Text-Based Information Retrieval Using Relevance Feedback

Sharenya Krishnan

ROYAL INSTITUTE
OF TECHNOLOGY

Master of Science Thesis

Kista, Sweden 2011

This thesis corresponds to 20 full-time working weeks.

ROYAL INSTITUTE
OF TECHNOLOGY

Text Based Information Retrieval using Relevance Feedback

Author: Sharenya Krishnan

Email: sharenya@kth.se

School of Information and Communication Technology (ICT)

Department of Software and Computer Systems (SCS)

The Royal Institute of Technology (KTH)

Kista, 2011

Supervisor: Professor Magnus Boman

Examiner: Professor Magnus Boman

Abstract

Europeana, a freely accessible digital library with an idea to make Europe's cultural and scientific heritage available to the public was founded by the European Commission in 2008. The goal was to deliver a semantically enriched digital content with multilingual access to it. Even though they managed to increase the content of data they slowly faced the problem of retrieving information in an unstructured form. So to complement the Europeana portal services, ASSETS (Advanced Search Service and Enhanced Technological Solutions) was introduced with services that sought to improve the usability and accessibility of Europeana. My contribution is to study different text-based information retrieval models, their relevance feedback techniques and to implement one simple model. The thesis explains a detailed overview of the information retrieval process along with the implementation of the chosen strategy for relevance feedback that generates automatic query expansion. Finally, the thesis concludes with the analysis made using relevance feedback, discussion on the model implemented and then an assessment on future use of this model both as a continuation of my work and using this model in ASSETS.

Key Words: Information Retrieval, Relevance Feedback, Query Expansion, Rocchio classification, Probabilistic model, Lucene, Similarity scoring function, Kullback-Leibler Divergence (KLD).

Acknowledgements

This work was carried out at Swedish Institute of Computer Science (SICS). First of all, I would like to thank Magnus Boman, my supervisor for giving me the opportunity to work at SICS with this interesting topic and also for explaining me the thesis in the earlier stages and for providing support throughout my thesis. I would also like to thank Oscar Täckström for guiding me in the technical part of my thesis and being helpful in providing the necessary resources during my thesis.

Special thanks to my room colleagues Julia Pettersson and Ali Merikh, who contributed to a pleasant working environment and motivation. I would like to thank Anni Järvelin for organizing “after works” at SICS, which helped me know everyone in the department.

Finally, I would like to thank my family and my friends for giving me the moral support that helped me to carry out the thesis with the same zeal and enthusiasm.

Tack så mycket!

Table of Contents

1. Introduction

I will focus on implementing an information retrieval (IR) model for improving text- based search. The purpose of this section is to give a view on the background of the approach used to improve the text-based IR. The problems faced will be described followed by the solutions to these problems and a brief explanation on the methods that will be implemented.

1.1. Background

Europeana [1] is a freely accessible Digital Library with an idea to make Europe's cultural and scientific heritage available to the public. It was founded by the European Commission on the 20th of November 2008. The goal was to deliver a semantically enriched digital content with multilingual access to it [1] [3]. Networks of partners make up Europeana. These networks of partners are the content providers and they include Europe's libraries, museums, national audiovisual archives, etc. The content providers deliver information via the Europeana portal which can be made available to users for several purposes like work, study, tourism, education, etc. It helps people explore the digital resources and enjoy the richness of the diverse cultural heritage. Europeana succeeded in increasing the digital content items to 15 million with the support of the content providers. These digital content items were made available in different formats such as text, images, video and audio. They also made public books, newspapers, film photographs, drawings and many more media accessible through this digital library portal [1]. Even though they managed to increase the data content, the biggest constraint was retrieving information in a structured form. So to complement the Europeana portal services, an Advanced Search Service and Enhanced Technological Solutions (ASSETS) for the European Digital Library were introduced. ASSETS is one of Europeana's support projects and is a best-practice network co-founded by the CIP (Competitive and Innovation framework Programme) PSP (ICT Policy Support Programme) program [1] [2]. The ASSETS consortium includes 24 partners from ten European countries and one partner from Japan. The ASSETS project aims to increase the usability and the accessibility of Europeana with the provided set of value-added services

that focus on providing generated enriched metadata where the user exchanges information with the digital library offering advantages to both (a) users for getting the information relevant to their need and, (b) the library for improving the description of the object in line with the user preferences [1]. Through these value-added services the users are allowed to search, navigate and view content, thereby enriching the user experience and offering customized search results.

1.2. Problem Statement

The problem of retrieving information relevant to the search is ubiquitous. This is due to the increase in the amount of digital information available and the great demand for text search, as a result of the World Wide Web. People are surrounded with large quantities of information but are unable to use that information effectively because of its abundance. This makes the relevance low and the search functionality limited as the information is retrieved automatically based on a basic keyword match.

The Solutions to this problem will be to first enhance the search functionality by using a specific API (Lucene) that can support full-text search and then implement a ranking function to retrieve information based on the user feedback. So that the information will not be retrieved based on a basic keyword match but rather based on the term weight.

1.3. Objective

The objective is to implement a ranking function based on relevance feedback to improve the effectiveness of the information that will be retrieved. The purpose of using this method is to make the relevance of the search information high and improve the search functionality.

1.4. Methodology

SICS, as one among the 24 partners in the ASSETS project, initially was assigned to implement relevance feedback in IR and see how this can be achieved with text, images and

3D images [1] [2]. There are several IR models and methods available that helps in improving the retrieval process. My objective is to study different text retrieval models, their RF techniques and to apply one model. To implement and test all the available IR models is out of the scope of this thesis, so only one model with RF technique will be examined. The choice of IR model suggested is a probabilistic model and the suggested relevance feedback model is based on the Rocchio framework. Though the notion is to implement a ranking function, to make the search efficient in IR, it is necessary to index all the available information from the database and make the search process easier to retrieve promptly. Lucene [8] will be used to implement this indexing and the search process. But the document content used for the work is collected from the internet and copied as .txt files because my code is not implemented in the ASSETS workspace which had a repository connected to it. This is one reason for using a simple example in explaining my models rather using the example used in ASSETS. A ranking function will be implemented to rank the query terms from the relevant documents selected by the user. So, from the user feedback a new query will be formed. The ranking function that will be used is an information-theoretic approach from the concept of KLD (Kullback-Leibler Divergence) [14] [15]. The Rocchio ranking function basically calculates the term weight for the given user query and reformulates the original query into a new query. This approach is simple and computationally efficient, but it has the disadvantage that each term weight reflects more the usefulness of that term with respect to the entire collection rather than its importance with respect to the user query. So assessing the appropriateness of a term is based on distribution analysis. That is to compare the occurrence in relevant documents with the occurrence in all documents, for each given query. In other words, one may assume that the difference between the distribution of terms in a set of relevant documents and the distribution of the same term in the overall documents collection is an index of semantic difference.

Now the question arises on how to compare the difference or the similarity between two distributions of documents. In our case it is the probability distribution of terms in relevant documents and probability distribution of terms in the whole collection. This can be done by adding relative entropy between the two probability distributions.[1] It is measured by KLD and

[1] Relative entropy is a measure of distance between two probabilistic distributions.

it will give the frequency of the terms which mostly contribute to make the probability of relevant documents divergent with respect to the probability of documents in the whole collection. In fact there are many other ranking functions that calculate term weight apart from Rocchio and KLD. They include Robertson Selection value (RSV), CHI-squared (CHI2), Doszkocs' variant of CHI-squared (CHI1) [15]. The reason to apply KLD ranking in the model is because all these ranking functions are confined to an IR scenario and do not cover automatic query expansion [15].

1.4.1. Tools

The ASSETS workspace is set up already and is supported by SVN (Subversion). With this in the background, we will be using the following tools for our model.

1. Eclipse IDE (Integrated Development Environment), software development environment comprising an integrated development environment used for developing applications in Java. The entire work will be built on Eclipse IDE.
2. Lucene, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. It is used in our model to enable the available resources to be inserted into the application and manage them.
3. Apache Tomcat Server, an open source servlet container. It implements Java servlet and Java server page and provides a pure Java HTTP webserver environment to run the Java code.
4. MYSQL, an RDBMS that runs as a server to provide multi-user access to a number of databases. It is used for Web applications to manage the queries.

1.4.2. Company Description

The Swedish Institute of Computer Science (SICS) started in 1985 as a non-profit research organization and now has approximately 110 researchers including 45 PhDs. SICS works in close collaboration with the industry with national and international research community. Their mission is to contribute to the competitive strength of industry by conducting advanced research in strategic areas of computer science. Their research focus includes distributed and networked interactive real-time system including IR.

1.5. Thesis Overview

Chapter 2 covers the literature study that a reader needs to understand in order to fully comprehend the models that will be implemented. Chapter 3 presents two models that will be implemented to make the retrieval process efficient. The first model will explain the process on how the indexing and searching process is done with a basic similarity scoring function used followed by another model explaining the retrieval process using the term-ranking function. Chapter 4 will explain the results obtained from the ranking function that was used in the models. It will be tested with the available database that is connected to the server to see if the results are relevant. Chapter 5 will have discussion on the models followed by future work and conclusion.

2. Literature Review

2.1. Overview

In this chapter, the technical background needed to follow the ideas and concepts of the thesis will be covered. The rest of the sub-sections will explain the fundamentals of IR, their assumptions, query-based search, the Rocchio algorithm explaining the RF concept and the different IR models that can be used for text retrieval.

