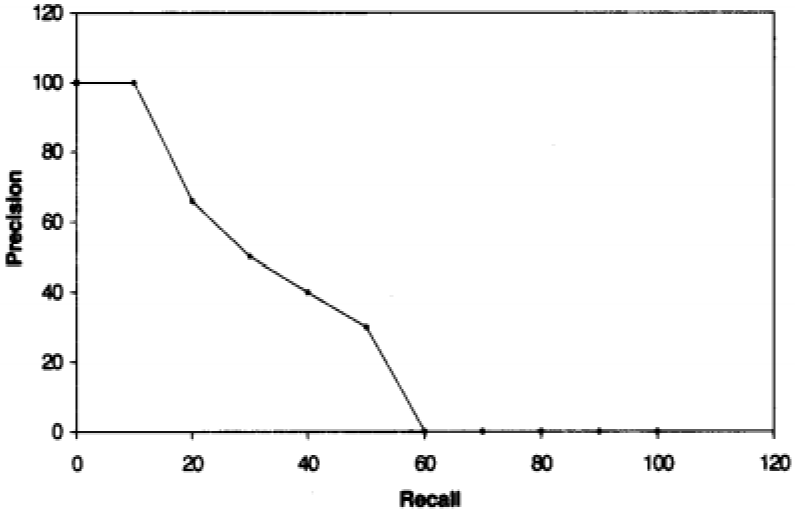


Figure 4. 11 Precision vs Recall

In the above example, the precision and recall figures are for a single query, but what for several distinct queries? In this case, for each query a distinct precision vs recall curve is generated. To evaluate the retrieval performance of an algorithm over all test queries, we average the precision figures at each recall level as follows.

$$P\ (r) = \sum_{i=0}^{Nq} \frac{\mathrm{Pi(r)}}{Nq}$$

Where P(r) is the average precision level at point r, Nq is the number of queries, Pi(r) is the precision recall level r for the i-th query.

Since recall and precision are not always the most appropriate measure for evaluating retrieval performance, so some of the alternative measures are discussed below.

4.5.1.1 The Harmonic Mean (F- Measure)

One possible solution for having more appropriate measure for evaluating retrieval performance of search engine is to get a mashup kind of precision and recall measures. To get the mashup we take harmonic mean F of precision and recall, which is computed as:

$$F(i) = \frac{2}{\frac{1}{r(i)}+\frac{1}{p(i)}}$$

where r(i) is the recall for the i-th document in the ranking, p(i) is the precision of the i-th document in the ranking and F(i) is the harmonic mean for recall r(i) and precision p(i). The function F have value in the interval [0, 1]. If F = 0, it means that no relevant document have been retrieved and if F = 1 says that all the documents are relevant. The function F will have higher value for the higher value of both recall and precision. Therefore it finds the best possible compromise between precision and recall.

Chapter 5

System Design

5.1 Introduction

In this chapter we will start designing our product. System design for our system includes sequence diagram and class diagram. There are three sequence diagrams, each for client side, server side and for query generation. Since we got two major functions of our product, one is search engine and second is automated query formulation so two class diagrams were made. One for search engine and second for automated query formulation.

5.2 Sequence Diagrams

Figure 5.1 represents the sequence diagram for client side.

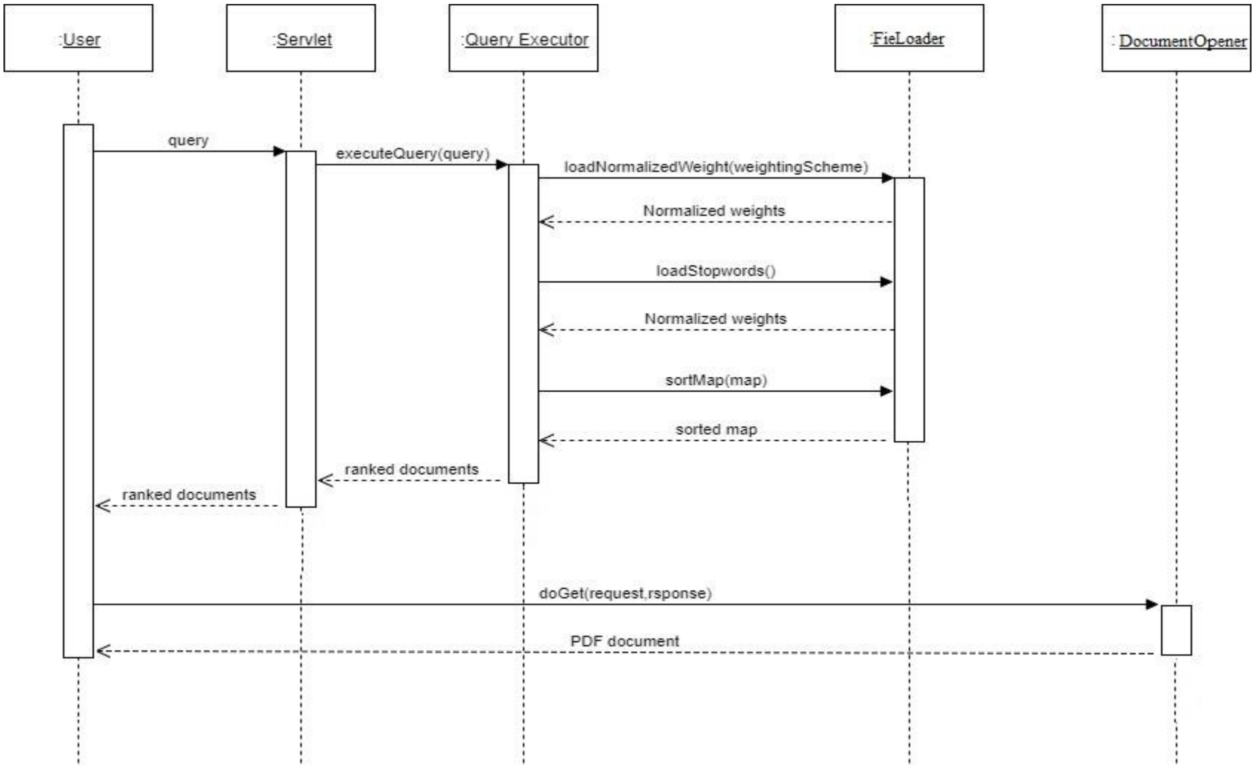


Figure 5. 1 *Client Side Sequence Diagram*

Figure 5.2 represents sequence diagram for server side of our search engine.

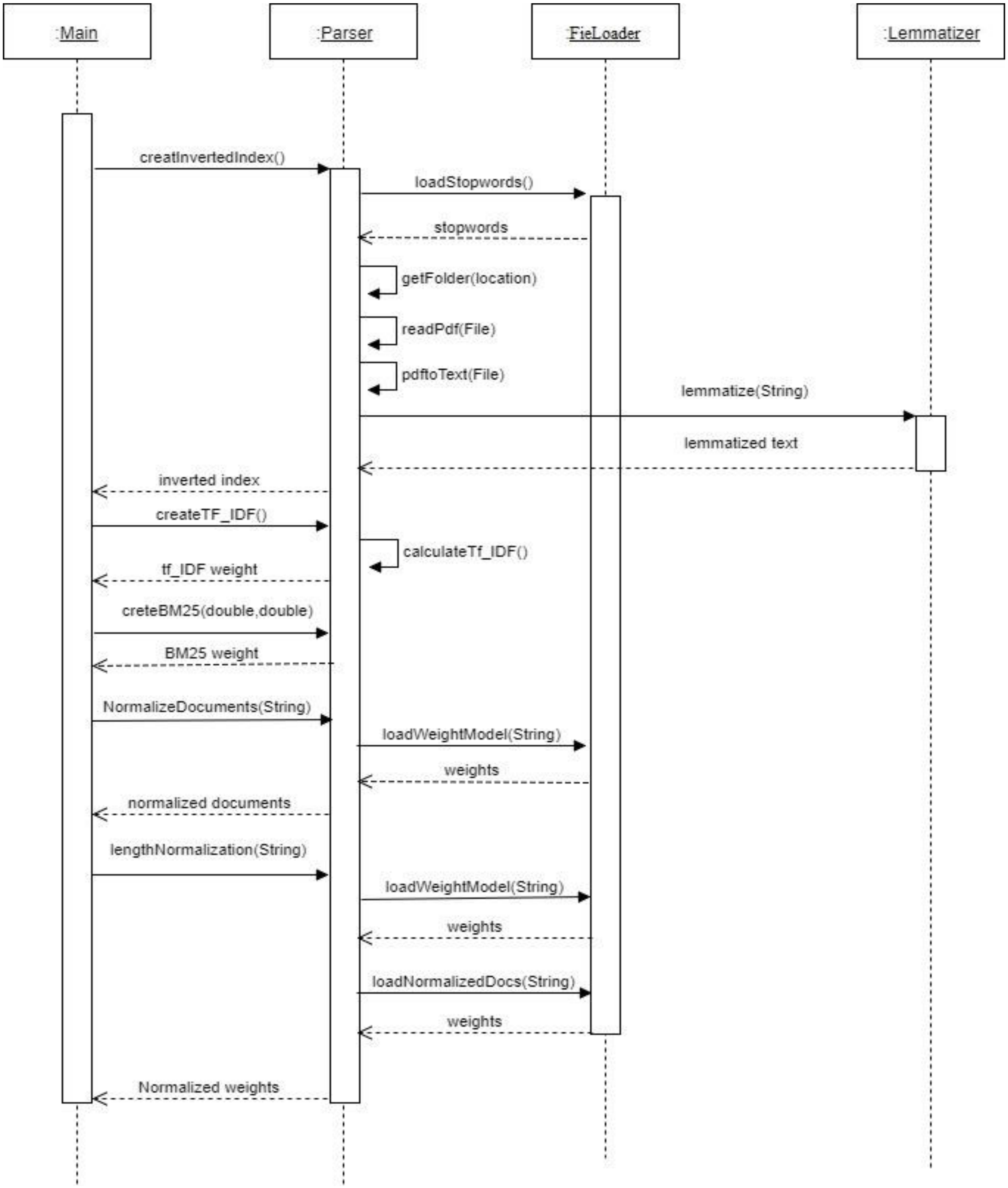


Figure 5. 2 *Sequence Diagram for Server Side*

Figure 5.3 shows sequence diagram for query formulation.

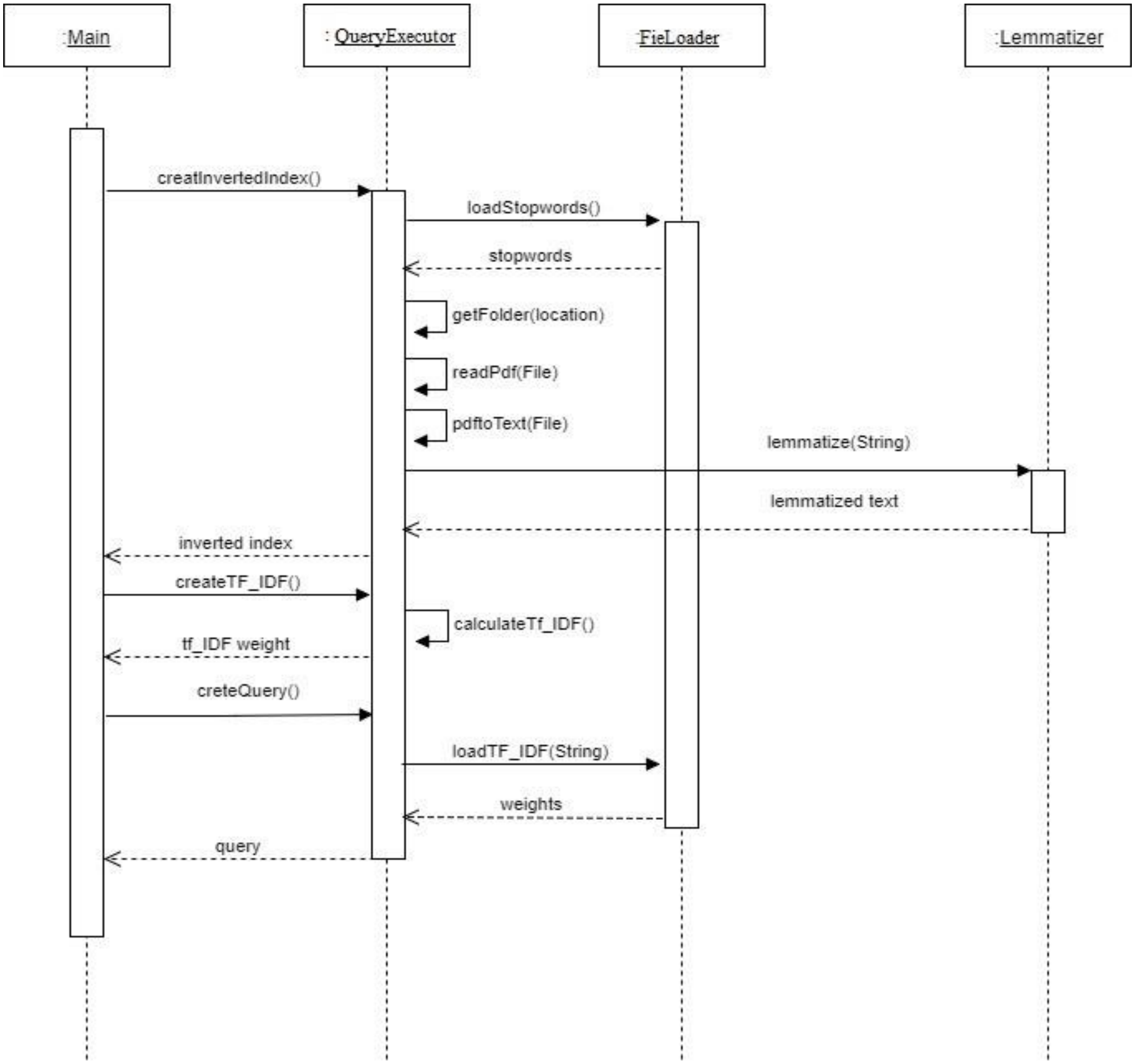


Figure 5. 3 Sequence Diagram for Query Formulation

5.3 Class Diagrams

Since there two class diagrams so figure 5.4 shows class diagram for search engine and figure 5.5 shows class diagram for query formulation.

