University of Glasgow | School of Computing Science

Mining Academic Expertise from Funded Research (Expert Search System)

Peeranat Fupongsiripan 2056647

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 28, 2014

Abstract

This project is concerned with development of mining academic expertise in Scottish Universities from funded research. This system can be used to extract, analyse and find experts in particular areas in Scottish Universities. http://experts.sicsa.ac.uk/ is an existing academic search engine that assists in identifying the relevant experts within Scottish Universities, based on their recent publication output. The aim of this project is to develop mining tools for the data, and research ways to integrate it with existing deployed academic search engine to obtain the most effective search results.

Acknowledgements

The dissertation would not have been completed without the help and support that I have received. First of all, I would like to thank Dr. Craig Macdonald for his advice and supervision throughout this project. He always encouraged me to work hard and helped me overcome difficulties encountered during the development of this project. Second of all, I would like to thank Dr. Iadh Ounis for his advice about this project. Finally, I would like to thank Professor. Joemon Jose and Dr. Simon Rogers whose lectures are of use for the scope of this project.

Contents

Chapter 1

Introduction

This chapter is about the introduction of the project. It will give an overview of The Scottish Informatics and Computer Science Alliance (SICSA), the existing search engine, definition of the project and aims. It will also outline the structure of the project.

1.1 The Scottish Informatics and Computer Science Alliance (SICSA)

"The Scottish Informatics and Computer Science Alliance (SICSA) is a collaboration of Scottish Universities whose goal is to develop and extend Scotland's position as a world leader in Informatics and Computer Science research and education" [13]. SICSA achieves this by working cooperatively rather than competitively, by providing support and sharing facilities, by working closely with industry and government and by appointing and retaining world-class staff and research students in Scottish Universities. A list of members of SICSA is given in table 1.1.

1.2 What is Expert Search?

With the enormous in the number of information and documents and the need to access information in large enterprise organisations, "collaborative users regularly have the need to find not only documents, but also people with whom they share common interests, or who have specific knowledge in a required area" [26, P. 388]. In an expert search task, the users' need, expressed as queries, is to identify people who have relevant expertise to the need [26, P. 387]. An expert search system is an Information Retrieval [2] system that makes use of textual evidence of expertise to rank candidates and can aid users with their "expertise need". Effectively, an expert search systems work by generating a "profile" of textual evidence for each candidate [26, P. 388]. The profiles

University
University of Aberdeen
University of Abertay
University of Dundee
University of Edinburgh
Edinburgh Napier University
University of Glasgow
Glasgow Caledonian University
Heriot-Watt University
Robert Gordon University
University of St Andrews
University of Stirling
University of Strathclyde
University of the West of Scotland

Table 1.1: A list of members of SICSA

represent the system's knowledge of the expertise of each candidate, and they are ranked in response to a user query [26, P. 388]. In real world scenario, the user formulates a query to represent their topic of interest to the system; the system then uses the available textual evidence of expertise to rank candidate persons with respect to their predicted expertise about the query.

1.3 Definition of Mining Academic Expertise from Funded Research and Aims

http://experts.sicsa.ac.uk/ [12] is a deployed academic search engine that assists in identifying the relevant experts within Scottish Universities, based on their recent publication output. However, integrating different kinds of academic expertise evidence with the existing one may improve the effectiveness of the retrieval system. The aim of this project is to develop mining tools for the funded projects, and research ways to integrate them with the existing academic search engines to obtain the most effective search results. The sources of the new evidence, funded projects, are from Grant on the Web [1] and Research Councils UK [11]. To integrate academic funded projects and publications together, Learning to Rank Algorithms for Information Retrieval (IR) are applied in this project.

1.4 Context

This project was initially developed by an undergraduate student a few years ago. It used academic's publications as an expertise evidence to find experts. I have access to funded projects data in the UK. This data is integrated with existing data to improve the effectiveness of http://experts.sicsa.ac.uk/ [12] at answering expert search queries. The internal name of the codebase is *AcademTech*. Some chapters might use this name.

1.5 Overview

In this dissertation, section 2 aims to explain the backgrounds of the project to readers. Section 3 includes discussions of system and interface designs and architecture and proposals to the new system. This section is necessary for readers to understand other sections. Section 4 discusses about implementations of the system and new system user interface design. Section 5 provides results and analysis of the techniques used. This section can be viewed as the most important part of the whole project since it analyses whether applying a learning to rank technique improves the effectiveness of the system or not. The last section is the conclusions of the project.