2.2. Extended Background

There is a significant amount of cultural heritage material now available through online digital library portals. One such portal is Europeana started which aims at delivering a semantically enriched digital content with multilingual access to it [1]. But it faced problems in searching and retrieving relevant information due to excessive data. For this reason, ASSETS focussed on developing services on searching, browsing and interfaces in order to improve the usability of Europeana [2]. It came up with large-scale services for (1)searching objects based on metadata and on content similarity, (2)browsing objects for rapid navigation through semantic cross-links, (3)interfaces especially designed for interacting with multimedia objects, planning long-term access to digital information, (4)ingestion on metadata requiring normalization, cleaning, knowledge extraction and managing to a common structure [2]. Here, the metadata describes the data about data, in other words it describes the content and the context of data files. A diagrammatic view on the ASSETS services is shown in Figure 1.

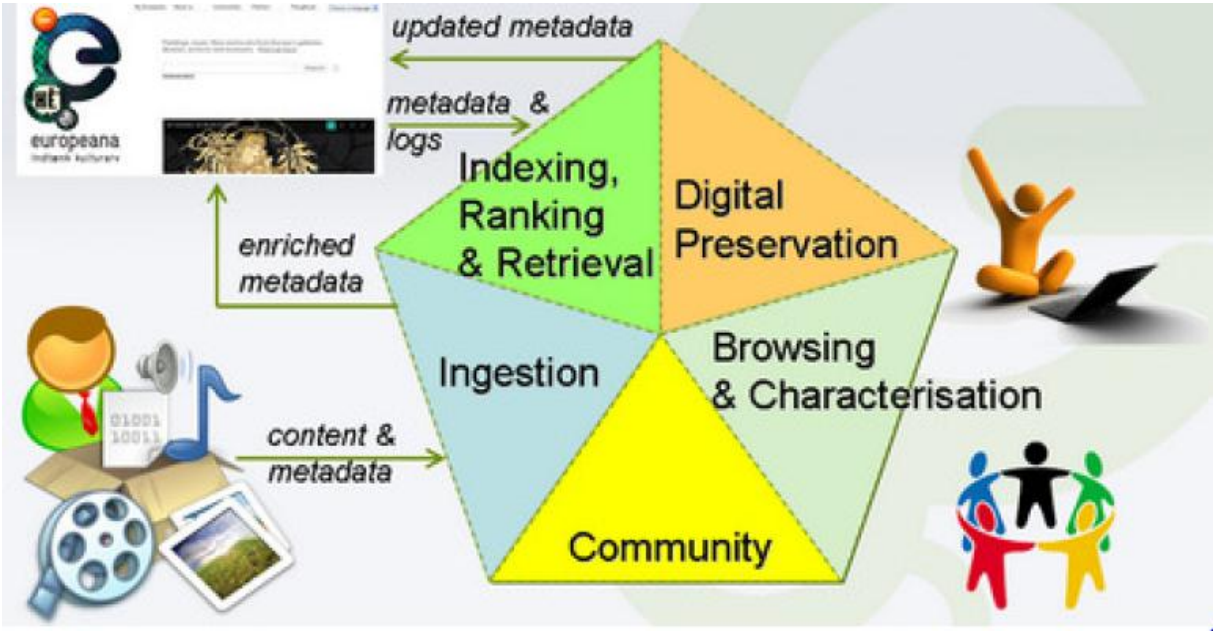

Figure 1. An Overview of ASSETS services. Source: [2], p.2.

Out of these five services, SICS plays a part in implementing the browsing service which involves to [2]:

1. Establish semantic links between pieces of information presented in different media through the annotation propagation service.
2. Appropriately train the annotation propagation service through the training service.
3. Allow to manually correct and complete the ingested metadata and provide training examples to the training service through the manual annotation and annotation correction service.
4. Select the most informative examples for manual annotation through the content selection service.
5. Exploit user feedback information in order to improve the quality of search results through the RF service.
6. Perform query log analysis through the query log analysis service.

As mentioned in 1.4, SICS's role is to implement points 5 and 6. The details on RF will be explained later in this section and by query log analysis we mean the resources to explore the search behaviour of users and thereby improve the entire search experience. As our work is to deal with IR using RF, we will look at some of the most common methods of IR by

following a single query, a query for “information on circus elephant”. Note that in places where examples are referred and explained, the information retrieved is expressed as ‘document’ and document set as ‘corpus’. So, for the given query let us assume we got the following results.

1. Elephant are wild animals.
2. Elephant trained in circuses.
3. Information on savannah elephants.

With this simple corpus, it seems reasonable that our search for that query should return document 2 as the most relevant document as it is the only document that contains any content about the given query. To understand how we conduct a search that ranks these documents and retrieve them according to the user search, I will discuss some basic IR models. But first, I will talk about the basic IR assumption about the corpus that we apply to all of the models that we will discuss. Then I will describe these models, showing how they perform this query and finally, I will discuss the model implemented in this thesis.

2.3. Basic IR Assumptions

Any IR model assumes that there are some features that belong to each document in a corpus, and that these features are sufficient to determine if a document is relevant or irrelevant. Since we are dealing with text, each term can be defined as a word, a semantic meaning, bag of sentences, pair of words, a phrase, or even an entire sentence.

In our examples and in the testing we do later, we assume that each term in the document is considered as a word, but it is important to remember that this is just an additional assumption. We make this assumption because we want our term to appear often enough throughout the corpus to tell us something meaningful about their distribution. During the retrieval, we will be making generalizations about documents from these terms. Now that we know the basic IR assumptions let us look at from where these assumptions are made.

2.4. Query-Based Search

Most of the time the information needed is lost as the user often tries to express complex information as a simple query and the system tries to interpret the given query by returning list of documents related to the query. This is because the queries are most often in free text. The suggested information that is retrieved by the system may vary depending upon the query logs which records the Web user's behaviour when they search for information via a search engine. There are two types of analysis, offline analysis of query logs and online analysis of query logs. Query suggestion and query expansion are performed off-line and on the other hand RF is performed completely online [3]. These two analyses of query logs will be explained below.

2.4.1. Query Expansion and Query Suggestion

Query suggestion is done when the system suggests a related query to the user in order to get related information [4]. It helps to express the information needed by the user. For example, if the user wants to know the abbreviation of MIR related to her/his subject, he/she gets a list of related answers with respect to the query from which the user can get the answer related to her/his needs. On the other hand the query expansion expands the query given by the user. It helps in reformulating the user query in order to increase the quality of the user search results [4]. These two are more useful for fixing spelling errors and for disambiguation of queries with multiple distinct interpretations. They are performed without any effort at the user's end.

2.4.2. Relevance Feedback

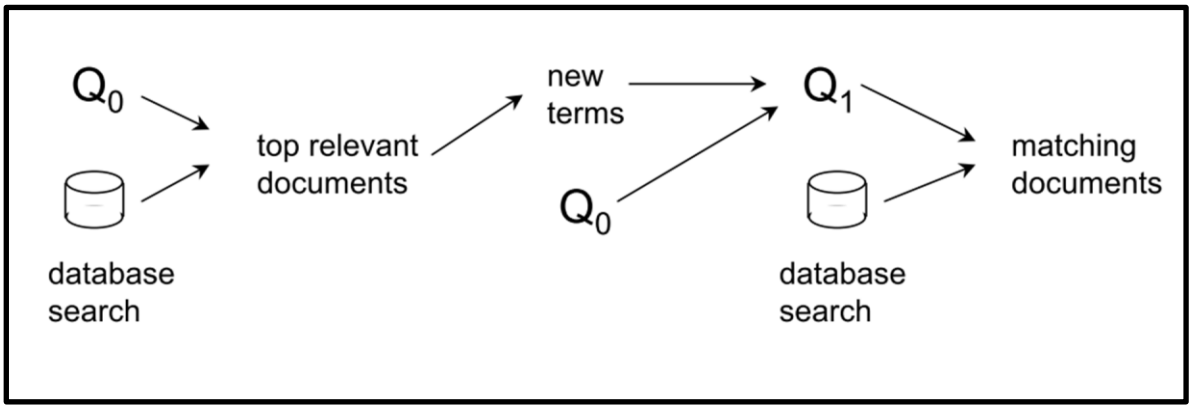

Figure 2. The process of matching documents using Relevance Feedback. Source: [7], p.3.

From the diagram above, a typical user interaction scenario using relevance feedback (RF) can be seen. It involves the following steps and this process can go through one or more iteration of this sort [4],

1. The user issues a query that is short and simple.
2. The system returns an initial set of retrieval results.
3. The user marks some returned documents as relevant or non-relevant.
4. The system computes a better representation of the information needed based on the user feedback.
5. The system displays a better representation of the information needed based on the user feedback.
6. The system displays a revised set of retrieval results.

It acts like a user guide for the system to understand the user query better than before and retrieve information with respect to the given feedback. An example to understand the process of relevance feedback is given in Figure 3.