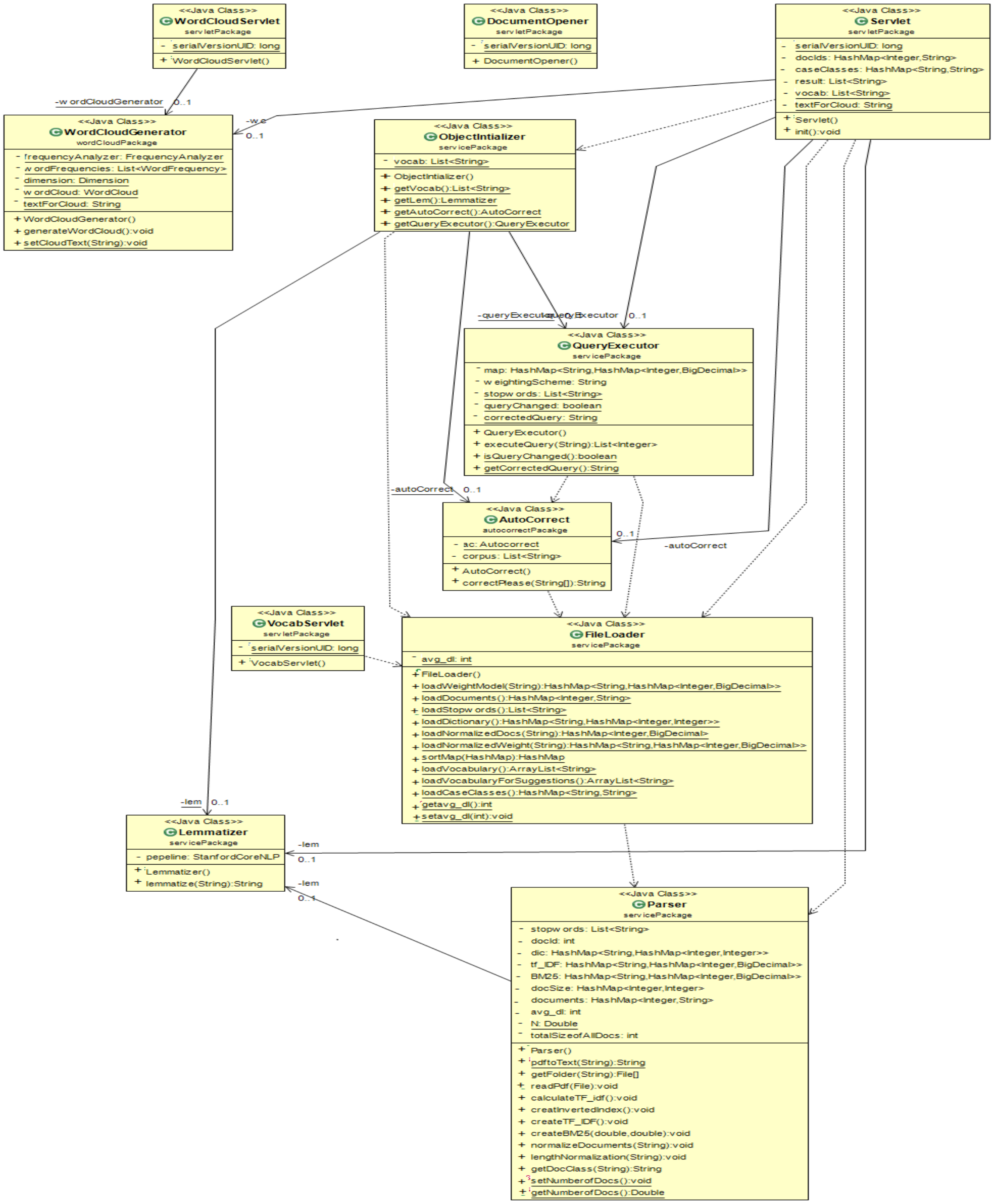

<<Java Class>>
WordCloudServlet
servletPackage
- serialVersionUID: long
+ WordCloudServlet()
<<Java Class>>
DocumentOpener
servletPackage
- serialVersionUID: long
+ DocumentOpener()
<<Java Class>>
Servlet
servletPackage
- serialVersionUID: long
- docIds: HashMap<Integer,String>
- caseClasses: HashMap<String,String>
- result: List<String>
- vocab: List<String>
- textForCloud: String
+ Servlet()
+ init():void
-wordCloudGenerator
0..1
-wc
0..1
<<Java Class>>
WordCloudGenerator
wordCloudPackage
- frequencyAnalyzer: FrequencyAnalyzer
- wordFrequencies: List<WordFrequency>
- dimension: Dimension
- wordCloud: WordCloud
- textForCloud: String
+ WordCloudGenerator()
+ generateWordCloud():void
+ setCloudText(String):void
<<Java Class>>
ObjectIntializer
servicePackage
- vocab: List<String>
+ ObjectIntializer()
+ getVocab():List<String>
+ getLem():Lemmatizer
+ getAutoCorrect():AutoCorrect
+ getQueryExecutor():QueryExecutor
-queryExecutor
-queryExecutor
0..1
0..1
<<Java Class>>
QueryExecutor
servicePackage
- map: HashMap<String,HashMap<Integer,BigDecimal>>
- weightingScheme: String
- stopwords: List<String>
- queryChanged: boolean
- correctedQuery: String
+ QueryExecutor()
+ executeQuery(String):List<Integer>
+ isQueryChanged():boolean
+ getCorrectedQuery():String
-autoCorrect
0..1
<<Java Class>>
AutoCorrect
autocorrectPacakge
- ac: Autocorrect
- corpus: List<String>
+ AutoCorrect()
+ correctPlease(String[]):String
0..1
-autoCorrect
<<Java Class>>
VocabServlet
servletPackage
- serialVersionUID: long
+ VocabServlet()
<<Java Class>>
FileLoader
servicePackage
- avg_dl: int
+ FileLoader()
+ loadWeightModel(String):HashMap<String,HashMap<Integer,BigDecimal>>
+ loadDocuments():HashMap<Integer,String>
+ loadStopwords():List<String>
+ loadDictionary():HashMap<String,HashMap<Integer,Integer>>
+ loadNormalizedDocs(String):HashMap<Integer,BigDecimal>
+ loadNormalizedWeight(String):HashMap<String,HashMap<Integer,BigDecimal>>
+ sortMap(HashMap):HashMap
+ loadVocabulary():ArrayList<String>
+ loadVocabularyForSuggestions():ArrayList<String>
+ loadCaseClasses():HashMap<String,String>
+ getavg_dl():int
+ setavg_dl(int):void
-lem
0..1
<<Java Class>>
Lemmatizer
servicePackage
- pepeline: StanfordCoreNLP
+ Lemmatizer()
+ lemmatize(String):String
-lem
0..1
-lem
0..1
<<Java Class>>
Parser
servicePackage
- stopwords: List<String>
- docId: int
- dic: HashMap<String,HashMap<Integer,Integer>>
- tf_IDF: HashMap<String,HashMap<Integer,BigDecimal>>
- BM25: HashMap<String,HashMap<Integer,BigDecimal>>
- docSize: HashMap<Integer,Integer>
- documents: HashMap<Integer,String>
- avg_dl: int
- N: Double
- totalSizeofAllDocs: int
+ Parser()
+ pdftoText(String):String
+ getFolder(String):File[]
+ readPdf(File):void
+ calculateTF_idf():void
+ creatInvertedIndex():void
+ createTF_IDF():void
+ createBM25(double,double):void
+ normalizeDocuments(String):void
+ lengthNormalization(String):void
+ getDocClass(String):String
+ setNumberofDocs():void
+ getNumberofDocs():Double

Figure 5. 4 *Class Diagram for Search Engine*

Figure 5.5 shows the class diagram for query formulation.

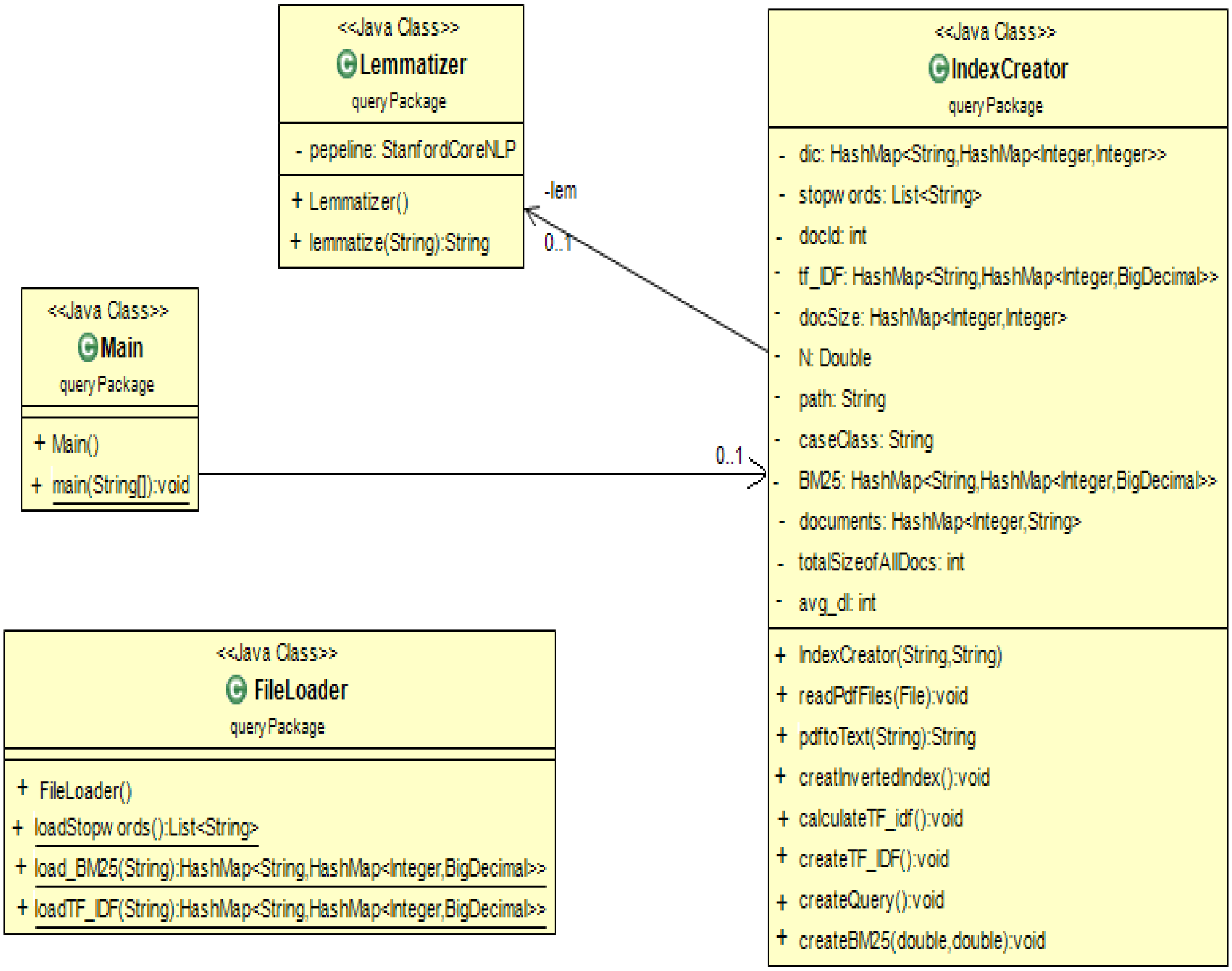

<<Java Class>>
Lemmatizer
queryPackage
- pepeline: StanfordCoreNLP
+ Lemmatizer()
+ lemmatize(String):String
-lem
0..1
<<Java Class>>
IndexCreator
queryPackage
- dic: HashMap<String,HashMap<Integer,Integer>>
- stopwords: List<String>
- docId: int
- tf_IDF: HashMap<String,HashMap<Integer,BigDecimal>>
- docSize: HashMap<Integer,Integer>
- N: Double
- path: String
- caseClass: String
- BM25: HashMap<String,HashMap<Integer,BigDecimal>>
- documents: HashMap<Integer,String>
- totalSizeofAllDocs: int
- avg_dl: int
+ IndexCreator(String,String)
+ readPdfFiles(File):void
+ pdftoText(String):String
+ creatInvertedIndex():void
+ calculateTF_idf():void
+ createTF_IDF():void
+ createQuery():void
+ createBM25(double,double):void
<<Java Class>>
Main
queryPackage
+ Main()
+ main(String[]):void
0..1
<<Java Class>>
FileLoader
queryPackage
+ FileLoader()
+ loadStopwords():List<String>
+ load_BM25(String):HashMap<String,HashMap<Integer,BigDecimal>>
+ loadTF_IDF(String):HashMap<String,HashMap<Integer,BigDecimal>>

Figure 5. 5 *Class Diagram for Query Formulation*

Chapter 6

Legal Document Collection

6.1 Introduction

In this chapter we will be moving our discussion forward towards document collection phase, this phase is a first step towards search engine implementation phase. Since we are going to build a 'Legal Document Search Engine', the first step will be collection of legal documents. For this, there will be a need resource(s) from where we can collect legal documents also the method used in the collection phase will be described. Moreover we will classify each legal document to some class manually i.e., civil petition cases class, criminal appeal cases class and human right cases etc. and will describe what kind of cases these classes will contain i.e. definition of the classes.

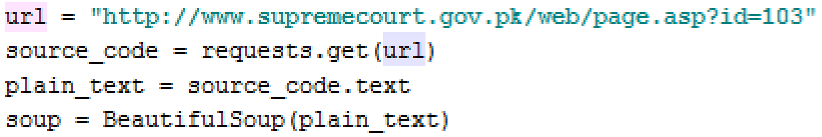
6.2 Collecting Legal Documents

Before collecting documents we must be clear what we are looking for i.e., what documents are referred as legal documents? As described earlier in section 2.2, a legal document is a document that grants some rights and states some contract based relationship. These documents involve some text. Some of these texts may be a part of legislation, some of them consist of contracts, police statements, official pleadings, warrants and court decisions. The most commonly legal document used in information retrieval systems are legislation and court decisions.

Now after the understanding of a legal document we start to collect legal documents. We will be dealing with court decisions of Supreme Court of Pakistan only. There are a huge number of court decision made by Supreme Court of Pakistan but for this search engine we will be considering only 500 decision to be indexed. These decisions can be found at official website of Supreme Court of Pakistan http://www.supremecourt.gov.pk. These decision are in portable document format (pdf). There were two ways of getting those documents, by downloading each document one by one and the other option was to build a web scraper. We preferred making a web scrapper.