Chapter 2

Background

This chapter is aimed to give backgrounds required to understand the rest of the chapters. It begins with Information Retrieval (IR) and Search Engine (Section 2.1). Then Section 2.2 will dicuss about Learning to Rank (LETOR) In Information Retrieval. Tools used in this project will be briefly discussed in Section 2.3. And finally, Section 2.4 will discuss expert search and techniques related to it in more details.

2.1 Information Retrieval (IR) and Search Engine

"Information Retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources" [2]. An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. However, the submitted query may not give the satisfying results for the user. In this case, the process begins again. Figure 2.1 illustrates search process. As information resources were not originally intended for access (Retrieval of unstructured data)[P. 7] [29], it is impossible for a user query to uniquely identify a single object in the collection. Instead, several objects may match the query, with different degrees of relevancy. In the IR field, there are various types of retrieval models used to compute the degree of relevancy. This will be discussed in more details in section 2.1.2.

A search engine is an information retrieval system designed to help find information stored on a computer system [22]. The search results are usually presented in a list ordered by the degree of relevancy and are commonly called hits. Search engines help to minimize the time required to find information and the amount of information which must be consulted [22]. The special kind of search engine is web search engine. It is a software system that is designed to search for information on the World Wide Web [25] such as Google, Bing, and Yahoo.

2.1.1 Brief Overview of Information Retrieval System Architecture

In IR systems, two main objectives have to be met [28] - first, the results must satisfy user - this means retrieving information to meet user's information need. The second, retrieving process must be fast. This section is devoted to a brief overview of the architecture of IR systems which is very important to meet the IR main objectives. It also explains readers how documents are retrieved and the data structure used in IR systems. To understand how retrieval process works, we must understand indexing process first. This process is performed offline and only one time. There are 4 steps involved in indexing process and each process is performed sequentially [28]:

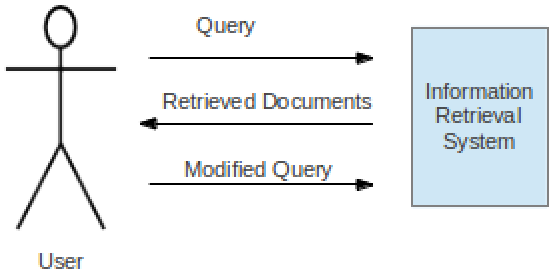


Figure 2.1: Search Process

```
<DOC>
<DOCNO>1</DOCNO>
<CONTENT>
There are only two ways to live your life. One is as though nothing
is a miracle. The other is as though everything is a miracle.
</CONTENT>
</DOC>
```

Figure 2.2: Document

1. Tokenisation
2. Stopwords Removal
3. Stemming
4. Inverted Index Structure Creation

Given a document containing Albert Einstein's quote about life, it can be illustrated in a terms-frequency table.

> *There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.*

Term	frequency
there	1
are	1
only	1
two	1
ways	1
live	1
your	1
life	1
to	1
one	1
is	3
as	2
though	2
nothing	1
a	2
miracle	2
the	1
other	1
everything	1

Table 2.1: Terms and Frequency

Term	frequency
two	1
ways	1
live	1
life	1
one	1
nothing	1
miracle	2
everything	1

Table 2.2: Terms and Frequency After Stopwords Removal

Term	frequency
two	1
way	1
live	1
life	1
one	1
nothing	1
miracle	2
everything	1

Table 2.3: Terms and Frequency After Stemming

Table 2.1 shows all the terms and frequency of each term in the document. It can be seen that there are some words in the document which occur too frequently. These words are not good discriminators. They are referred to as "stopwords". Stopwords include articles, prepositions, and conjunctions etc.

Tokenisation is the process of breaking a stream of text into words called tokens(terms) [24]. The stream of text will be used by other indexing steps.

Stopwords Removal is the process of removing stopwords in order to reduce the size of the indexing structure [28, P. 15]. Table 2.2 shows all the terms and frequency of each term after stopwords removal process.

Stemming is the process of reducing all words obtained with the same root into a single root [28, P. 20]. A stem is the portion of a word which is left after the removal of its affixes (i.e. prefixes and suffixes). For example, connect is the stem for the variants connected, connecting, and connection. This process makes the size of the data shorter. There are various stemming algorithms such as Porter Stemming, and Suffix-stripping algorithms.