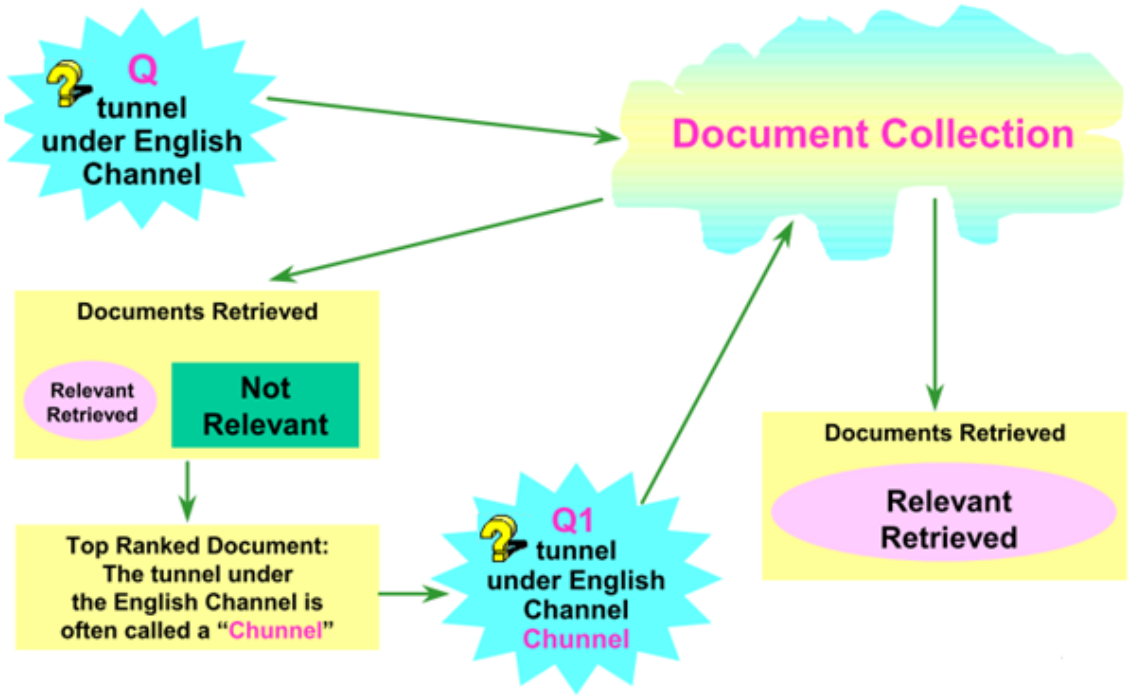

Figure 3. An example explaining the process of Relevance Feedback. Source: [7], p.4.

The ground truth of relevance is that the information provided by the system is known to be either relevant or non-relevant to a particular query. This assumption is made from the evaluation done by two basic measures, precision and recall. These two measures evaluate the performance of the IR process in a system. It depends on the outcome of the query and its relation to all relevant and non-relevant information. During the IR process, precision and recall are defined in terms of a set of retrieved information and set of relevant information. The retrieved information is the list of information retrieved by the Web search engine and the relevant information is the list of information on the internet that is relevant to certain topic.

Precision is the fraction of retrieved documents that are relevant to the search [4].

Recall is the fraction of documents that are relevant to the query that are successfully retrieved [4].

They can be better understood with a set diagram shown below in Figure.4;

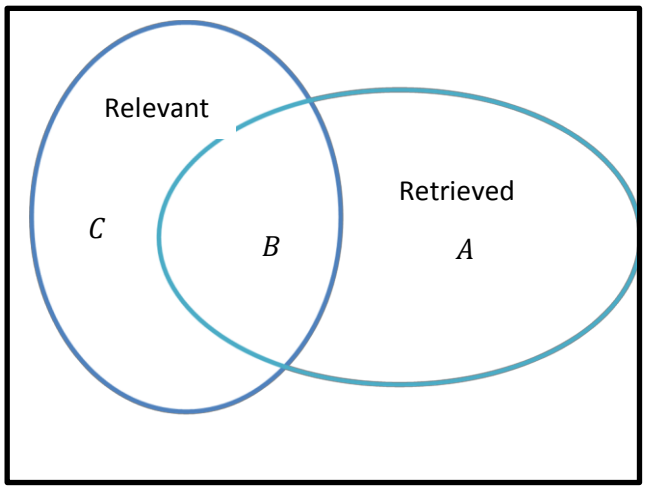

Figure 4. Interaction between retrieved and relevant set of documents.

As we can see, A represents retrieved documents, C represents relevant documents and B represents the intersection of the two set. We can then represent $(A \cup B)$ as the set of retrieved documents, while $(B \cup C)$ is the set of relevant documents.

From the given diagram we can represent,

$$Recall = \frac{|B|}{|B \cup C|}$$

$$Precision = \frac{|B|}{|A \cup B|}$$

The feedback given is either positive or negative. Positive feedback will lead to generate more relevant documents and expand the query with the retrieved information and negative feedback is often the key to good results. But too much negative feedback might be a problem as it might deviate from the given query. In order to avoid this problem, a separate weight must be given to positive and negative information to distribute them accordingly. For this purpose an algorithm was implemented.

2.4.3. Rocchio Algorithm

The Rocchio algorithm is based on the assumption that most users have a general idea of what should be denoted as relevant and non-relevant. The user's query is revisited to include an arbitrary percentage of relevant and non-relevant documents as a means to increase the search engine's recall and possibly the precision as well [4] [7]. It reformulates the query by going into the direction of the positive feedback and away from the negative feedback. This method was developed using Vector Space Model with an idea to maximise the similarity with relevant documents while minimizing the similarity with non-relevant documents as shown below [4]. The Vector Space Model will be explained in section 2.5.

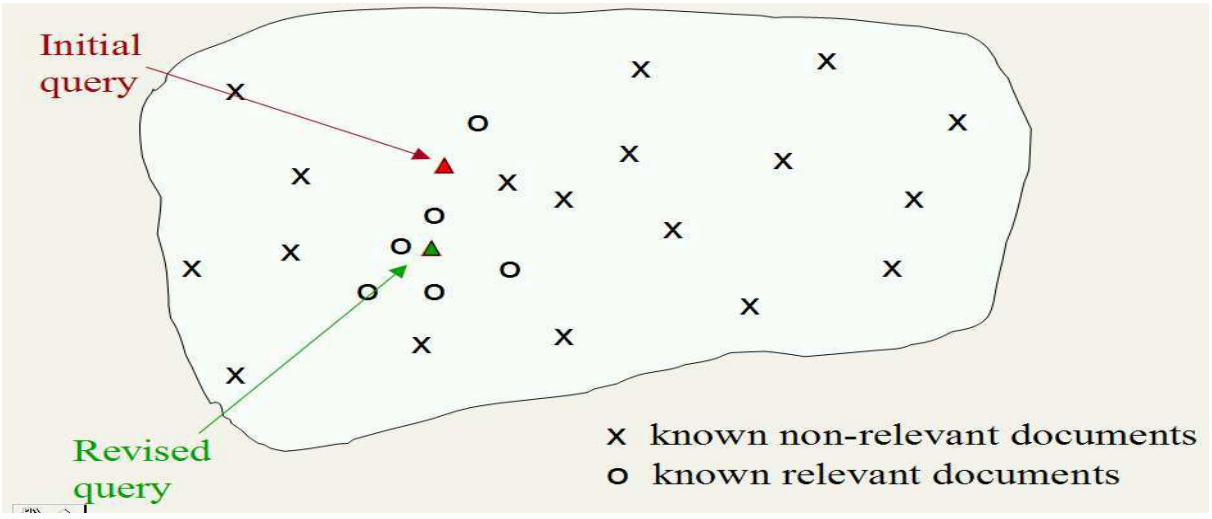

Figure 5. Rocchio Classification. Source: [4], p.182, Figure.9.4.

The formula for the Rocchio's classification can be expressed as [4],

$$\boldsymbol{q'} = \boldsymbol{\alpha} * \boldsymbol{q_o} + \boldsymbol{\beta} * \frac{\mathbf{1}}{|\boldsymbol{D_r}|} \sum_{\boldsymbol{d_j} \in \boldsymbol{D_r}} \boldsymbol{d_j} - \boldsymbol{\gamma} * \frac{\mathbf{1}}{|\boldsymbol{D_{nr}}|} \sum_{\boldsymbol{d_j} \in \boldsymbol{D_{nr}}} \boldsymbol{d_j} \qquad \textbf{Equation 1}$$

where,

q_0 - Original query

q' - Modified new query.

D_r - Set of positive information

D_{nr} - Set of negative information

α, β, γ - Constants (Rocchio weights)

Now that we know how the query is formed and how the retrieved information is given a weight based on RF, we will now look at the different IR models and their implementation.

2.5. Information Retrieval

We have two common types of IR models: vector space retrieval, and retrieval using probabilistic models. A vector space model views information and queries as vectors in Euclidean space, reducing the search in finding the closest information to a query [4]. In probabilistic models each document has been created by sampling from a probability distribution. Since the notion of relevance is at the center of IR and it is important to know how to decide relevance. We will describe these models in detail but before that, let us look at the scoring methods that describe how each term in the given query is scored.

2.6. Scoring Methods

With a large collection of information, the problem of retrieving relevant information is often difficult. In order to overcome this problem and sort the information based on the relevance of the query, a rank/score must be given. Whenever a query is sent, each term in the query has a score and its occurrence is calculated so that the relevance can be judged based on the total rank/score of information. In the IR process, there are different scoring methods used to calculate the overall score of information that was retrieved to be relevant to the given query.

2.6.1. Term Frequency Factor

If a term i occurs many times in a document then the documents having the term i will be retrieved [4] [6]. It can be expressed as,

$$\boldsymbol{tf(i,j) = no.of\ occurence\ of\ i\ in\ j} \qquad \textbf{Equation 2}$$

Occurrences where, i represent the term and j represents the document.