6.2.1 Web Scraping

Web scraping also known as web data extraction, is an automated software technique of extracting information from web. We made web scraper in python using Beautiful Soup library in *pyCharm* IDE. This library is used for pulling data out of HTML and XML files. The general idea for web scraping is to extract data from web and convert it to a suitable format which can be analysed. We began explaining web scraper using following code:



```
url = "http://www.supremecourt.gov.pk/web/page.asp?id=103"
source_code = requests.get(url)
plain_text = source_code.text
soup = BeautifulSoup(plain_text)
```

Figure 6. 1 *Beautiful Soup Object*

In the above code *request* method sends a GET request to the URL http://www.supremecourt.gov.pk/web/page.asp?id=103 (Supreme Court of Pakistan) and gets the text of the listed URL. After that whole source code is converted into text then the *plain_text* is formatted in Beautiful Soup object. The *soup* object contains all of the HTML in the original document. Beautiful Soup is essentially a set of wrapper functions that make it simple to select common HTML elements like <p> and <a> tags [19]. After having all of the HTML document in a formatted manner we than traverse through the information we need. In the website there was a hyperlink which contains another hyperlink and the second hyperlink was our targeted file which is to be downloaded. Diagram below can explain the architecture.

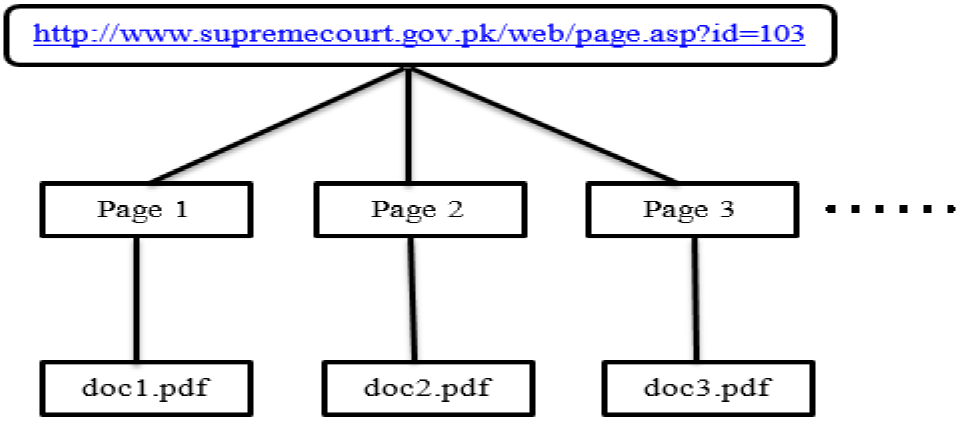
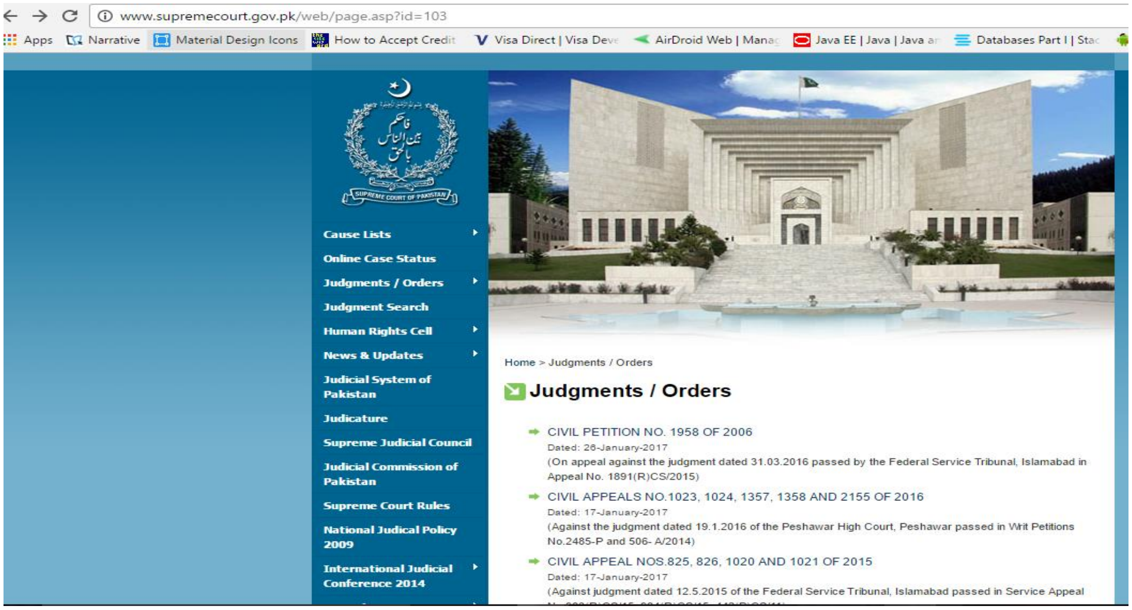


Figure 6. 2 *Supreme Court of Pakistan Website Structure*

So for this purpose we had to go to doc1.pdf, download it using command *wget.download(href)* and then go back to main page jump to page 2, download doc2.pdf and then go back to main page and the process goes on until all the documents are downloaded. Below is the picture shown of the main page containing the links to the documents.



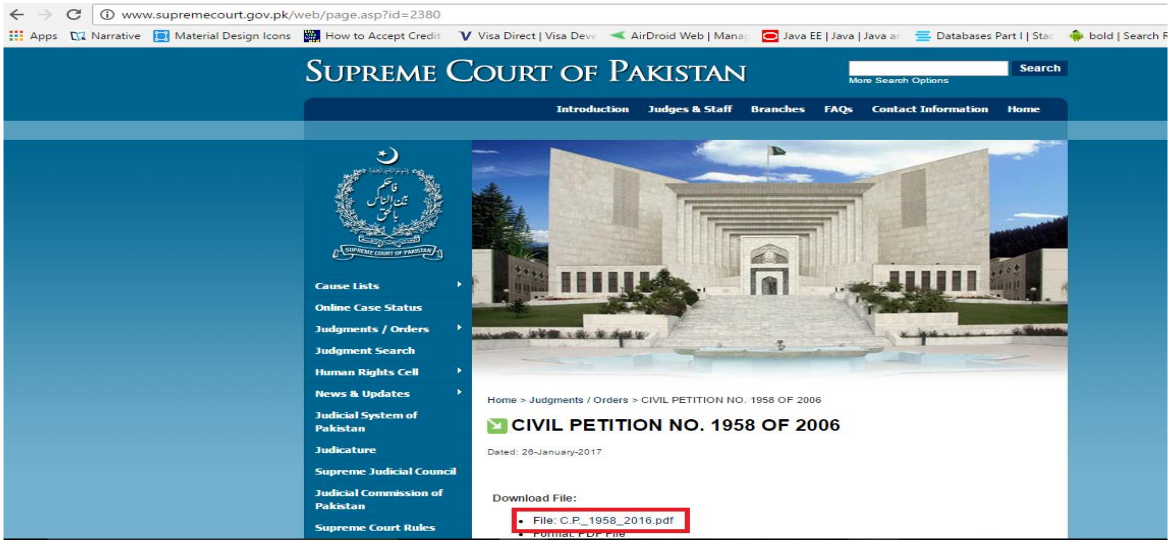
Home > Judgments / Orders

Judgments / Orders

- CIVIL PETITION NO. 1958 OF 2006
  Dated: 26-January-2017
  (On appeal against the judgment dated 31.03.2016 passed by the Federal Service Tribunal, Islamabad in Appeal No. 1891(R)CS/2015)
- CIVIL APPEALS NO.1023, 1024, 1357, 1358 AND 2155 OF 2016
  Dated: 17-January-2017
  (Against the judgment dated 19.1.2016 of the Peshawar High Court, Peshawar passed in Writ Petitions No.2485-P and 506- A/2014)
- CIVIL APPEAL NOS.825, 826, 1020 AND 1021 OF 2015
  Dated: 17-January-2017
  (Against judgment dated 12.5.2015 of the Federal Service Tribunal, Islamabad passed in Service Appeal

Figure 6. 3 Main page

The page that contains the document in .pdf extension is shown below in figure 6.4.



Home > Judgments / Orders > CIVIL PETITION NO. 1958 OF 2006

CIVIL PETITION NO. 1958 OF 2006

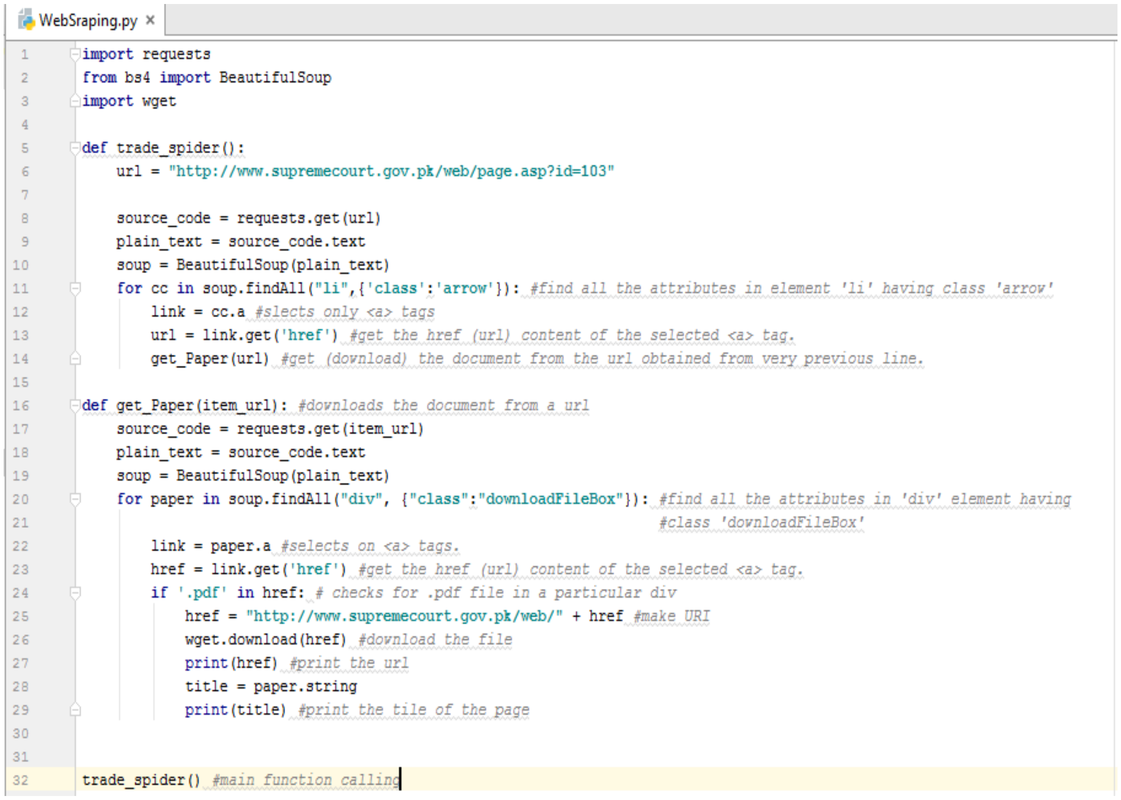
Dated: 26-January-2017

Download File:

- File: C.P._1958_2016.pdf
- Format: PDF File

Figure 6. 4 Page Containing Document .pdf

So this concludes the document collection phase. We described the sources from where we collected legal documents and then described the way all the documents are collected. We described what technique used for getting legal documents (court decision) from the source. The only source from where all the documents are collected is an official website of Supreme Court of Pakistan. Python programming language is used for making web scraper. The IDE used for python programming is pyCharm 2016.3.1. One can get confused in the difference between web crawler and web scraper. Web crawler also known as web spider, is a software program that visits the websites and reads their pages and other information to build entries for a search engine index. Whereas web scraping also known as web data extraction, is an automated software technique of extracting information from web for example to download the files from web and reading the related information. I made web scraper for downloading the legal documents from the website of Supreme Court of Pakistan. At the end of this discussion we are showing the code for making a web scraper in python in the figure given below.



```python
import requests
from bs4 import BeautifulSoup
import wget

def trade_spider():
    url = "http://www.supremecourt.gov.pk/web/page.asp?id=103"

    source_code = requests.get(url)
    plain_text = source_code.text
    soup = BeautifulSoup(plain_text)
    for cc in soup.findAll("li",{'class':'arrow'}): #find all the attributes in element 'li' having class 'arrow'
        link = cc.a #slects only <a> tags
        url = link.get('href') #get the href (url) content of the selected <a> tag.
        get_Paper(url) #get (download) the document from the url obtained from very previous line.

def get_Paper(item_url): #downloads the document from a url
    source_code = requests.get(item_url)
    plain_text = source_code.text
    soup = BeautifulSoup(plain_text)
    for paper in soup.findAll("div", {"class":"downloadFileBox"}): #find all the attributes in 'div' element having
                                                                   #class 'downloadFileBox'
        link = paper.a #selects on <a> tags.
        href = link.get('href') #get the href (url) content of the selected <a> tag.
        if '.pdf' in href: # checks for .pdf file in a particular div
            href = "http://www.supremecourt.gov.pk/web/" + href #make URI
            wget.download(href) #download the file
            print(href) #print the url
            title = paper.string
            print(title) #print the tile of the page


trade_spider() #main function calling
```

Figure 6. 5 *Web Scraping Code*

6.3 Document Classification

There are almost 500 documents that are collected in document collection phase. Now these documents have to be arranged in such a way that each of them belong to some class. This classification is done manually. After reading all the document there are two broader characteristics of court decision identified, described below.

- **Appeal**
  When court gives decision about a particular case, there is a right of appeal to both of the parties, the appellant and the respondents, to request for reviewing the case in higher court. Any of the two parties can appeal against a decision if they are not satisfied with it.
- **Petition**
  A petition is a way of getting your voice heard in Parliament. It's a written request, asking the House of Commons, or the Government, to take action on something. One will prepare petition and get people who agree with you to sign it. More formally "a petition is a written application from a person, public official or a group of person asking that some authority to be exercised to provide right, favours and relief" [20].

In our collection of legal documents, following are the case classes that are identified to be indexed with a brief explanation.

- **Civil Cases Appeal**
  Appeal can be made on civil cases. These cases involves dispute among two civilians or two parties. When a party files a complaint about any other party, they are involved in a civil case. Some examples of civil cases are contract violation, child custody and divorce matters etc. Total of 150 court decisions are collected of this category.
- **Civil Shariat Cases**
  Shariat is an Islamic law derived from Quran and Hadith. The cases involve in Shariat law are Islamic law related issues for example blasphemy (Toheen e Risaalt). Only 1 decision collected of this category.
- **Criminal Cases Appeal**
  Appeal can be made on Criminal cases. These cases deals with those acts that are criminal of offensive. Example of these cases are murder cases etc. Total of 30 decisions are collected of this category.
- **Intra Court Appeal**
  Intra court appeal is an appeal that is made before the bench of two or more judges against a decision made by a single judge. Only 3 decisions are collected of this category.

- **Constitutional Petition**
  Petitions that are submitted to enforce those fundamental rights which are provided by constitution are called constitutional petitions. Fundamental rights include right to live, right to business etc. Total of 60 decisions collected of this category.
- **Criminal Petition**
  Petition can also be made on criminal cases. These cases deals with the criminal or offensive act. Total of 40 decision collected of this category.
- **Human Right Cases**
  This type of case involve human right issues. For example action taken on unprecedented load shedding in the country. Total of 25 decision collected of this category.
- **Civil Miscellaneous Cases**
  Appeals made on the decision made by lower court in higher court are referred as miscellaneous cases. For example: if a decision is made by high court and one of the party isn't satisfied with the decision, they can appeal to review the decision in higher court like Supreme Court. These type of cases in higher court are referred as miscellaneous cases. Total of 25 decision collected of this category.
- **Suo Moto Cases.**
  Suo Moto is a Latin word that means 'on its own motion'. When government of court official acts of its own initiative. It is not a result of party asking. A recent example is: A three-member bench of Supreme Court of Pakistan headed by Chief Justice of Pakistan heard a suo moto case regarding the 10 year old child torture [Dawn news Jan 18, 2017]. Total of 30 decision are collected on sue moto cases.
- **Constitution of Pakistan**
  Rules and regulation for smooth running of a state are defined in this document.
- **Cyber Crime Bill of Pakistan**
  Prevention of electronic crimes bill was passed in National Assembly in 2016 in which punishment of unauthorized access are defined.

So this concludes the document classification phase. In this phase we described what the main characteristics of court decision are. Moreover different classes are stated with a brief explanation of each class. With this discussion our documentation phase completes. The next phase will be implementation phase of "Legal Document Search Engine".

Chapter 7

Implementation

7.1 Introduction

In this phase we will be heading towards implementation of our search engine. We will be providing tools that are used in implementation phase and language used. Word Cloud also has been made for this system. It will make a word cloud for the top ten documents using frequency analyser i.e., the word which occurs the most will be given a larger font and then the font will vary according to the frequency of a particular word. This word cloud is dynamic i.e., it is regenerated in response to every query. Furthermore, auto-corrector also has been made for this system. It will find the best match of a given word against the vocabulary and will execute accordingly after correcting it, if needed. Levenshtein minimum edit distance algorithm is used for auto-correction. Moreover, for the persuasion of the user, system provides suggestion for the typed word. These suggestions are generated from the vocabulary of the system.

7.2 Language Selection

- **JAVA**
  JAVA is used for implementation of Legal Document Search Engine. Java is an object oriented language that enable less dependencies in implementation. Since it is a web based application so JSPs (Java Server Pages) and Servlets is used.
  - **Servlets**
    A servlet is a Java class that runs in a Java-enabled server. An HTTP servlet is a special type of servlet that handles an HTTP request and provides and HTTP response, usually in the form of and HTML page.
  - **Java Server Pages**
    Java Server Pages (JSPs) are a Sun Microsystems specification for combining Java with HTML to provide dynamic content for Web pages. When you create dynamic content, JSPs are more convenient to write than HTTP servlets because they allow you to embed Java code directly into your HTML pages, in contrast with HTTP servlets, in which you embed HTML inside Java code.
- **HTML**
  HTML is hypertext mark-up language is used for developing front end of our web application. Bootstrap is also used with it in order to make the user interface more attractive.

- **JavaScript**
  JavaScript is an object oriented programming language used to create interactive effects within web browser.

  - **JQuery**
    It is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browser.

7.3 Tools

Tools that are used in the implementation are:

- Notepad++ , Sublime
- Apache Tomcat 7.0

7.4 IDEs

Eclipse IDE will be used for the implementation of the system. Two different version will be used.

i. **Eclipse Java Mars 1.0**
This IDE will used for query generation and tasks related query generation like calculating F-measures, average precision, mean average precision etc.

ii. **Eclipse JEE Neon**
This IDE will be used for making front end of our system and execution of query and generating results (ranked results) and rest of the function. Below are some of the screen shots of the front end of the Legal Documents Search Engine.

Figure 7.1 shows the main page of our search engine.

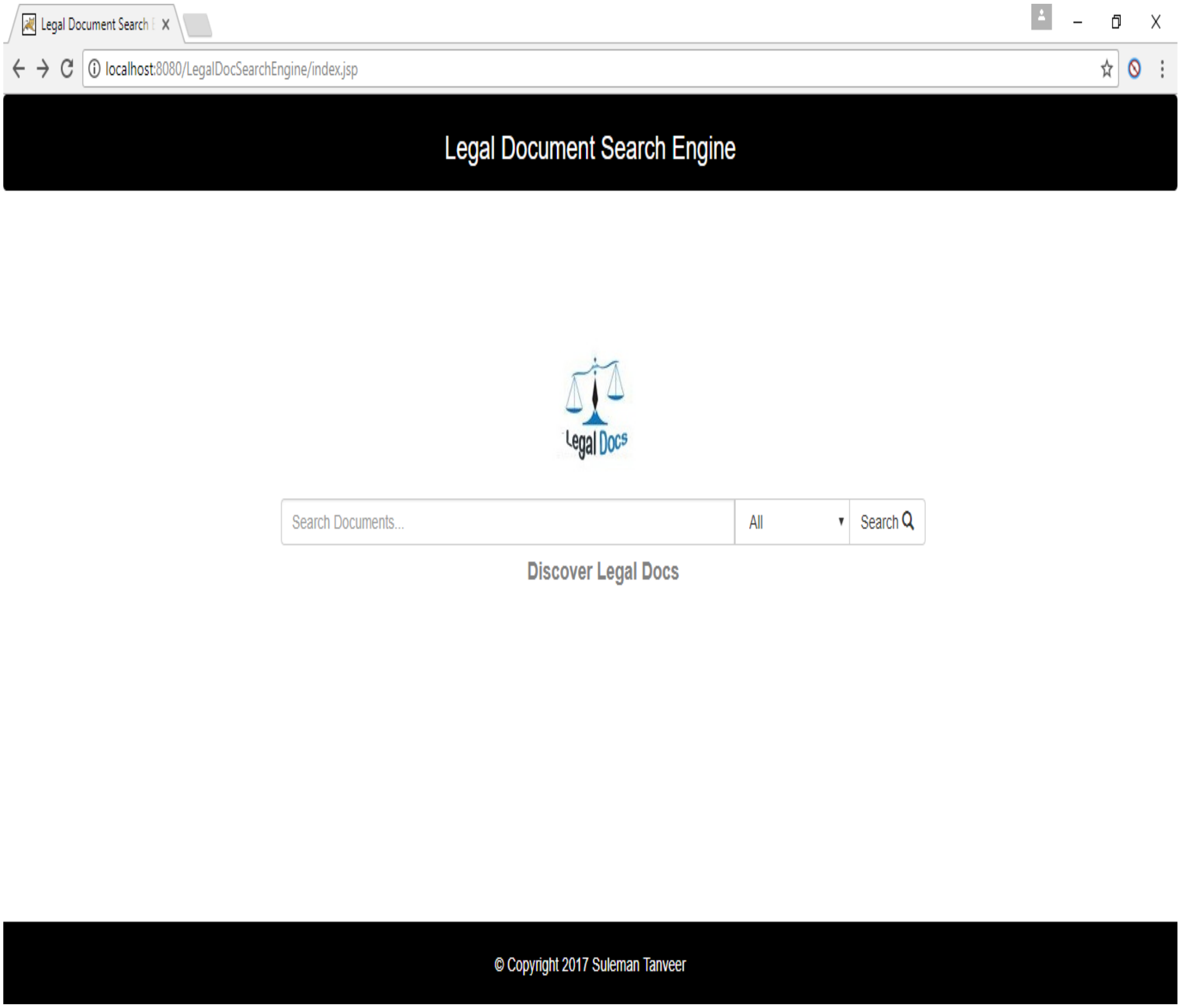
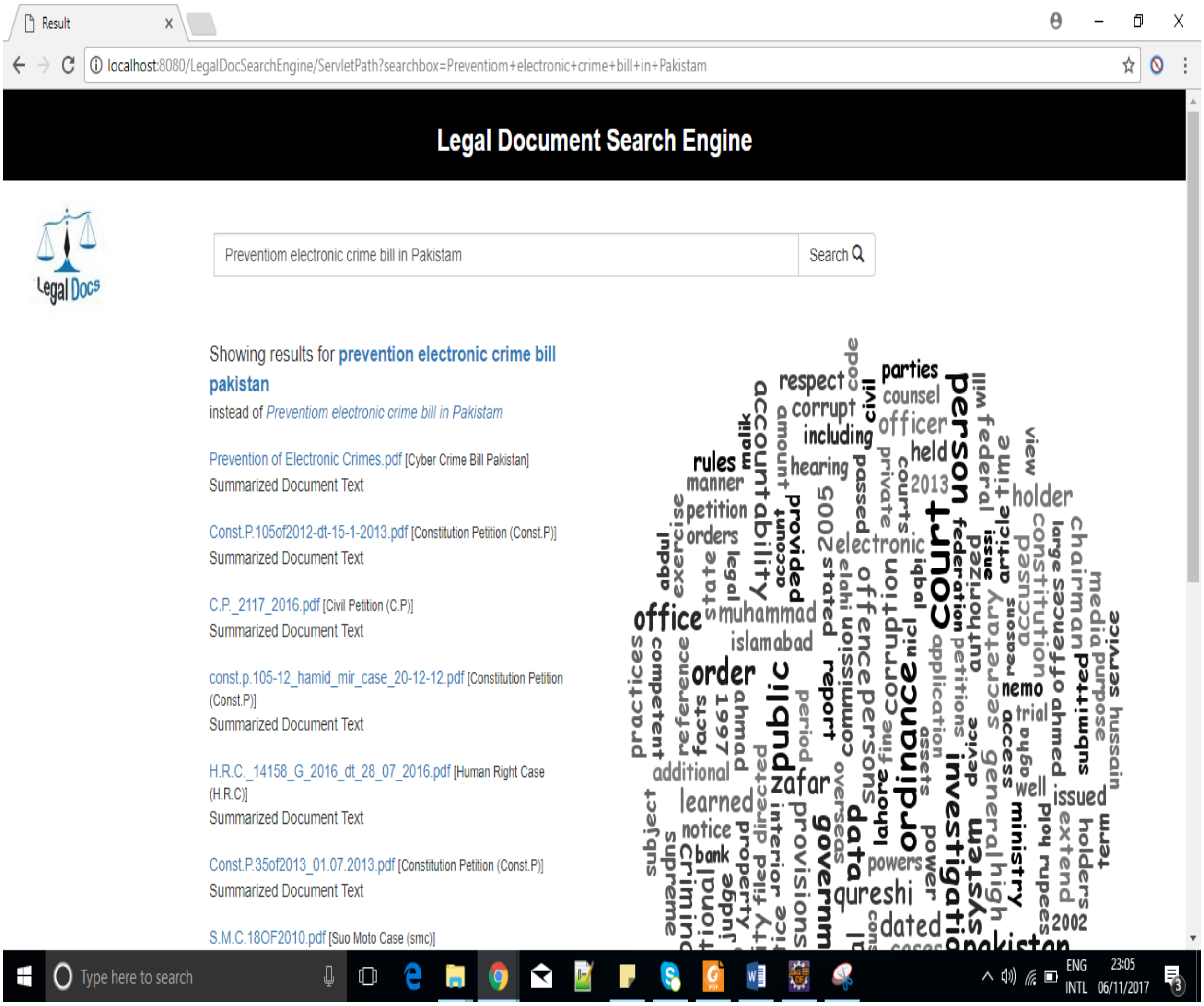

Legal Document Search
localhost:8080/LegalDocSearchEngine/index.jsp
Legal Document Search Engine
Legal Docs
Search Documents...
All
Search
Discover Legal Docs
© Copyright 2017 Suleman Tanveer

Figure 7. 1 *Front end main page*

When user enters some query, a list of ranked document is returned. Figure 7.2 shows the list that is returned on the response of a query.



Legal Document Search Engine

Preventiom electronic crime bill in Pakistam

Showing results for **prevention electronic crime bill pakistan**
instead of *Preventiom electronic crime bill in Pakistam*

Prevention of Electronic Crimes.pdf [Cyber Crime Bill Pakistan]
Summarized Document Text

Const.P.105of2012-dt-15-1-2013.pdf [Constitution Petition (Const.P)]
Summarized Document Text

C.P._2117_2016.pdf [Civil Petition (C.P)]
Summarized Document Text

const.p.105-12_hamid_mir_case_20-12-12.pdf [Constitution Petition (Const.P)]
Summarized Document Text

H.R.C._14158_G_2016_dt_28_07_2016.pdf [Human Right Case (H.R.C)]
Summarized Document Text

Const.P.35of2013_01.07.2013.pdf [Constitution Petition (Const.P)]
Summarized Document Text

S.M.C.18OF2010.pdf [Suo Moto Case (smc)]

Figure 7. 2 Front end with results

Chapter 8

Evaluation, Findings and Conclusion

8.1 Introduction

In this phase we will be heading towards evaluation of our implemented search engine. Different evaluation measures are already discussed in chapter 4 and will be used in this phase. Different measures will be compared in order to find the best technique for our search engine. At the end of this, conclusion will be provided along with the efficiency of our system and future tasks.

8.2 Evaluation

Our search engine is implemented using vector space model and using this model we used two different weighting schemes. One is classical tf-IDF (term frequency-Inverse Document Frequency) and the other is Okapi BM25, that means two different weighting schemes are used in one search engine in order to find the best weighting scheme for high precision of our system. First system is made robust enough to handle both type of weighting scheme and then a query set is applied on each of them in order to find the best results. For evaluation we used the following measures:

- Recall
- Precision
- Average Precision (AP)
- Mean Average Precision (MAP)
- F-measures.

8.2.1 Formation of Query Set

For evaluation of our system we must have to have a query set with the help of which different (used) weighting scheme can be compared on the basis of results against queries. There are two basic method of query formulation. One is: take the system to the concerned people and ask them to generate some query according to their information need. Note those queries along with their results. Second one is: build an automated system for generation of queries. We choose the second option. We made a system that takes most used words (a word that occurs the most number of times in the collection) in a class (a class consist of many documents) and combine them to form a query. For whole corpus (collection) the vocabulary size is *45492* and among that word types query words will be selected. Fig 8.1 shows the process of formation of query.