For example, consider a set of documents when one wants to determine which document is most relevant to the query “The big bang theory” using term frequency. A simple way to start is by eliminating documents that do not contain all the four word “the”, ”big”, ”bang”, and “theory”, but you are still left with many information relevant to these terms. So we further distinguish them with the concept of term frequency by counting the number of times each term occurs in each document and sum them all together, i.e. by the number of times a term occurs in a document.

2.6.2. Inverse-Document Frequency Factor

Let the document frequency $df(i)$ be the number of times a document in the collection contains the term i and N be the total number of documents in the collection, then the inverse document frequency will be.

$$\boldsymbol{idf(i,j) = log(\frac{N}{df(i,j)})} \qquad \textbf{Equation 3}$$

The reason for incorporating inverse-document factor is to diminish the weight of the terms that occur very frequently in the collection and increase the weight of terms that occur rarely.

Consider the same example mentioned above for term frequency. The first step will be to eliminate words that are very common in everyday usage of the sentence such as ‘the’, ‘is’ etc. The idea behind the usage of stop word is to not index words that do not contribute in any way to retrieval and scoring. For instance, the term “the” is not a good keyword to distinguish relevant and non-relevant documents. By doing this it will be able to correctly emphasize documents with words that are more meaningful and give enough weight to the words such as “big” , “bang”, “theory” , as per the example.

These two definitions can be combined to produce a composite weight for each term in each document called $tf - idf$ weighting [4] as given below.

$$\boldsymbol{w(i,j) = tf(i,j) * idf(i,j)} \qquad \textbf{Equation 4}$$

Where, i is the term and j is the document. Implementing these factors might solve the issue of identifying terms with high weights in each document in the collection but the issue of scoring them as a whole and ranking them according to the relevance will become crucial. For this reason, a basic score measure was introduced to not just add up the number of occurrences of each query term i in document j, instead the $tf - idf$ weight of each term i in document j will help retrieving relevant information corresponding to the given query [4]. The score measure is given as,

$$\boldsymbol{Score(q,j) = \sum_{i \in q}(tf - idf_{i,j})} \qquad \textbf{Equation 5}$$

The score of a document j is the sum, over all query terms, of the number of times each of the query term occurs in j [4]. So, high weight in $tf - idf$ is reached by a high term frequency (in the given document) and a low term frequency in the whole collection of documents. The weight hence tends to filter out common terms. The $tf - idf$ value for a term will always be greater than or equal to zero [4].

2.7. IR Models

2.7.1. Vector Space Model

Vector Space Model (VSM) is one of the widely used traditional IR models. It creates a space where queries and documents are represented as vectors. The relevance is computed from the distance or the angle between vectors in the given space. The VSM is fundamental to a host of IR operations ranging from scoring documents on a query, document classification and document clustering [4] [10][11].

The documents and the query in VSM are represented as $\vec{V}(d)$ and $\vec{V}(q)$. Each document can be viewed as a vector, one component for each term. The terms are axes of vector space and documents are points in this vector space [4]. In a given query, each word is

considered as a term and this term is given a weight in the set of documents in the collection which are represented as set of vectors in the vector space.

This model can be described in two stages [4]

1. The first stage is the weighting of the indexed terms to enhance retrieval of documents relevant to the user.
2. The second stage ranks the document with respect to the query according to a similarity measure.

The second stage can be achieved by using the score function. But if there are two documents with similar content then this method cannot be implemented as it will suffer from a drawback where documents with very similar content can have a significant vector difference simply because one is much longer than the other. To compensate the effect of document length, the standard way of quantifying the similarity between two documents is to compute the cosine similarity [4] [10].

2.7.2. Probabilistic model

Given a probabilistic generative model for the corpus, a probabilistic model must retrieve and rank documents. In the case of IR, an event may consist of documents that are relevant and non-relevant. So the probability of both might give an outcome for a particular event but in order to expect an event to occur based on relevancy we require a ranking principle [4] [12]. The probability ranking principle is to rank by the probability that a document is relevant. Given the probability ranking principle, if we know the probability $Pr(Rj|j)$, then we can produce results [12], where Rj represents a document j to be relevant. For example, assume if the probability that Document 3 is relevant with the rank 0.67, while the probability that Document 1 is relevant with the rank 0.12, then we can rank the two documents using those probabilities, placing 3 ahead of 1. This is basic, but there are different ranking algorithms based on the ranking principles used in probabilistic IR model. There are algorithms to retrieve relevant documents based on their scores. Such as RF, BM25 and BM25F [12].

BM25

BM25 is a ranking function of several field attributes such as term frequency, document length and document frequency. The BM25 weight looks very much like the traditional scoring method $tf * idf$.The idf is replaced by W_i^{RSJ} so that BM25 can be used with or without relevant information [12]. The term frequency differs here as it involves saturation function ($\frac{tf}{tf + k}$), where k is a constant that controls the non-linear growth of tf due to the increase in the probability of relevance of a document as the term frequency of a query term increases. For these reasons, a saturation function is used to weight the document term that appears in the query. Before applying the saturation function, the document length must be considered as it might affect tf, because document length varies based on verbosity[2] and the scope depending on the author of that document. For this reason, length normalization is applied to the document length (dl) and to the average document length ($avdl$). The length normalization is expressed as $B = ((1 - b) + b\left(\frac{dl}{avdl}\right)), 0 \leq b \leq 1$. It is used to normalize tf before applying the saturation function [12] [16].

So, $tf' = \frac{tf}{B}$ and,

$$\mathbf{W_i^{BM25}} = \frac{tf'}{tf' + \mathbf{k}} * \mathbf{W_i^{RSJ}} \qquad \textbf{Equation 6}$$

BM25F

BM25F, the state-of-the-art in structured information is an extension of the BM25 ranking function in which the documents are considered to be composed from several fields such as headlines, mail text, anchor text, etc., with possibly different degree of importance [12]. The documents are structured into a set of fields or streams. This is done so that some fields in the document will be more predictive of relevance than others. For example, a query match on the title might be expected to provide stronger evidence of possible relevance than an equivalent match on the body text. Each field relevant in the document can be taken and can

[2] Verbosity is nothing but using more words to say the same things and scope is writing a single document containing or covering more ground, [5].

then be combined together in some linear combination such as with field weights for the final document score. So a relative weight w_c is applied to each field [12].This is the same as BM25 with additional weight variant added to the saturation function.

So therefore,

$$W_i^{BM25F} = \frac{\widetilde{tf}_i}{\widetilde{tf}_i + k} * W_i^{RSJ} \qquad \textbf{Equation 7}$$

Where, $\widetilde{tf} = \sum_{i=1}^{n} w_c \frac{tf_i}{B_f}$ and B is as mentioned above in BM25 [5] and [12].

2.8. Discussion

We have just looked at various IR models and the basic scoring techniques used for ranking. We found that these models can be used for searching and ranking documents. From this we know that understanding the corpus from which we retrieve is essential in building a model for IR. So with all these details, we will implement RF using a probabilistic model with a ranking function based on term weight. For example there are different ranking functions that calculate the score of the given term to make the retrieval process more relevant to the user query.

They are [15],

Rocchio's weight:

$$score(t) = \sum_{k=1}^{r} w(t)_{Doc_k} \qquad \textbf{Equation 8}$$

Robertson Selection Value (RSV):

$$score(t) = \sum_{k=1}^{r} w(t)_{Doc_K} * P_R(t) \qquad \textbf{Equation 9}$$

Doszkocs' variant of Chi-squared:

$$score(t) = \frac{[P_R(t) - P_C(t)]}{P_C(t)} \qquad \textbf{Equation 10}$$

Kullback-Leibler distance (KLD):

$$score(t) = P_R(t) * \log\left[\frac{P_R(t)}{P_C(t)}\right] \qquad \textbf{Equation 11}$$

Where, $score(t)$ indicate the score of the query term, $P_R(t) and\ P_c(t)$ indicates the probability of occurrence of a term t in the set of relevant documents R and in the whole collection C, and $w(t)_{Doc_K}$ is the weight of the term t in document k. Note that all these function calculates the term weight based on the given relevant documents and the non-relevant documents containing the term t is negligible. But our work is focused on automatic query expansion using a ranking function by applying term weight occurring in the relevant documents. For that we will be using KLD ranking function in my model as the other function are confined to an interactive retrieval scenario by not covering automatic query expansion. The ranking function used in my model will be explained in the next chapter.

3. Text- based IR model

3.1. Outline

The purpose of this chapter is to describe a model to improve the retrieval of information using RF in a particular search application. As we have already discussed the fundamentals of a probabilistic model in Section 2, we will focus only on the details of implementing the probabilistic model with respect to the RF technique. The choice of the model and the reasons to use RF are explained in section 1.4. So, this chapter will concentrate on the measures taken to implement this model and a detailed explanation on each of the measures taken to improve the search application.

3.2. Measures

Our main focus is to make the search process efficient and improve the retrieval of relevant information, irrespective of the size of the content available. To implement RF, we need to perform a search based on the original user query. For this, a generic model will be implemented to perform the search and retrieve information based on a basic similarity scoring function used by Lucene. Later, an RF model will be implemented based on term ranking function that will compute a new query based on the user feedback. The RF model is an improvement of the first part with the notion to improve IR using user feedback.

3.3. Generic Model

For a typical search application, the three main things that must be done are indexing, searching and retrieval. To make this easier with less effort we used Lucene [8] [9]. It is a search engine library used for full text search. Lucene helps in handling these three processes and leaves the implementation part to the programmer. The implementation part involves selecting the data files, parsing the data files, getting the search query from the user and displaying the search query to the user. Figure 6 shows the usage of the different built-it functions of Lucene used for the search process.