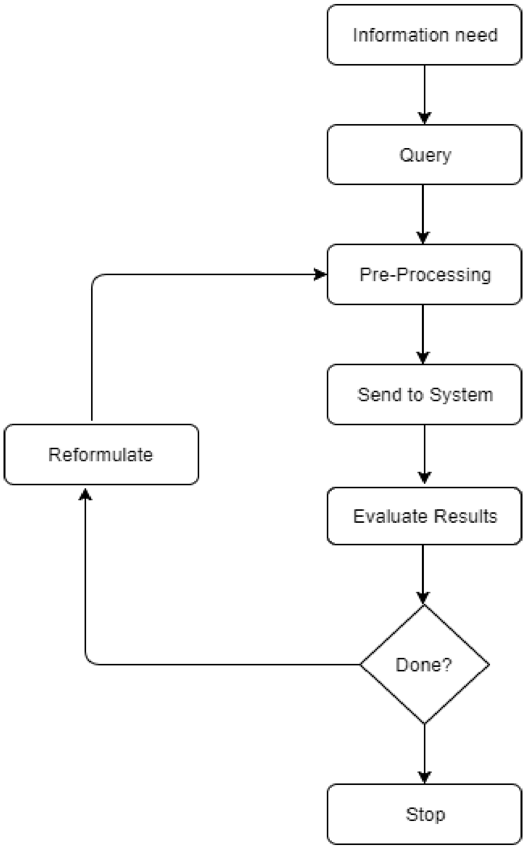


Figure 8. 1 *Query Formation Process*

Before using an automated system for generating queries there is a need to understand what will be the relevancy criteria of a retrieved document. For our system the relevancy criteria will be its class i.e., a query is made up against a class, and if the retrieved document belongs to the same class of which the query is, we say the document is *relevant*. Below is the query set shown in the form of table along with their classes.

Table 8. 1 Query Set

Query No.	Query	Class
q1	court election appellant respondent	Civil Appeal
q2	election court cma order	Civil Miscellaneous Appeal
q3	court petitioner respondent	Civil Petition
q4	constitution court pakistan assembly	Constitution of Pakistan
q5	court pakistan constitution asc	Constitution Petition
q6	criminal appeal court case	Criminal Appeal
q7	cma bar court council	Criminal Miscellaneous Appeal
q8	criminal original petitions	Criminal Original Petition
q9	criminal court petitioner	Criminal Petition
q10	justice court petition review	Criminal Review Petition
q11	Prevention electronic crimes bill	Cyber Crime Bill Pakistan
q12	court order justice dr	Human Right Case
q13	intra court appellant case	Intra Court Appeal
q14	privatization pakistan psmc government	Steel Mills Cases and Order
q15	suo moto sindh order case	Suo Moto Case

This query set will be used for the evaluation of our search engine. One by one every query will be executed on both of the system (one system is with tf-IDF and other is with BM25) and results will be compared in the very next section.

8.2.2 Executing Query Set and Results

One by one query results are shown below. All the results are calculated on the basis of first 10 retrieved documents against a query.

Table 8. 2 Query 1 Results

Ranking (Q1)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	R	R	R	R	R	R	R	IR	R	R
	Recall	0.006	0.013	0.020	0.026	0.033	0.040	0.046	0.046	0.053	0.060
	Precision	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.875	0.89	0.9
	F-measure	0.011	0.025	0.039	0.050	0.063	0.076	0.087	0.060	0.066	0.011
BM25	*Document Relevant (R) /Irrelevant(IR)*	R	IR	R	IR	R	IR	R	R	IR	IR
	Recall	0.006	0.006	0.013	0.013	0.020	0.020	0.026	0.033	0.033	0.033
	Precision	1.0	0.5	0.67	0.5	0.6	0.5	0.57	0.62	0.56	0.5
	F-measure	0.011	0.091	0.025	0.005	0.038	0.038	0.49	0.062	0.062	0.061

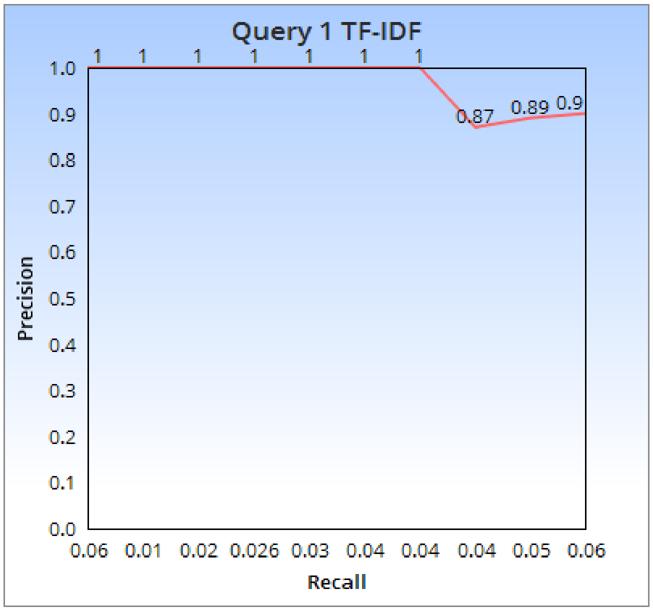


Figure 8. 2** Query 1 Result using **tf-IDF

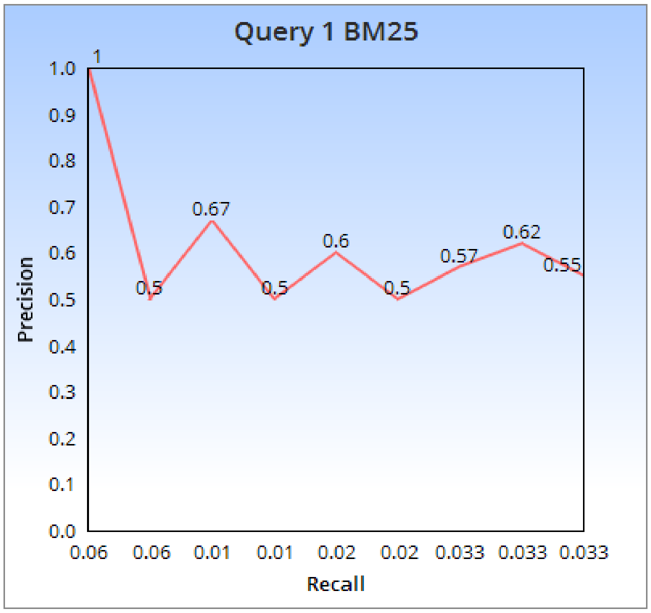


Figure 8. 3** Query 1 Result using **BM25

Table 8. 3 Query 2 Results

Ranking (Q2)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	IR	IR	**R**	IR	IR	**R**	**R**	**R**	**R**	**IR**
	Recall	0.0	0.0	0.020	0.020	00.020	0.041	0.062	0.083	0.104	0.104
	Precision	0.0	0.0	0.33	0.25	0.2	0.3	0.42	0.5	0.56	0.5
	F-measure	0.0	0.0	0.037	0.037	0.036	0.073	0.108	0.142	0.175	0.172
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	IR	IR	IR	IR	IR	**R**
	Recall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.020
	Precision	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	F-measure	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.033

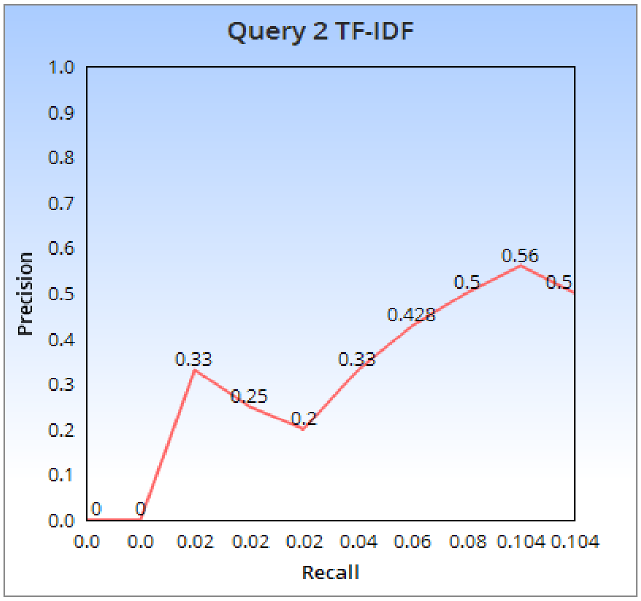


Figure 8. 4 *Query 2 Result using* ***tf-IDF***

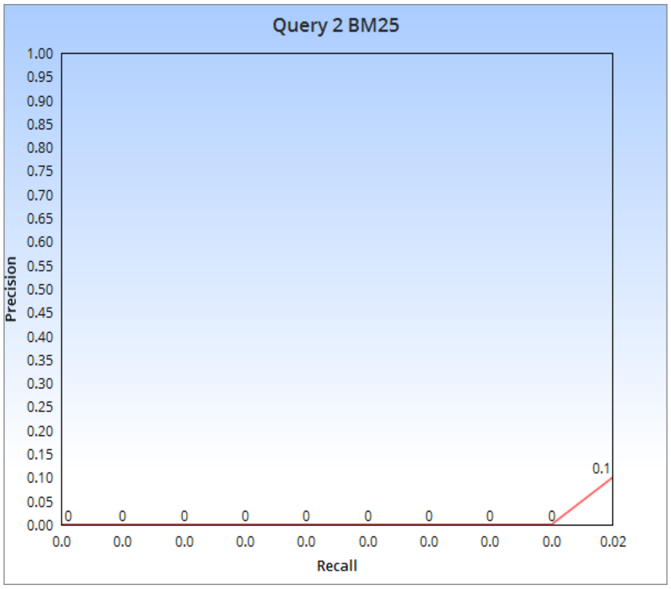
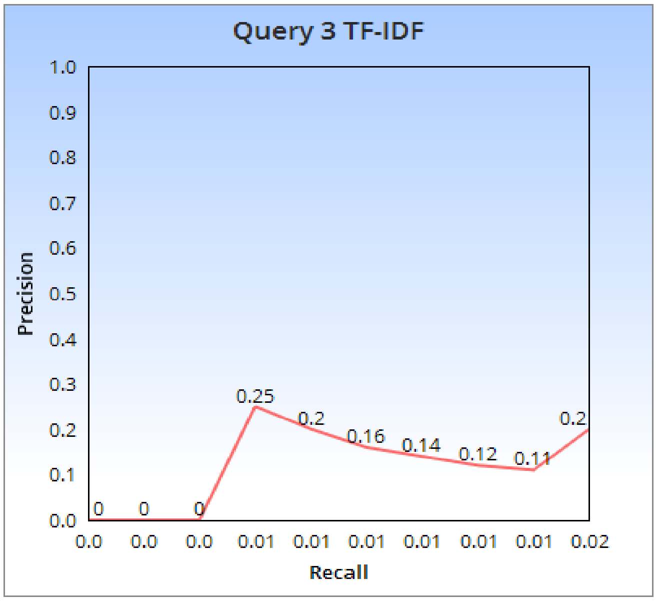


Figure 8. 5 *Query 2 Result using* ***BM25***

Table 8. 4 Query 3 Results

Ranking (Q3)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	IR	IR	IR	**R**	IR	IR	IR	IR	IR	**R**
	Recall	0.0	0.0	0.0	0.013	0.013	0.013	0.013	0.013	0.013	0.027
	Precision	0.0	0.0	0.0	0.25	0.2	0.167	0.142	0.125	0.111	0.2
	F-measure	0.0	0.0	0.0	0.024	0.024	0.024	0.023	0.023	0.023	0.232
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	**R**	IR	IR	**R**	IR	IR	IR	IR
	Recall	0.0	0.0	0.013	0.013	0.013	0.024	0.024	0.024	0.024	0.024
	Precision	0.0	0.0	0.33	0.25	0.2	0.33	0.28	0.25	0.22	0.2
	F-measure	0.0	0.0	0.025	0.024	0.024	0.049	0.049	0.048	0.048	0.047



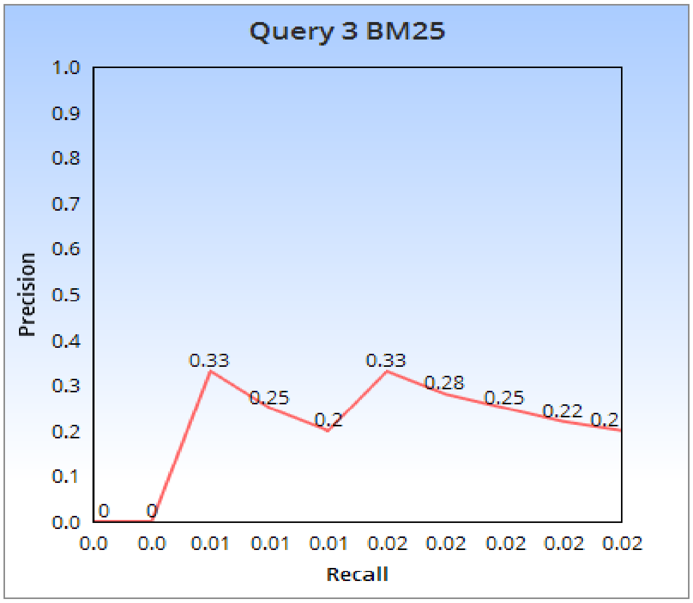


Figure 8. 6 *Query 3 Result using **tf-IDF***

Figure 8. 7 *Query 3 Result using **BM25***

Table 8. 5 Query 4 Results

Ranking (Q4)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Precision	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	F-measure	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Precision	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	F-measure	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

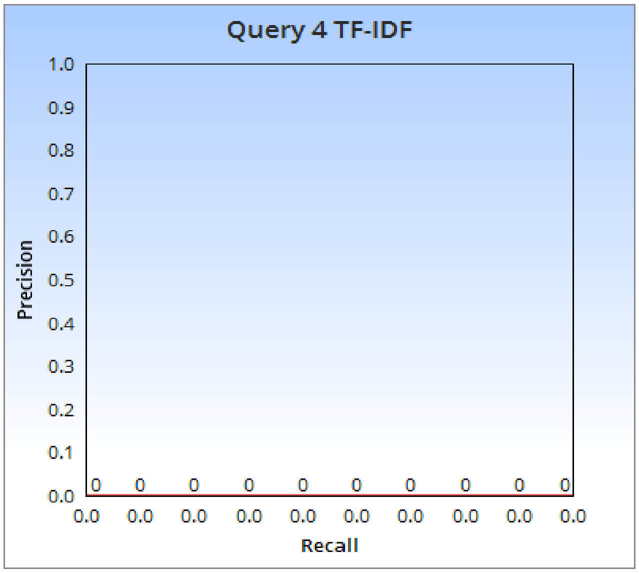


Figure 8. 8 *Query 4 Result using* ***tf-IDF***

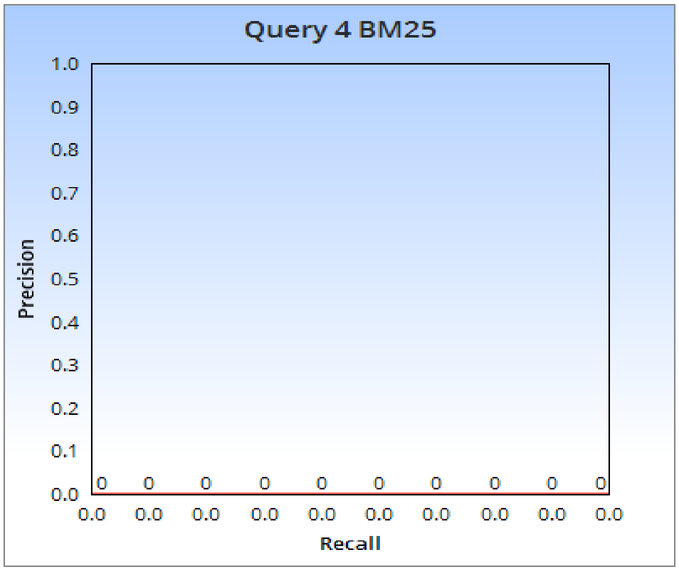


Figure 8. 9 *Query 4 Result using* ***BM25***

Table 8. 6 Query 5 Results

Ranking (Q5)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	**R**	IR	**R**	IR	**R**	**R**	**R**	IR
	Recall	0.015	0.030	0.046	0.046	0.0691	0.061	0.07	0.092	0.107	0.107
	Precision	1.0	1.0	1.0	0.75	0.8	0.67	0.714	0.75	0.78	0.7
	F-measure	0.029	0.058	0.087	0.086	0.113	0.111	0.127	0.163	0.188	0.185
BM25	*Document Relevant (R) /Irrelevant(IR)*	**R**	IR	IR	**R**	IR	**R**	**R**	**R**	**R**	**R**
	Recall	0.015	0.015	0.015	0.030	0.030	0.046	0.061	0.080	0.092	0.107
	Precision	1.0	0.5	0.33	0.5	0.4	0.5	0.57	0.62	0.67	0.7
	F-measure	0.029	0.029	0.28	0.056	0.055	0.084	0.110	0.141	0.161	0.185

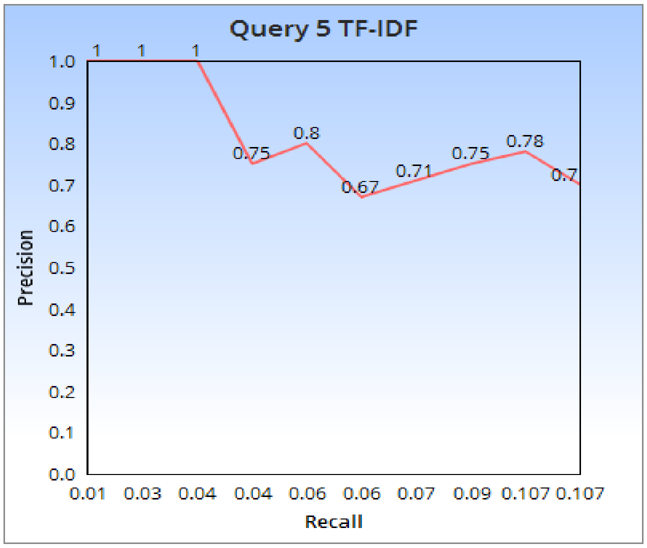


__Figure 8. 10__ Query 5 Result using __tf-IDF__

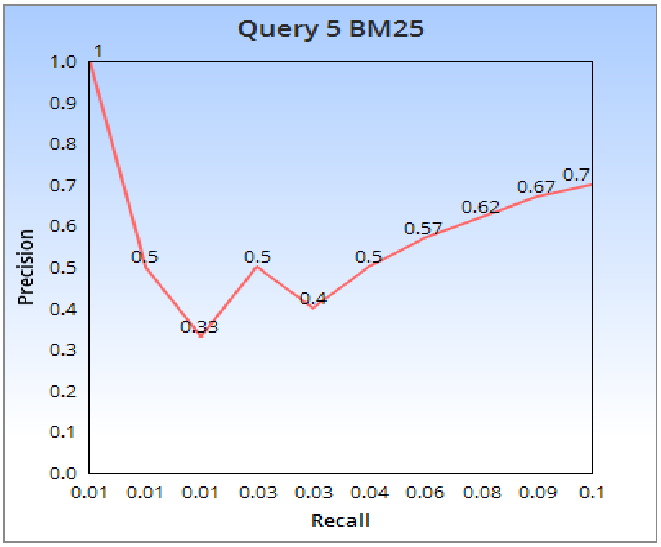


__Figure 8. 11__ Query 5 Result using __BM25__

Table 8. 7 Query 6 Results

Ranking (Q6)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	IR	IR	IR	IR	IR	IR	IR	IR	**R**
	Recall	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.071
	Precision	1.0	0.5	0.33	0.25	0.2	0.167	0.148	0.25	0.111	0.2
	F-measure	0.067	0.065	0.063	0.061	0.059	0.057	0.056	0.054	0.053	0.104
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	**R**	IR	IR	IR	IR	IR	IR	**R**	IR
	Recall	0.0	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035	0.035
	Precision	0.0	0.5	0.333	0.25	0.2	0.167	0.142	0.125	0.222	0.2
	F-measure	0.0	0.065	0.063	0.061	0.059	0.057	0.056	0.054	0.107	0.104

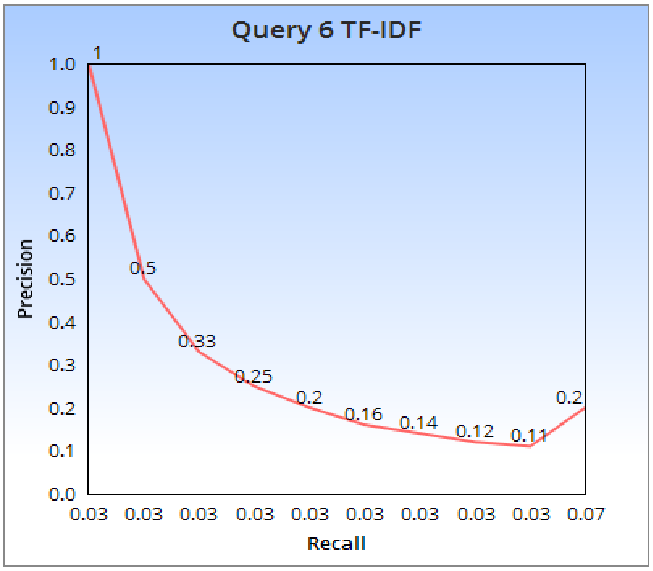


*Figure 8. 12 Query 6 Result using **tf-IDF***

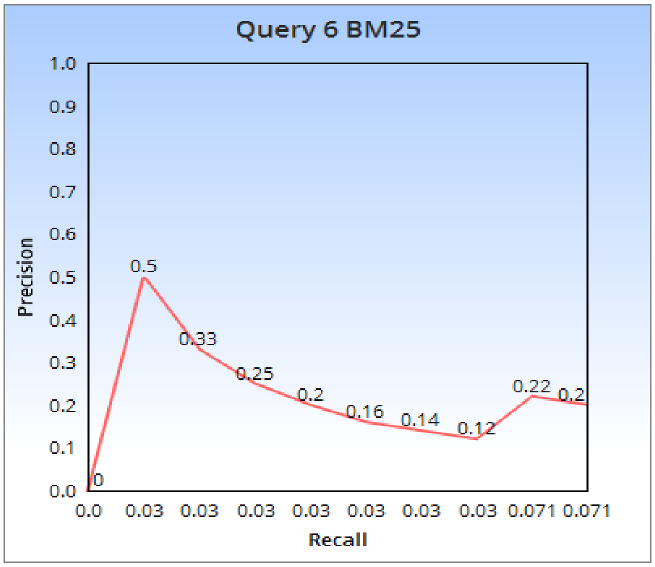


*Figure 8. 13 Query 6 Result using **BM25***

Table 8. 8 Query 7 Results

Ranking (Q7)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	Precision	1.0	1.0	0.67	0.5	0.4	0.3	0.3285	0.285	0.22	0.2
	F-measure	0.333	0.571	0.500	0.444	0.4	0.342	0.332	0.307	0.283	0.266
BM25	*Document Relevant (R) /Irrelevant(IR)*	**R**	**R**	IR	IR	**R**	IR	IR	IR	IR	IR
	Recall	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.6	0.6	0.6
	Precision	1.0	1.0	0.67	0.5	0.6	0.5	0.42	0.37	0.33	0.3
	F-measure	0.333	0.571	0.500	0.444	0.6	0.545	0.499	0.461	0.428	0.4

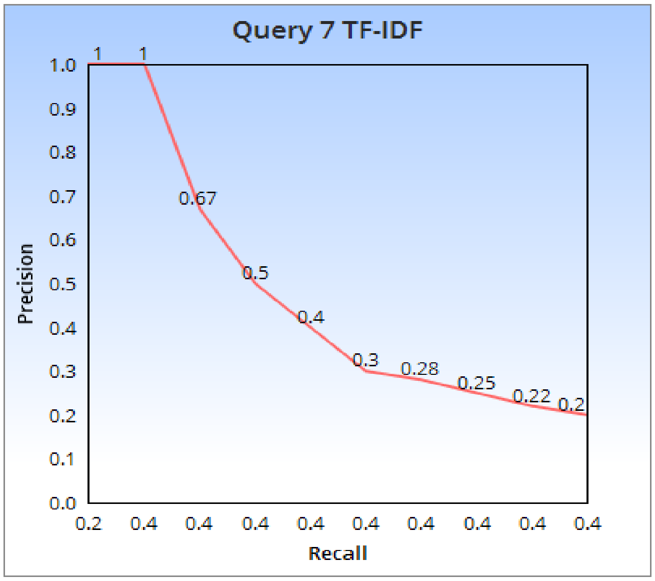


Figure 8. 14 Query 7 Result using ***tf-IDF***

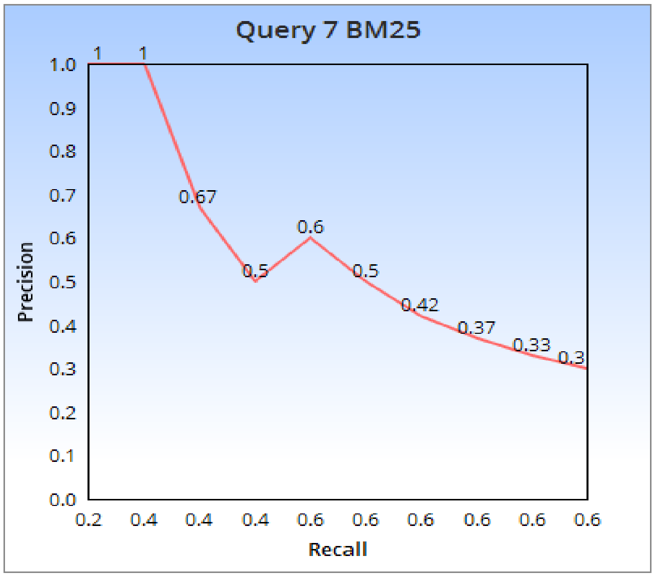
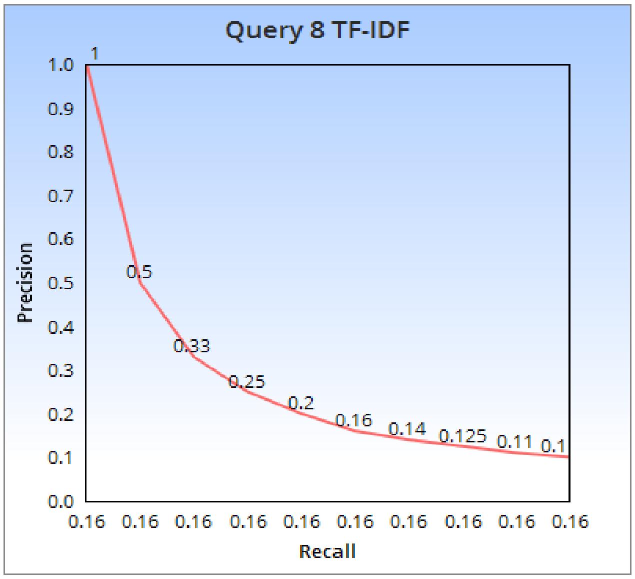


Figure 8. 15 Query 7 Result using ***BM25***

Table 8. 9 Query 8 Results

Ranking (Q8)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	R	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167
	Precision	1.0	0.5	0.33	0.25	0.2	0.167	0.142	0.125	0.11	0.1
	F-measure	0.286	0.250	0.221	0.200	0.182	0.167	0.153	0.142	0.133	0.125
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	**R**	IR	**R**	IR	IR	IR
	Recall	0.0	0.0	0.0	0.0	0.167	0.167	0.333	0.333	0.333	0.333
	Precision	0.0	0.0	0.0	0.0	0.2	0.16	0.28	0.25	0.22	0.2
	F-measure	0.0	0.0	0.0	0.0	0.182	0.167	0.304	0.285	0.266	0.249