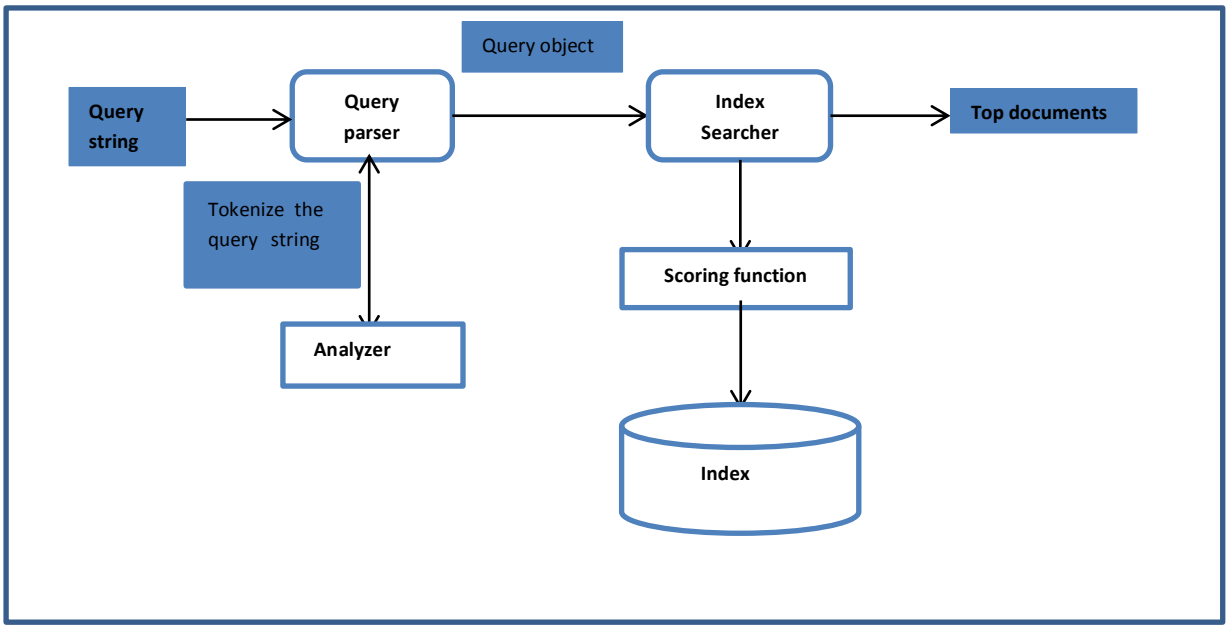

Figure 6. Search process using Lucene built-in functions.

Note that all the components used in Figure 6 are built-in functions used in Lucene [9]. When the user sends the query string, the Query parser function is called to fetch the query and generate a query object. This query objects are then parsed through another function called the Analyzer. The Analyzer avoids commonly occurring but meaningless words so that the search is done with the words that are more unique and meaningful [9]. This will increase the chance of increasing the weight of the term used rarely. The parsed query object is then fed into the Index Searcher to search for information matching the query. The Index searcher performs the searching of documents relevant to the query object [9]. The searching of the term is done by setting the similarity score to the Index searcher. It initiates a scoring function called similarity scoring function [8] and is expressed as,

$$\sum_{i \in q} (\boldsymbol{tf}(\boldsymbol{i},\boldsymbol{j}) * \boldsymbol{idf}(\boldsymbol{i})^2 * \boldsymbol{i}.\boldsymbol{getboost}(\boldsymbol{i},\boldsymbol{j}) * \boldsymbol{norm}(\boldsymbol{i},\boldsymbol{j})) * \boldsymbol{coord}(\boldsymbol{q},\boldsymbol{j}) * \boldsymbol{querynorm}(\boldsymbol{q})$$

Equation 12

It gives scores to documents and computes to determine the score for each document matching each term in a given query [8].The larger the similarity score, the better the match of the document to the given query. It is initiated every time a search function is called. The $getboost()$ is done by Lucene during the indexing of the content in the database. The $norm()$ explains the measure of the importance of a term according to the total number of terms in

the document. The $coord()$ and $querynorm()$ are calculated when the query object is sent. The $coord()$ explains the number of terms in the query that were found in the document and the $querynorm()$ acts as a normalization factor. So, when the query object is sent to the Index Searcher, it takes in the Query object and retrieves documents that have terms matching the documents. It is done by similarity scoring function set as default to the index searcher to score them according to how relevant a given document is to a user query. Then to specify the score of each document and sort them in the order of relevance, the $hits()$ function is called, it returns the set of top document that are of best match to the query [8]. The function can be set to any default value so that it sends the documents with respect to the value specified.

3.3.1. Workflow of the Generic model

Both the client and the server side of the model are explained with the help of an activity diagram. It explains the code that is implemented and the activities mentioned in these diagrams are the functions, methods and the class file declared in the code. Note that the numbers in both the diagrams explains the step by step process that is happening inside the code.

Client side

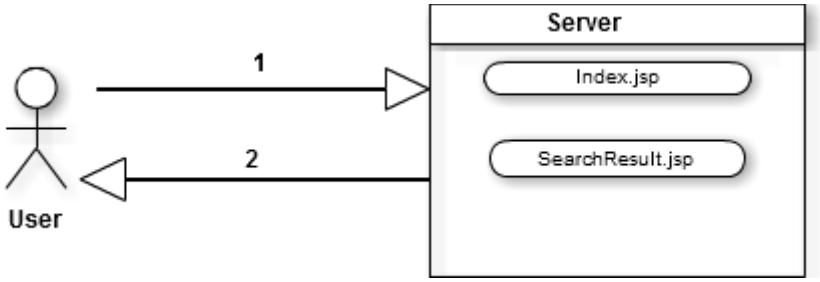

Figure 7. User interaction with the system (initial search).

1. The user initiates the search process by clicking the search button along with the search keywords (i.e. Query). This is done in 'Index.jsp'.

2. The ‘SearchResult.jsp’ displays the retrieved relevant documents to the user as results.

The next diagram explains the server side of the generic IR model using the built-in functions of Lucene to retrieve documents based on the user query search.

Server side

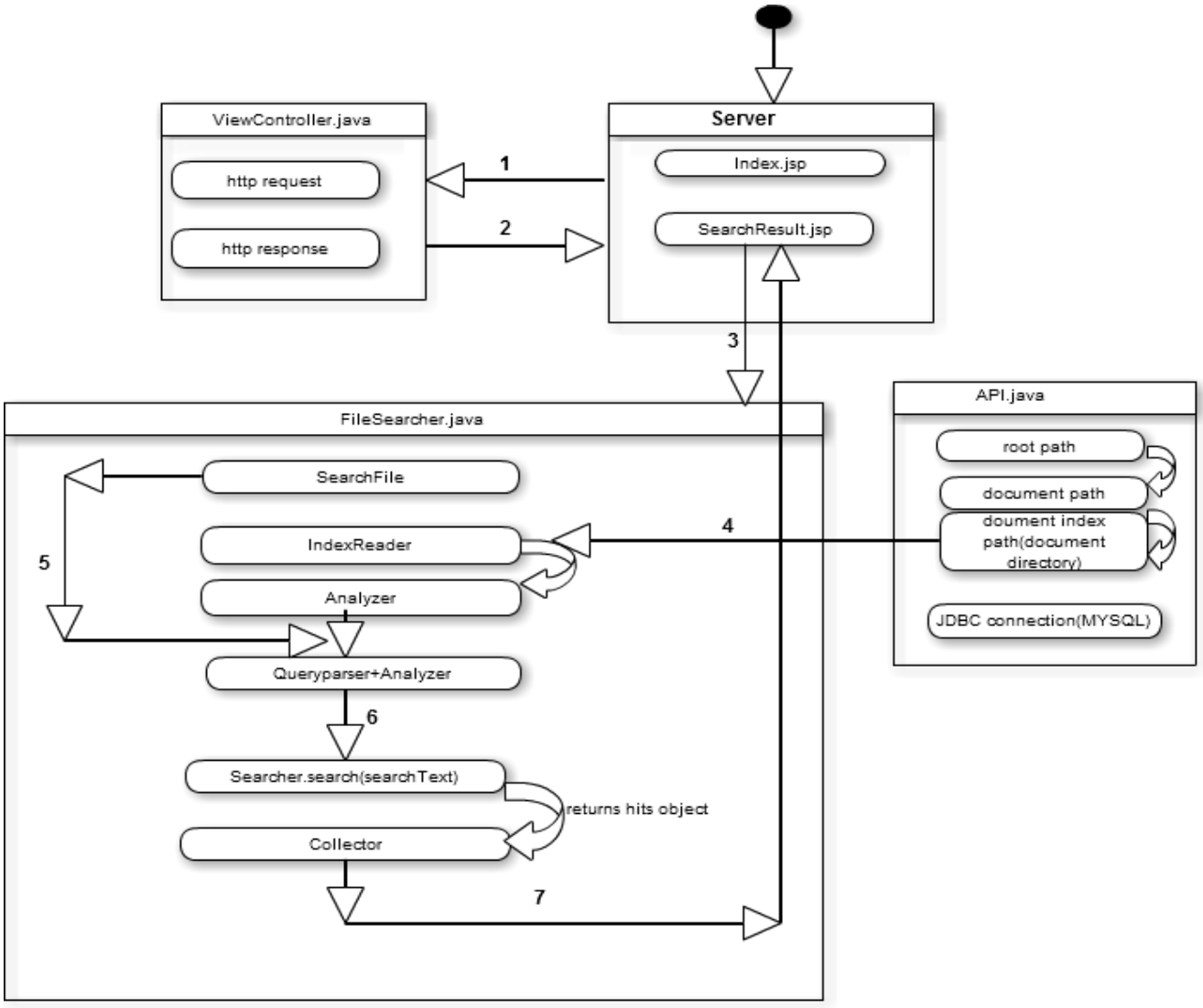

Figure 8. Server side interaction to retrieve relevant documents.