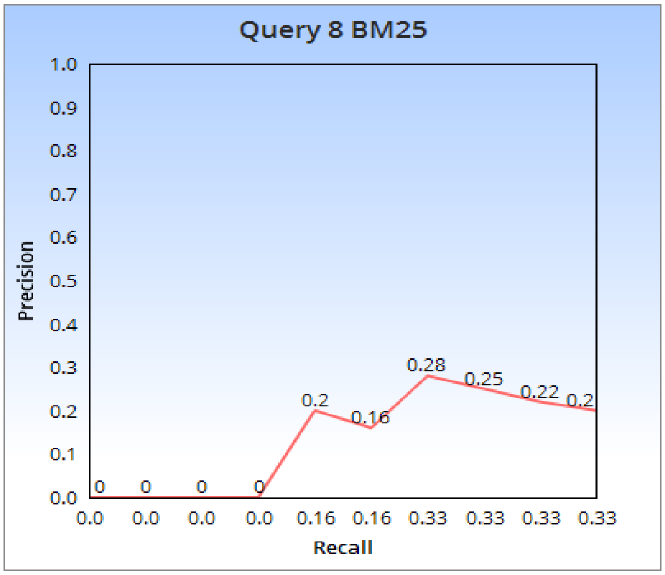
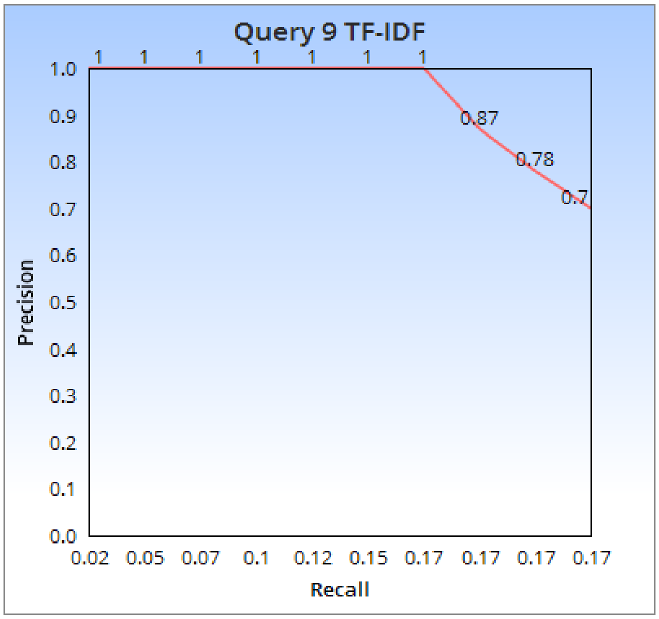


Figure 8. 16 *Query 8 Result using **tf-IDF***

Figure 8. 17 *Query 8 Result using **BM25***

Table 8. 10 Query 9 Results

Ranking (Q9)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	**R**	**R**	**R**	**R**	**R**	IR	IR	IR
	Recall	0.025	0.05	0.075	0.1	0.125	0.15	0.175	0.175	0.175	0.175
	Precision	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.875	0.787	0.8
	F-measure	0.048	0.029	0.139	0.181	0.222	0.260	0.297	0.291	0.285	0.279
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Precision	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	F-measure	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



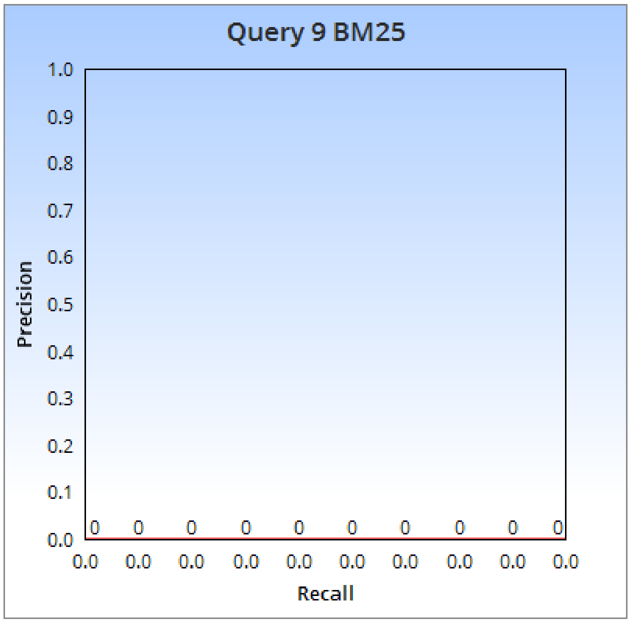
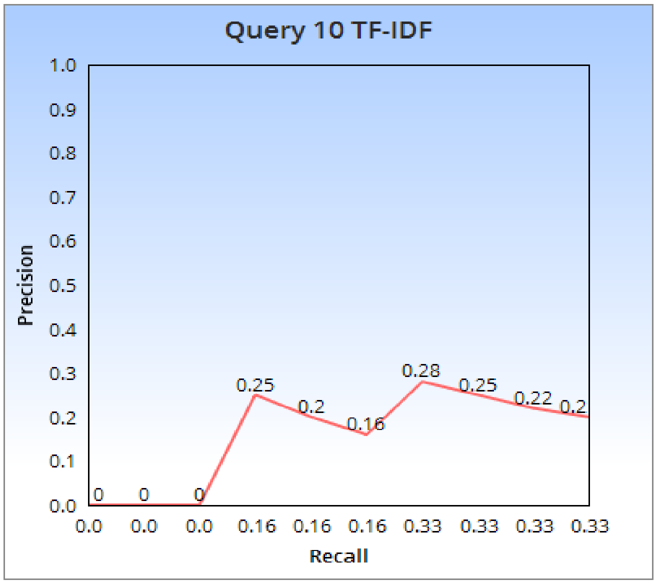


Figure 8. 18 *Query 9 Result using* ***tf-IDF***

Figure 8. 19 *Query 9 Result using* ***BM2***

Table 8. 11 Query 10 Results

Ranking (Q10)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	IR	IR	IR	**R**	IR	IR	**R**	IR	IR	IR
	Recall	0.0	0.0	0.0	0.167	0.167	0.167	0.333	0.333	0.333	0.333
	Precision	0.0	0.0	0.0	0.25	0.2	0.167	0.28	0.25	0.22	0.2
	F-measure	0.0	0.0	0.0	0.200	0.182	0.167	0.304	0.284	0.264	0.249
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	**R**	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167
	Precision	0.0	0.5	0.33	0.25	0.2	0.17	0.14	0.12	0.11	0.1
	F-measure	0.0	0.2510	0.222	0.200	0.182	0.167	0.153	0.142	0.133	0.125



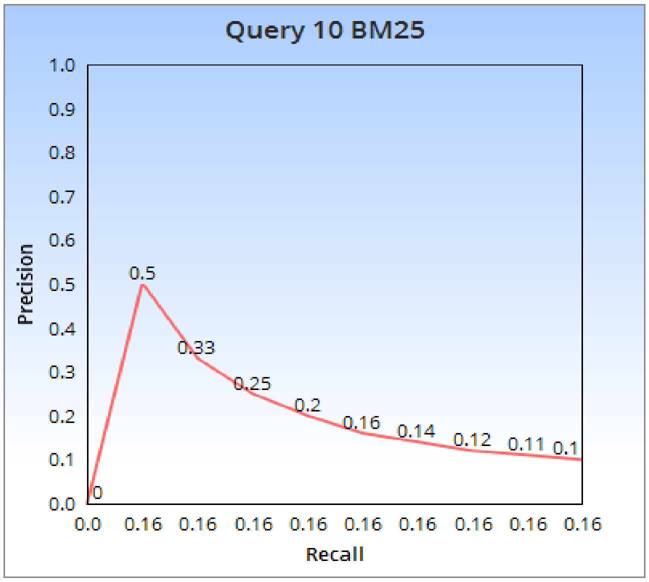


Figure 8. 20 *Query 10 Result using* ***tf-IDF***

Figure 8. 21 *Query 10 Result using* ***BM25***

Table 8. 12 Query 11 Results

Ranking (Q11)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Precision	1.0	0.5	0.3	0.25	0.2	0.16	0.14	0.12	0.11	0.1
	F-measure	1.0	0.666	0.46	0.4	0.33	0.286	0.248	0.222	0.199	0.181
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	**R**	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Precision	0.0	0.0	0.33	0.25	0.2	0.17	0.14	0.12	0.11	0.1
	F-measure	0.0	0.0	0.49	0.4	0.33	0.29	0.24	0.222	0.199	0.181

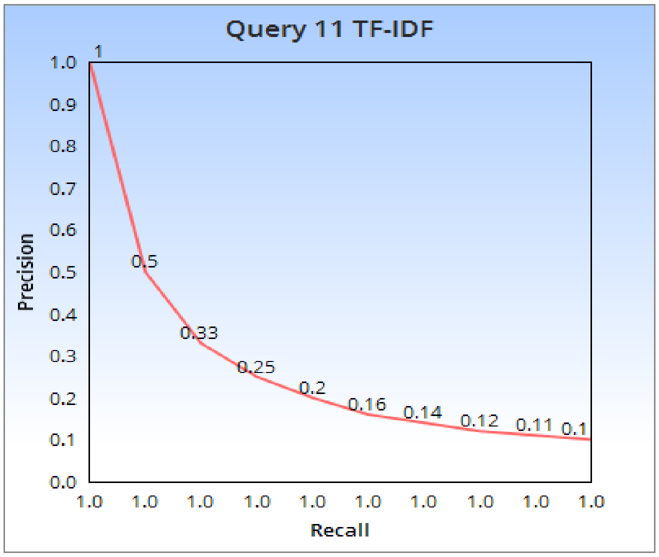


Figure 8. 22 *Query 11 Result using* ***tf-IDF***

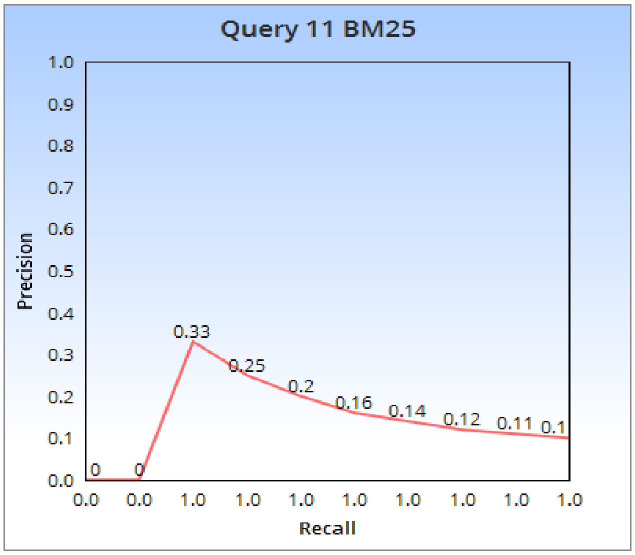


Figure 8. 23 *Query 11 Result using* ***BM25***

Table 8. 13 Query 12 Results

Ranking (Q12)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	**R**	IR	IR	IR	IR	IR	IR	IR
	Recall	0.041	0.083	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
	Precision	1.0	1.0	1.0	0.75	0.6	0.5	0.428	0.375	0.333	0.33
	F-measure	0.078	0.097	0.222	0.214	0.206	0.2	0.193	0.187	0.181	0.176
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Precision	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	F-measure	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

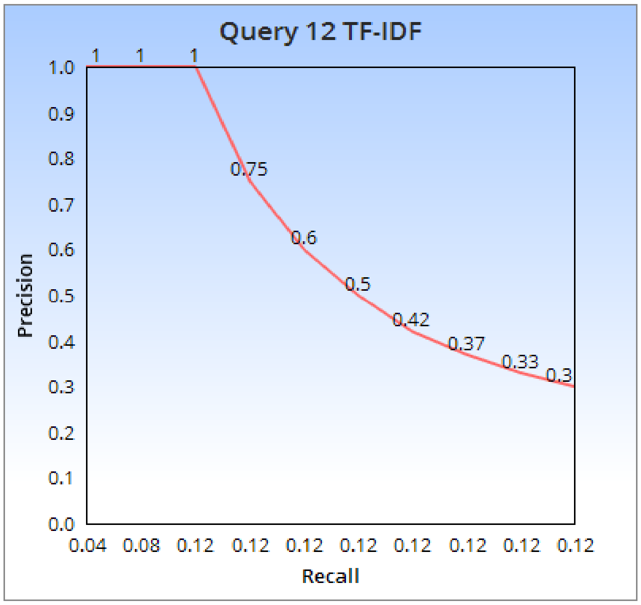


Figure 8. 24 *Query 12 Result using* ***tf-IDF***

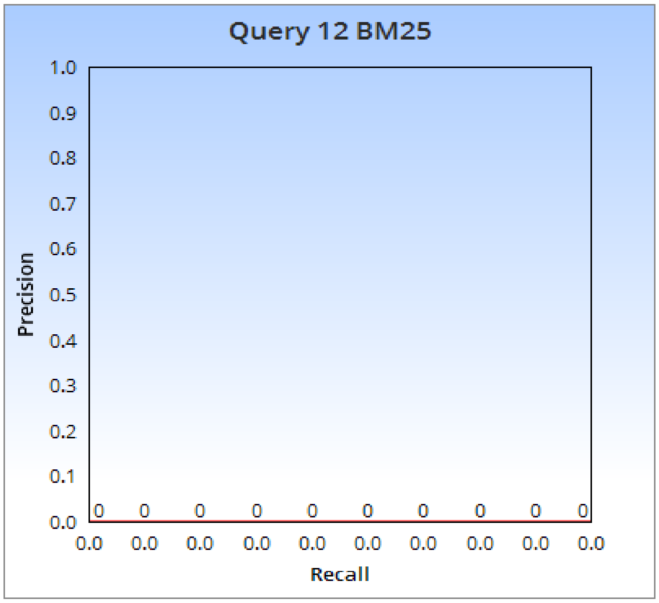


Figure 8. 25 *Query 12 Result using* ***BM25***

Table 8. 14 Query 13 Results

Ranking (Q13)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	IR	**R**	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.0	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333
	Precision	0.0	0.5	0.333	0.25	0.2	0.167	0.142	0.125	0.111	0.1
	F-measure	0.0	0.399	0.331	0.295	0.249	0.222	0.199	0.181	0.166	0153
BM25	*Document Relevant (R) /Irrelevant(IR)*	IR	IR	IR	IR	**R**	IR	IR	IR	**R**	IR
	Recall	0.0	0.0	0.0	0.0	0.333	0.333	0.333	0.333	0.67	0.67
	Precision	0.0	0.0	0.0	0.0	0.2	0.167	0.142	0.125	0.222	0.2
	F-measure	0.0	0.0	0.0	0.0	0.249	0.222	0.199	0.181	0.333	0.308