1. In ‘Index.jsp’, a form with action=ViewController is given. This operates when the search button is clicked by the user. It will submit the form to ViewController servlet in

the ‘ViewController.java’ class file. In the class file the ‘doGet’ method for *http* servlet request and *http* servlet response will get the form and search for the keys. The key searching is done in ‘Index.jsp’ as <input name=*"search"* style="width: *200px*; height: *30px*" type=*"submit"* value=*"Search"*>. Whenever a search button is clicked, the keyword written by the user as query in the text box is searched.

2. The search keyword is forwarded to the search result page (i.e., to the ‘SearchResult.jsp’). This is where the results of all the search process initiated by the user are displayed.
3. The search keyword is received and is passed to the FileSearcher object which is in the ‘FileSearcher.java’ class file. In the FileSearcher object, the ‘SearchFile’ method is called that has the search keyword declared as ‘searchText’. The ‘searchText’ consists of the query string as parameters from the ‘SearchResult.jsp’.
4. An ‘IndexReader’ for the indexed directory is created so that it will be easy to find the path of the existing index and an ‘Analyzer’ used during the index creation.
5. Now the ‘searchText’ is passed into the ‘QueryParser’ that fetches the query string and generates a query object followed by tokenizing the argument string using the ‘Analyzer’ function.
6. The query object is then fed into the search function declared as ‘Searcher.search(searchText)’. This function performs the searching of documents relevant to the query object. It returns a hit object, which is a collection of documents objects for documents that matched the search parameters. The hit objects include scores for each document. The score is computed by the Lucene scoring function that indicates how well it matches. This is stored in the collector.
7. After the search process, an html tag, pointing to ‘SearchResult.jsp’ is sent for displaying the results (retrieved relevant documents).

3.3.2. Example

Recall the example used earlier in Chapter 2.

When the user sends the query “information on circus elephant”:

1. The query $\{i_1 = information + i_2 = on + i_3 = circus + i_4 = elephant\}$ is sent as a string to the QueryParser.
2. Query parser generates query object from the string and tokenize it using the Analyzer. So the term $\{on\}$ is dropped out as it is assumed to be used frequently in every document and might increase the chance of the term used rarely (elephant, circus etc.) to have low scores. So after the parsing we have our query object with terms $\{i_1 = information + i_3 = circus + i_4 = elephant\}$. They are next sent to the Index searcher to search and retrieve the relevant documents.
3. These terms are searched in the index performing the scoring function to retrieve documents relevant to the query. Let us say the score given to the relevant documents looked like:

$Document\ Id$	$Document$	$Score$
5	$c9.txt$	0.059488226
8	$c8.txt$	0.057488126
3	$c4.txt$	0.042766831
11	$c13.txt$	0.038976235
1	$c2.txt$	0.013495873

Table 1. List of documents retrieved with their scores.

The ordering of the documents based on the relevance is done by the hit function, which sorts the documents in descending order based on the retrieved hit objects. If the hit object is set to 5 then the retrieved results will contain five top most relevant documents. Now the sorted top five documents will be sent to the user.

3.4. RF model

The retrieved result is the top documents that the system assumed one to be relevant to the user query. The probability of retrieving relevant documents in the first search is smaller, so the feedback option will improve the IR.

3.4.1. RF Services

Feedback Option

Recall RF explained in chapter 2 that talks about two kind of feedback options given by the user and the distribution of documents based on Rocchio's algorithm. The same is applied here. With the choice given by the user, the system generates the next set of relevant information. This time the retrieved results are not assumed by the systems, it instead uses the user requirement from the feedback and retrieves the results. Once the feedback is given, the documents are refined and more similar results are retrieved. When the search is done based on the feedback, a ranking function is applied and the documents are distributed using the basic concept of Rocchio's classification where a new query is generated from the original query.

Ranking/Scoring Function

As mentioned, the ranking is done with the concept of Rocchio's framework in mind but with a slight change in the distribution of the documents. Instead of moving the non-relevant documents away from the relevant documents according to Rocchio classification, they are totally ignored and only the relevant documents are considered. The ranking/scoring function implemented is called Kullback-Leibler Divergence (KLD) [14] [15].

$$\boldsymbol{w(t) = P_R(t) * \log[\frac{P_R(t)}{P_C(t)}]} \qquad \textbf{Equation 13}$$

where,

t Query term.

R Set of terms in relevant document.

C Set of terms in the whole collection.

$w(t)$ Weight of the given term *t.*

$P_R(t)$ Probability of given term t occurring in the relevant documents.

$P_C(t)$ Probability of given term t occurring in the whole collection.

This formula is called the information–theoretic approach used for automatic query expansion [15]. By automatic query expansion we mean that the system automatically expands the original query with the feedback given on the retrieved relevant documents. So with the relevance of the number of times the term occurs, the weight for that term is given and all the documents with this term are retrieved accordingly.

3.4.2. Implementation

Let us see how the above explained method is implemented in detail to retrieve top ranked documents relevant to the user query. Each term in a query is considered and given a weight based on relevancy. The weight to each term is given with a probability of the term occurring frequently. To set the weight for the given term, the following details are taken into consideration. Both R and C gives the set of terms in the relevant document and whole collection. When the user selects the search button, the system invokes the ranking function and implements $w(t)$. From Equation 13, the probability of a given term t occurring in the relevant documents and the probability of given term t occurring in the whole collection as,

$$P_R(t) = \frac{no.of\ times\ the\ term\ occuring\ in\ the\ relevant\ document}{Total\ no.of\ terms\ in\ the\ relevant\ document}$$

$$P_C(t) = \frac{no.of\ relevant\ terms\ in\ the\ index}{Sum\ of\ total\ no.of\ relevant\ term\ occuring\ in\ index}$$

Here by index we mean the data content indexed using Lucene. Taking these two factors $[P_R(t), P_C(t)]$, the total weight of the term in the relevant documents can be computed because the system compares the difference between the distribution of terms in the relevant documents and the distribution of the same terms in the whole collection. Thereby the frequency of the search term will be higher in the top relevant documents than in the whole collection, while other terms will occur with the same frequency in both document sets.

Consider our same search query used earlier:

$q = "information\ on\ circus\ elephant"$

From the initial search result (refer Table 1), each document is retrieved with a feedback option for the user to mark one or more documents and submit the relevance. The following documents from the retrieved result were given relevance as,

$document\ Id$	$document$	$Relevant/Not\ Relevant$
5	$c9.txt$	$Relevant$
8	$c8.txt$	$Not\ Relevant$
3	$c4.txt$	$Not\ Relevant$
11	$c13.txt$	$Relevant$
1	$c2.txt$	$Relevant$

Table 2. Documents marked based on relevance.

So, when the relevance is submitted the term ranking function will be implemented to form a new query. Our initial search term from the original query was $t = \{t_1 = information, t_2 = circus, t_3 = elephant\}$

Assume the probability of these terms occurring in the index $P_C(t)$ to be

$total\ no.of\ t_1\ in\ the\ index = 10$

$total\ no.of\ t_2\ in\ the\ index = 5$

$total\ no.of\ t_3\ in\ the\ index = 15$

$Sum\ of\ total\ no.of\ t_1, t_2\ and\ t_3\ occuring\ in\ the\ index = 30$

Assume the $P_R(t)$ to be

$\boldsymbol{document}$	$\boldsymbol{t_1 = information}$	$\boldsymbol{t_2 = circus}$	$\boldsymbol{t_3 = elephant}$
$c9.txt$	1	0	2
$c13.txt$	0	3	4
$c2.txt$	3	0	3

Table 3. The number of occurrence of each term from the relevant documents.

From the table above, the total number of times each terms occurring in the relevant document is given and with this, the sum of total number of terms in the relevant documents can be calculated, which is 16.

So,

$P_R(elephant) = 9/16$

$P_C(elephant) = 15/30$

$P_R(information) = 4/16$

$P_C(information) = 10/30$

$P_R(circus) = 3/16$

$P_C(circus) = 5/30$

The weight for these terms can be calculated as,

$w(elephant) = \frac{9}{16} * \log\left(\frac{9}{16}/\frac{15}{30}\right) = 0.02877$

$w(information) = \frac{4}{16} * \log\left(\frac{4}{16}/\frac{10}{30}\right) = -0.03123$

$w(circus) = \frac{3}{16} * \log\left(\frac{3}{16}/\frac{5}{30}\right) = 0.00959$

Here, the term $t_3 = information$ has a negative score, this is because the probability of the occurrence of the term t in R is smaller than the corresponding one in the entire collection. So, the term is ignored for this search session. Note that the log used in this calculation is natural logarithm. A new query is formed automatically by performing the term weight function from the given relevance feedback which will help in retrieving documents pertaining to the user requirement. Thus the search will be done on the documents matching new query formed using the ranking function which retrieves relevant documents with their score in the same way as explained in the previous model.

3.4.3. Workflow of the RF model

Both the client and the server side of the model are explained with the help of an activity diagram. It explains the code that is implemented and the activities mentioned in these diagrams are the functions, methods and the class file declared in the code. Note that the numbers in both the diagrams explains the step by step process that is happening inside the code.

Client side

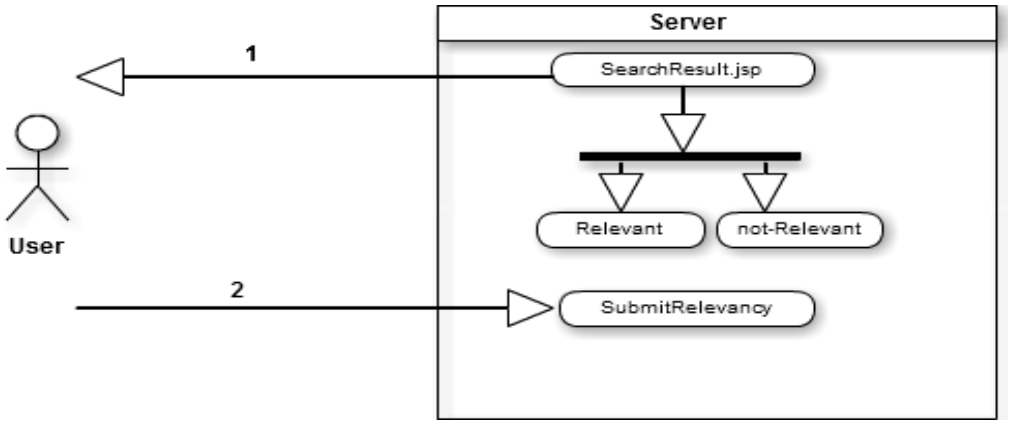

Figure 9. User interaction with the system to submit relevance.

The diagram explains the client side of the IR model using RF services,

1. The 'SearchResult.jsp' displays the set of relevant documents with the user feedback option, where the user can select documents based on relevancy.
2. The user submits the relevancy on all the retrieved documents by clicking the 'SubmitRelevancy' button.

The next diagram explains the server side of the IR model using RF services.

Server side

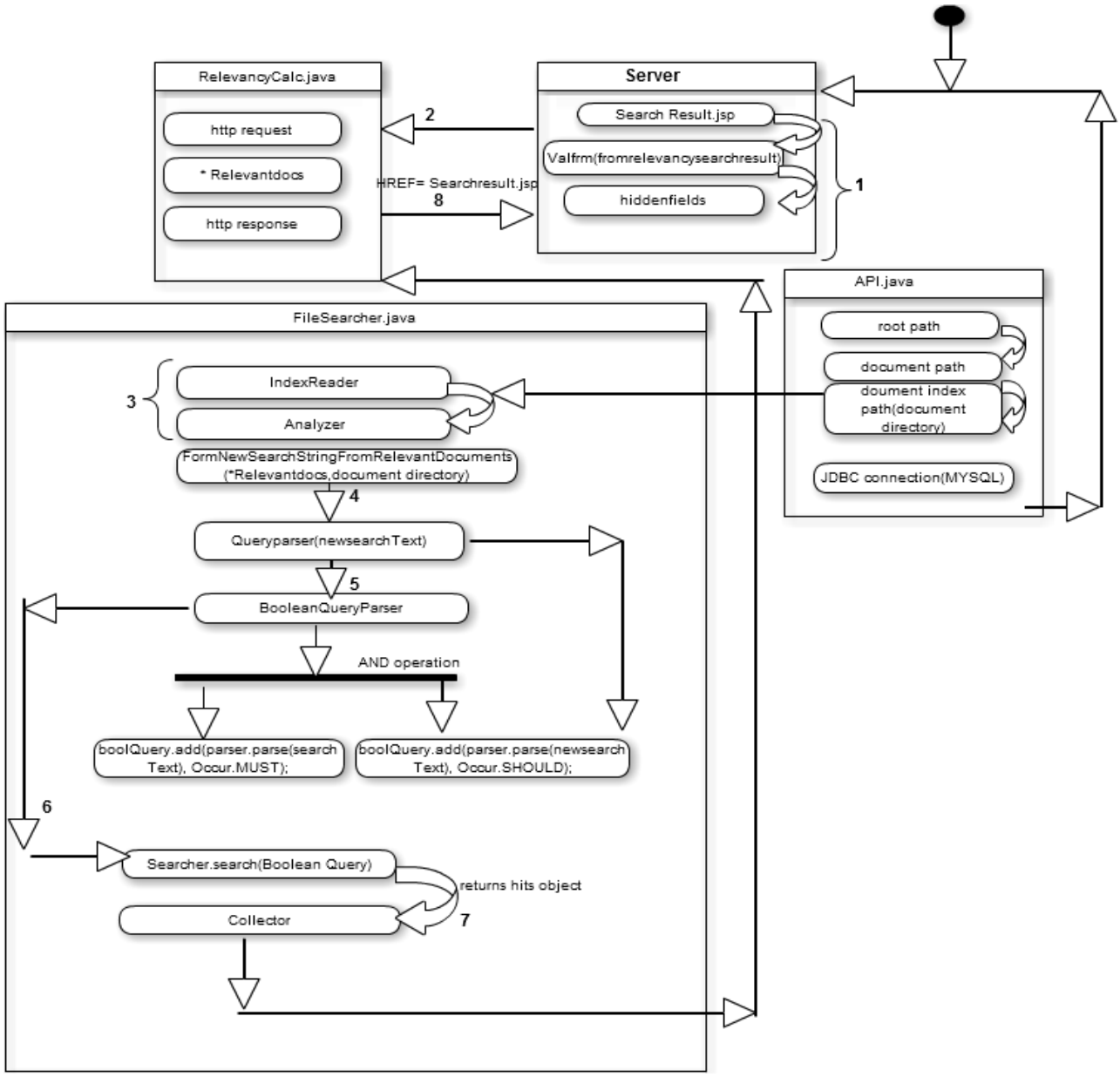

Figure 10. Interaction within the system to generate new query and retrieve relevant documents.

1. In the 'SearchResult.jsp', when the user clicks the submit relevancy button a java script will be called, 'Valfrm (frmrelevancysearchresult)'. This function will validate if the user has set relevancy to all documents and then it will store the document id and the document name to a hidden field and submits the form. The hidden file contains 'relevantDocIds', 'relevantDocNames', 'notRelevantDocIds' and 'notRelevantDocNames'.

2. The submitted form will be sent to RelevancyCalc servlet declared in the 'RelevancyCalc.java' class file. Then in the 'RelevancyCalc.java', the 'doPost ' method for *http* servlet request will get the documents and will set a session for getting only relevant documents with their id and name. It is declared as 'Relevantdocs'.
3. We know the use of 'IndexReader' and 'Analyzer' in the 'FileSearcher.java' class file. Refer to model 1 for the explanation.
4. FormNewSearchStringFromRelevantDocuments(Relevantdocs,FileDirectory) function is declared to collect new terms from the relevant documents and to form a newsearchText. The 'Relevantdocs' from the 'RelevancyCalc.java' is called in the 'FileSearcher.java' class file, to create a 'newsearchText'. Here is where the term weight is calculated. The terms in the relevant documents and the terms matching from the file directory are taken. The terms with negative score is ignored. So the new search text will have the terms with the maximum weight.
5. Now the new search text is sent to the 'Query Parser' to form the query object in order to search and retrieve the next set of relevant information. But now 'BooleanQueryParser' will be used to differentiate from the normal parsing because by implementing Query Parser, the query object parsed will be searched with terms relevant to it and if our new query object contains terms that can overtake the needed search term then the possibility to retrieve non-relevant documents is more. So if this condition is applied in the parsing method, the search will be relatively efficient. A new query object is formed by parsing the query object that was formed in the initial search text and parsing the query that was formed using the relevant documents.
6. The new query object is searched in the 'IndexSearcher' and returns the hit object matching the parsed query object using Lucene's scoring function and sorts the documents with their corresponding scores.
7. The documents are collected in the collector.
8. In the 'RelevancyCalc.java' class file the list of documents are called and the RelevancyCalc servlet performs the 'doPost' method for http servlet response, where a Hrefs to 'SearchResult.jsp' is given that will display the results. The process might be iterative until the user has got the required information.

Thus a solution to retrieve information based on user relevance feedback was found. But this cannot be concluded to be the best approach. This can be further modified to make it more efficient when there are large contents available in the index. Our next section will explain the analysis and the evaluation that was done using this method.

4. Analysis and Performance

The purpose of this section is to analyze the performance of the models implemented in the previous chapter followed by the results from both the models and finally to assess the pros and cons of using these two models. Note that these models were tested by me and the word 'user' mentioned in these models refers to me and 'user query' is given by me as well. This was not trialed with multiple real users.

4.1. Generic Model

Since our focus is to make the search process efficient and improve the retrieval of information using RF we divided the task into two and created two models even though it is treated as one model.

4.1.1. Analysis

As the focus is to improve the search functionality, the main thing needed is to index the available data content to make the search process easier. This was achieved using Lucene as it made the process smoother with less effort. As our model is based on text-based IR any kind of text format can be indexed into the index. Searching is made efficient by using Lucene's scoring function to score documents with respect to the query sent by the user. The complexity of the scoring process is hidden from the user and only the score of each document is presented to the user with a brief view on how relevant the documents are to a query. This model will be effective whenever a search application is built from the scratch.