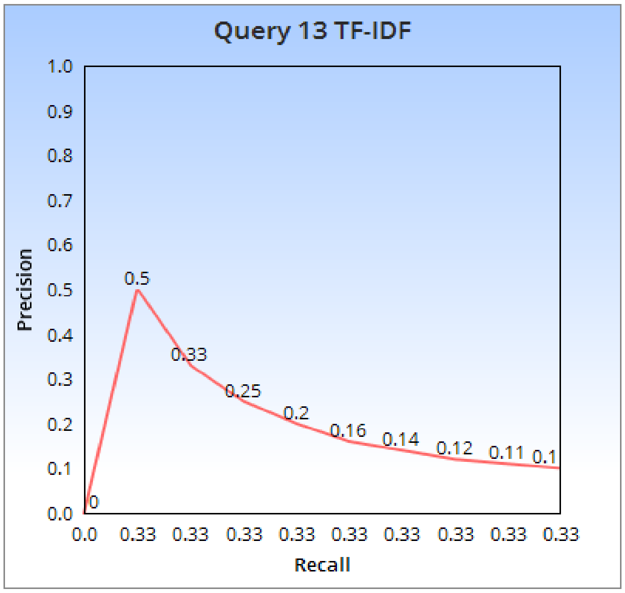


Figure 8. 26 *Query 13 Result using* ***tf-IDF***

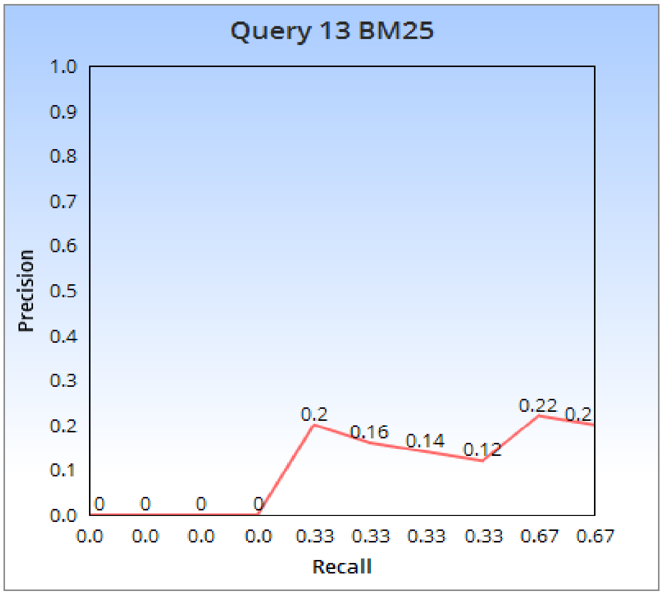


Figure 8. 27 *Query 13 Result using* ***BM25***

Table 8. 15 Query 14 Results

Ranking (Q14)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Precision	1.0	1.0	0.67	0.5	0.4	0.33	0.28	0.25	0.22	0.2
	F-measure	0.666	1.0	0.802	0.666	0.57	0.499	0.437	0.4	0.360	0.333
BM25	*Document Relevant (R) /Irrelevant(IR)*	**R**	**R**	IR	IR	IR	IR	IR	IR	IR	IR
	Recall	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Precision	1.0	1.0	0.67	0.5	0.4	0.33	0.28	0.25	0.22	0.2
	F-measure	0.666	1.0	0.802	0.666	0.57	0.499	0.437	0.4	0.360	0.333

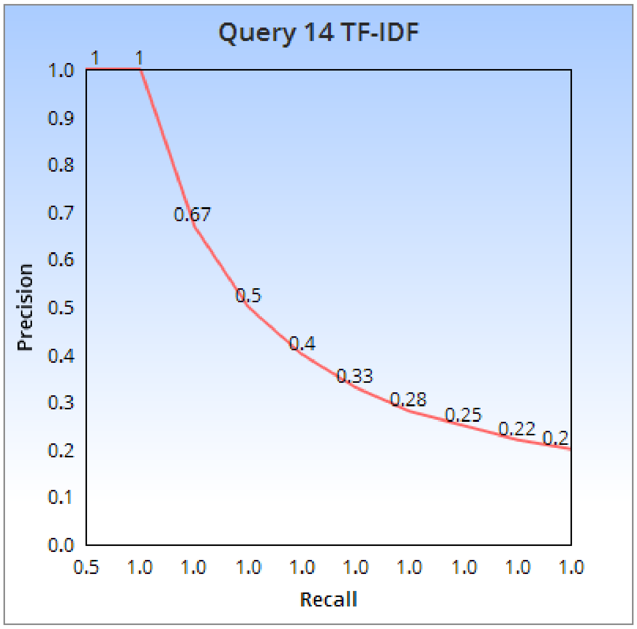


Figure 8. 28 Query 14 Result using ***tf-IDF***

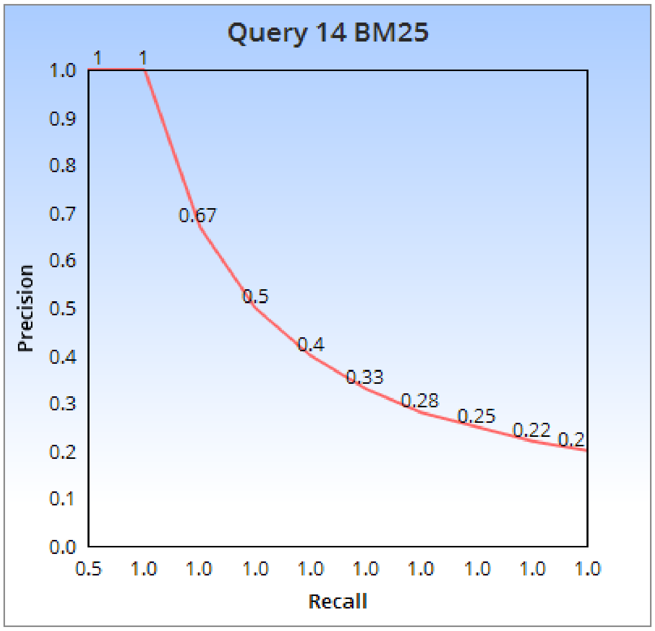


Figure 8. 29 Query 14 Result using ***BM25***

Table 8. 16 Query 15 Results

Ranking (Q15)		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
tf-IDF	*Document: Relevant (R) /Irrelevant(IR)*	**R**	**R**	**R**	**R**	**R**	IR	**R**	**R**	IR	**R**
	Recall	0.034	0.068	0.103	0.137	0.172	0.172	0.206	0.241	0.241	0.275
	Precision	1.0	1.0	1.0	1.0	1.0	0.83	0.85	0.87	0.78	0.8
	F-measure	0.065	0.127	0.186	0.240	0.293	0.285	0.332	0.377	0.368	0.409
BM25	*Document Relevant (R) /Irrelevant(IR)*	**R**	IR	**R**	IR	IR	**R**	IR	**R**	**R**	IR
	Recall	0.034	0.034	0.068	0.068	0.068	0.103	0.103	0.137	0.172	0.172
	Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.428	0.5	0.56	0.5
	F-measure	0.065	0.404	0.123	0.119	0.116	0.170	0.166	0.215	0.263	0.255

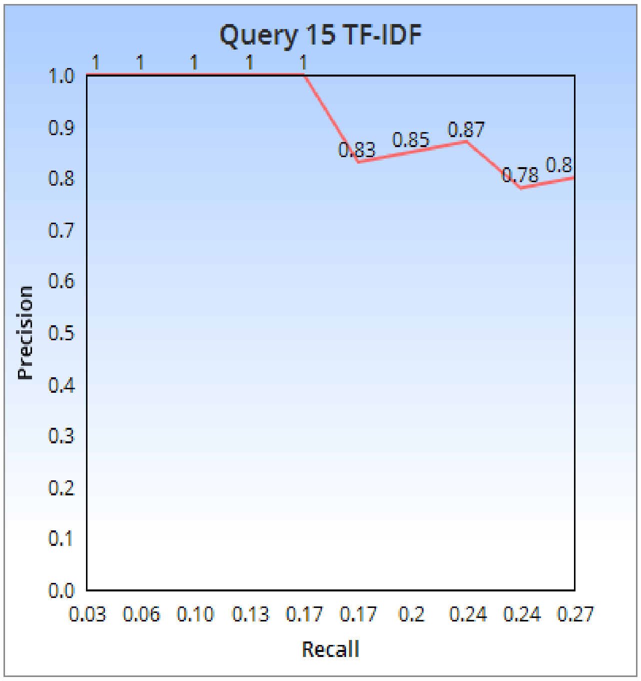


*Figure 8. 30 Query 15 Result using **tf-IDF***

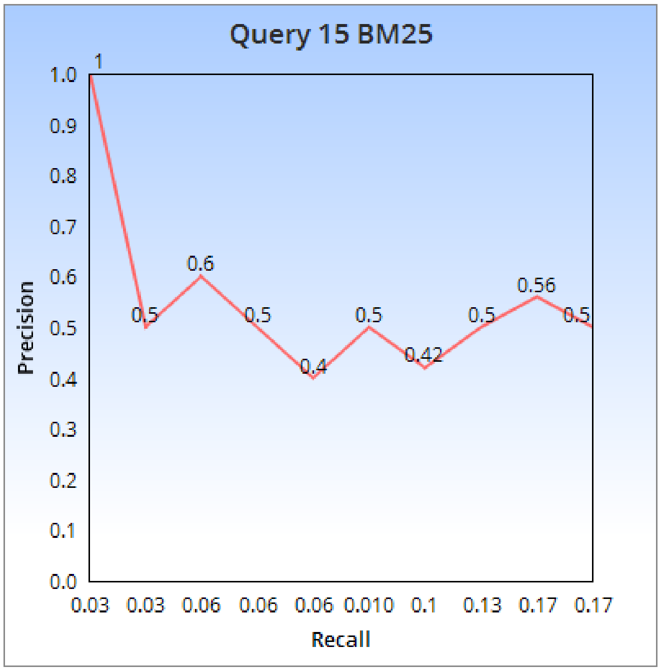


*Figure 8. 31 Query 15 Result using **BM25***

8.3 Findings and Conclusion

Our goal was to implement our product using two different weighting schemes in order to find the best one in terms of relevancy and ranking. We made a query set and applied them on both of the systems (tf-IDF and BM25). On some queries tf-IDF was better than BM25, somewhere BM25 was better than tf-IDF and against some queries the result was same. But on average we concluded that tf-IDF was much better than BM25 for the kind of vocabulary that our search engine is using. We can prove our argument on the basis of average precision (AP) and mean average precision (MAP).

$$\mathrm{AP}_{\mathrm{q(i)}} = \frac{\sum_{i=0}^{n} Presicion\ of\ Relevant\ Documents}{\text{Total No. of Relevant Document in q(i)}}$$

$$\mathrm{MAP} = \frac{\sum_{i=0}^{n} APq(i)}{\text{Total No.of Queries (n)}}$$

Table 8. 17 Average Precision and Mean Average Precision

tf-IDF	**Query**	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
	AP	0.98	0.43	0.23	0.0	0.863	0.6	1.0	1.0	1.0	0.265	1.0	1.0	0.5	1.0	0.941
	MAP	**72.06%**														
BM25	**Query**	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15
	AP	0.69	0.1	0.333	0.0	0.65	0.361	0.867	0.24	0.0	0.5	0.333	0.0	0.211	1.0	0.65
	MAP	**40.00%**														

From the above table we can see that MAP (mean average precision) of the system using tf-IDF is higher than by using BM25 although for some queries BM25 shows better results but overall tf-IDF is better. At the end we conclude that tf-IDF is good for such kind of vocabulary that we are using for our search engine as compared to BM25.

8.4 Future Tasks

In future this system can be implemented using some other weighting scheme like SMART and some other schemes and then can be compared with this system in order to get higher precision and accuracy. Moreover this system uses vector space model, in future a touch of probabilistic model can be included in order to make this system more efficient.

In future SEO (search engine optimization) can be done on this system to improve the efficiency of this system. It is the process of getting traffic from the “free”, “organic”, “editorial” or “natural” search results on search engines. In future a lawyer profile can also be made in order to book a lawyer for some cases. Also legal dictionary can also be integrated with this system if anyone wants to search meaning of a legal term. Moreover, document summarization can also be implemented in order to give a short view of document to the user before opening it.

REFRENCES

1. Thomas Reuters WESTLAW. http://westlawinternational.com (Retrieved on October 2016).
2. John B. West, West publishing Company, “Multiplicity of Reports”. http://www.hyperlaw.com//90-99-docs/1909-multiplicity_of_reports_jbwest.html (Retrieved on October 2016).
3. Westlaw Next, The world’s most Advanced Search Engine. http://info.legalsolutions.thomsonreuters.com/pdf/wln2/L-355700_v2.pdf (Retrieved on October 2016).
4. Thomson Reuters Westlaw, http://legalsolutions.thomsonreuters.com/law-products/westlaw-legal-research/ (Retrieved on October 2016)
5. http://legalsolutions.thomsonreuters.com/law-products/westlaw-legal-research/all-content (Retrieved on November 2016)
6. “What does tf-idf means.”www.tfidf.com (Retrieved on December 2016).
7. C. D. Manning, P. Raghavan and H. Schutze. “Introduction to Information Retrieval, Cambridge University Press”. Cambridge University Press 2008.
8. Synonymous [2016]. Inverted Index for Text Retrieval. (http://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/ditp/ditp_ch4.pdf) (Retrieved on December 2016)
9. R. Baeza – Yates, B. Ribeiro – neto , Modern Information Retrieval. McGraw-Hill, Inc. New York, NY, USA, 1986.
10. H. Wu and R. Luk and K. Wong and K. Kwok. “Interpreting TF-IDF term weights as making relevance decisions”. ACM Transactions on Information Systems, 2008.
11. Marie-Francine Moen, “Innovative Techniques for Legal Text Retrieval”. Artificial Intelligence and Law vol. **9**: pp. 29–57, 2001
12. Raffaella Bernardi, “Term Frequency and Inverted Document Frequency” http://disi.unitn.it/~bernardi/Courses/DL/Slides_11_12/measures.pdf (visited on Nov 2016).
13. G. Salton, A. Wong, and C. S. Yang, “A Vector Space Model for Automatic Indexing”. Communications of the ACM, vol. 18, nr. 11, pp. 613-620, 1975.
14. Amit Singhal, “Modern Information Retrieval: A Brief Overview”. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 24, No. 4, pp. 35-42, 2001.
15. G. Salton and M. J. McGill. “Introduction to modern information retrieval”. McGraw-Hill, Inc. New York, NY, USA, 1983
16. Ben He & I. Ounis, “Term Frequency Normalization Tuning for BM25 and DFR Models”. D.E. Losada and J.M. Fern´andez-Luna (Eds.): ECIR 2005, LNCS 3408, pp. 200–214, 2005.
17. A. Singhal, C. Buckley, and M. Mitra. “Pivoted document length normalization”. Proceedings: SIGIR ’96 Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval’, pg 21-29, Zurich, Switzerland, 1996.
18. Leonard Richardson, “Beautiful Soup Documentation”. Release 4.4.0, January 2017.

References

19. Frances Zlotnick, Stanford University "Web Scraping with Beautiful Soup" (http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html) visited on January 2017).
20. Farlex, The free Legal Dictionary, http://legal-dictionary.thefreedictionary.com (Retrieved on January 2017).
21. R. Herbrich and T. Graepel, "Handbook of Natural Language Processing (2nd edition)". CRC Press USA, 2010.