4.1.2. Performance

This model is efficient in searching and retrieving the information. So when a query is sent, the system is able to search the information relevant to the query and retrieve relevant information with their scores to view the documents in the order of relevance. The results from my model showed how the documents are retrieved in the order of relevance to the user query. It is understood by the scores given to the matched documents. The score that were given to these models were checked with multiple tries on the same query in different session, to see if the scores changes. It did not until and unless the query was changed. So

the scores given to each documents were understood to be based on the terms in the given query. The following result was obtained from this model.

$$q = information + on + circus + elephant$$

$$Hits\ for\ \text{"information on circus elephant"}\ were\ found\ in\ quotes\ by$$

docId	*docName*	*docScore*
docId: 12	*docName : 17.txt*	*docScore: 0.08842075*
docId: 5	*docName : c9.txt*	*docScore: 0.03824398*
docId: 8	*docName : script.txt*	*docScore: 0.034033164*
docId: 1	*docName : c11.txt*	*docScore: 0.03245109*
docId: 3	*docName : c7.txt*	*docScore: 0.022080172*

Table 4. Results obtained from generic model (results based on the initial user query)

The retrieved results are the documents that the system assumed to be relevant. The reason for only five retrieved lists of document is that the 'Hits' per page was set to five in the code due to limited availability of information in the index. The retrieved documents can be a mixture of relevant and non-relevant documents. But with this model the user is not loaded with information (documents) but rather given a sorted set of information that the system assumed to be relevant. Added to it, the scoring makes it more useful as the information is ranked in the descending order of relevance.

4.2. RF model

4.2.1. Analysis

The focus to improve the IR using RF techniques is efficient and useful in cases where the user and the system try to understand the actual information that is required. By using the RF services such as feedback and implementing the ranking function, and KLD to rank the documents based on the user feedback, the retrieval process is improved.

4.2.2. Performance

This model retrieved relevant information from the given feedback when the user felt that the document was relevant. The information is retrieved when the search button is clicked by the

user that invokes the ranking function used in the model to calculate the weight of each query term refined from the initial search and applying the term weight to the documents having them and retrieving them with their scores in the order of relevancy. The table below shows the information retrieved from the initial search with the user feedback.

$docId$	$docName$	$Relevant/Not\ Relevant$
12	$17.txt$	$Relevant$
5	$c9.txt$	$Not\ Relevant$
8	$script.txt$	$Not\ Relevant$
1	$c11.txt$	$Relevant$
3	$c7.txt$	$Relevant$

Table 5.Retrieved documents with user feedback.

After selecting the documents, the “submit relevancy” button was clicked and the following results were retrieved

relevant Doc IDS: 12, 5, 1

Getting Terms on Document: 12

Getting Terms on Document: 5

Getting Terms on Document: 1

8 total matching documents

Hits for "information on circus elephant" were found in quotes by:

docId	*docName*	*docScore*
docId: 12	*docName : 17.txt*	*docScore: 1.8727843*
docId: 1	*docName : c11.txt*	*docScore: 0.5950377*
docId: 9	*docName : c13.txt*	*docScore: 0.45690736*
docId: 4	*docName : c8.txt*	*docScore: 0.1383166*
docId: 0	*docName : c10.txt*	*docScore: 0.06271084*

Table 6 . Results obtained from RF model (results based on the newly generated query).

There is a difference in the scores of the selected relevant documents from Table6. This is because the term weight of the query terms are calculated and added to the documents having those terms. Another important point is that the retrieval process is effective with RF using the KLD scoring function where the documents are diverged to get the most relevant documents not only within the relevant documents but also from the index.

4.3. Differences

The difference between these two models is that in the first model the information that is retrieved are assumed to be relevant by the system but in the second model the retrieved information is not assumed in priori, but based on the feedback given by the user. In both models, term weight is calculated based on the given term, the difference will be that the first model implements basic search and retrieval of information whereas in the second model the term weight is calculated with respect to information refined by the user and then search is done with respect to the retrieved list of relevant documents and the index. By doing this, the possibility of missing any documents is reduced and the user is given a large variety to choose.

5. Future work and Conclusion

5.1. Discussion

Targeted at a very specific application scenario, namely the real-time learning from user interaction during information retrieval and RF as classification or learning problem possesses very unique characterization and difficulties. A successful algorithm is one that is tailored to address these special issues. Both the models were implemented using specific algorithms to perform efficiently, for small datasets, in their own perspective. Although these two models are implemented at different stages of the search process they are still seen as one model. The reason behind this is because the two models are declared as two different methods in the code that will be called with respect to the search pattern. So for the initial search process, the first method is initiated/called that performs normal search that retrieves documents based on the similarity scoring function and the second method is initiated/called after the 'Submit Relevancy' button is submitted by the user were the query terms in the relevant documents are given weight and the documents are retrieved based on the similarity function. Since models use the same similarity scoring function to retrieve the documents, the code can be terminated after the first model itself just implementing the basic RF services, but the problem of retrieving relevant information to the user requirement might not be achieved. For this reason the second model is implemented so that the user is given a large choice of information that the user can choose relevant to his/her requirement. So by this, the aim to improve the search functionality and retrieve information based on RF is achieved to some extent. The model was not tested with multiple users connected with large content or through TREC[3] .

[3] TREC (Text Retrieval Conference) that provides the infrastructure to large-scale evaluation of text retrieval methodologies.

5.2. Future Work

There are still open issues that could not be developed in this thesis, and are work for the future.

- The model can be improved by saving user feedback so that next time the search on the same information will help get information faster.
- The search functionality can be improved more by providing similar search on each of the relevant information apart from the feedback option.
- The retrieved documents can be annotated with one or more tags so that the user is given more knowledge on the information needed.

5.3. Conclusion

A full text-based information retrieval model is presented that is efficient in both searching and retrieval of information. In this thesis, the various IR methods and models were studied to understand the technical background of the ideas and concepts that is used in the model followed by implementing an IR model to improve the search and retrieval functionality by getting RF from users, in order to confirm or reject the documents retrieved from the model. Then an RF ranking function was implemented on the user query and generated a new query to get more similar information based on the selected relevant documents. Then the model was analyzed to see if the results retrieved relevant information based on the user requirement. It can be inferred from the performance of this model that the study of user feedback in IR can be very good because the information searching is done based on the feedback given by the user.

References

[1] Competitive and Innovation Framework Programme (2010) *ASSETS-Description of work.*

[2] ASSETS WP and Task leads (2010) *Interface Specification and System Design-ASSETS.D2.0.2.AIT.WP2.V1*.

[3] Amaia Portugal (2011), *Contributor Enhancing Europeana, the great European digital library*, CORDIS-Community research and Development Information Service 27, January 2011.

Available at http://cordis.europa.eu/wire/index.cfm?fuseaction=article.Detail&RCN=25434

[4] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze (2009) *An Introduction to Information retrieval* New York, NY, USA: Cambridge University Press.

Avaialble at http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[5] Joaquin Pérez-Iglesias, Jose R. Perez-Agure, Victor Fresno, Yuval Z. Feinstein (2009). *Integrating the Probabilistic Models BM25/BM25F into Lucene.*

Avalaible at http://arxiv.org/abs/0911.5046

[6] S.E. Robertson, K. Spärck Jones (1994) *Simple, proven approaches to text retrieval* Microsoft Research Ltd, Cambridge and Computer Laboratory, University of Cambridge.

Available at http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-356.pdf

[7] Dr. David Grossman, Michael Lee (2001) *CS 529: Information Retrieval- Relevance Feedback*.

Available at http://www.ir.iit.edu/~dagr/cs529/files/handouts/07Feedback.pdf

[8] Michael McCandless, Erik Hatcher, Otis Gospodnetic (2010) *Lucene in Action* Manning Publication Co, Stamford. ISBN 978-1-933988-17-7.

[9] Steven J. Owens (2011) Lucene *Tutorial*.

Available at http://darksleep.com/lucene/

[10] Jan Larsen (1999) *Vector Space Model*. THOR Center for Neuroinformatics-projects-multimedia-text mining-introduction.

Available at http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html.

[11] Dr. Ian Soboroff (2002) *IR Models: The Vector Space Model*.

Available at http://www.cs.umbc.edu/~ian/irF02/lectures/07Models-VSM.pdf.

[12] Stephen Robertson and Hugo Zaragoza (2009). *The probabilistic Relevance Framework: BM25 and Beyond*, Foundation and Trends in Information Retrieval,Vol. 3,pg 333-389.

[13] V. Karyotis, T. Kasrinogiannis, G.Androulidakis, C.Malavazos, M.Lazaridis (2007) *VICTORY- Hypertech-D-WP4-V9-D4.4 Relevance Feedback Algorithms*.

[14] Claudio Carpineto, Renato De Mori. Giovanni Romano and Brigitte Bigi (2001) *Information- Theoretic Approach to Automatic Query Expansion*, ACM Transaction on Information Systems, Vol.19, No.1, Pages 1-27.

Available at citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.2399&rep=rep1&type=pdf

[15] Zaragoza, J. R.-A. (n.d.) (2008) *Robust and WSD tasks,* University Complutense of Madrid and Yahoo! Reseach.

Available at

http://www.clef-campaign.org/2008/working_notes/aguera-paperCLEF2008.pdf

[16] Vistor Fresno, Joaquin Perez Iglesias, Jane Greenberg, Javier Arroyo, Jose R. Perez- Aguera *Using BM25F for Semantic Search*.

Available at http://km.aifb.kit.edu/ws/semsearch10/Files/bm25f.pdf.