After stemming, all terms in the table are in its root forms. If a document is large in size, this process can reduce the size of the data considerably. However, there is one drawback. That is, it prevents interpretation of word meanings. For instance, the root form of the term "gravitation" is "gravity". But the meaning of "gravitation" is different from "gravity".

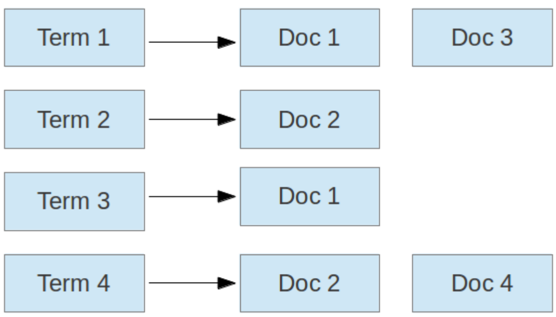


Figure 2.3: Simple Inverted Index

Inverted Index Structure Creation is the process that creates an index data structure storing a mapping from terms(keys) to its locations in a database file, or in a document or a set of documents(values) [18]. The purpose of this data structure is to allow a full text searches. In IR, a value in a key-value pair is called posting. There are a number of index structures used in practice. However, the index used in most IR systems and in this project is inverted index. Figure 2.3 shows a simple inverted index. Given a query (a set of terms), it is now possible to efficiently search for documents containing those terms. However, each posting may contain additional information or features about a document such as the frequency of the term etc.

2.1.2 Retrieval Models

In the previous section, basic indexing process was briefly explained. In this section, we will give a brief introduction to a few retrieval models including one used in this project. In general, retrieval models can be categorised into 2 categories: probabilistic approach and non probabilistic approach. This section will briefly explain Term Frequency–Inverse Document Frequency (tf-idf), BM25 and PL2 retrieval models.

Term Frequency–Inverse Document Frequency (tf-idf)

tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection [23]. As the name suggests, it consists of 2 parts: term frequency (tf) and inverse document frequency (idf). Term frequency is the number of occurrences a term appears in a document. Inverse document frequency (idf) is a measure of whether the term is common or rare across all documents [23]. This component is very important for instance, if a query term appears in most of the documents in the corpus, it is not appropriate to give a document containing that term a high score because that term is not a very good discriminator. On the other hand, it is appropriate to give high scores to documents containing terms rarely appear in the corpus. The following is the formula of **tf-idf** weighting model:

$$W_{fk} = f_{fd} log \frac{N+1}{D_k+1} \tag{2.1}$$

where N is the number of documents in the collection, f_{fd} is tf of k^{th} keyword in document d (term frequency), and D_k is the number of documents containing k^{th} keyword. The $log \frac{N+1}{D_k+1}$ is the idf of the model. The numerator and denominator are added by 1 to prevent possibility of zero. This model is a non-probabilistic approach and is one of the easiest to implement models in IR.

BM25

BM25 is a very popular probabilistic retrieval model in IR. Why use probabilities? In Section 2.1, we explained that IR deals with uncertain and unstructured information. In other words, we do not know specifically what the documents are really about. As a consequence, a query does not uniquely identify a single object in the collection. Therefore, probability theory seems to be the most natural way to quantify uncertainty [31, P. 7]. Based on a derivation from the probabilistic theory, the formula for BM25 is shown below:

$$score(d, \vec{q}) = \sum_{t \in \vec{q}} log_2 \frac{r_t/(R - r_t)}{(n_t - r_t)/(N - n_t - R + r_t)} \frac{(k_1 + 1) \cdot tf_{td}}{K + tf_{td}} \frac{k_2 + 1) \cdot tf_{tq}}{k_2 + tf_{tq}} \tag{2.2}$$

where $K = k_t((1-b)+b\frac{dl}{avdl})$, tf_{tq} and tf_{td} are the frequency of term t in the query $\vec{q}$ and document d respectively, and b, dl and $avdl$ are parameters set empirically. r_t is the number of documents containing term t relevant to a query term t. R is the number of documents in the collection relevant to a query term t. n_t is the number of documents in which the query term t appears and N is the number of all documents in the collection. The proof of BM25 is beyond the scope of the project.

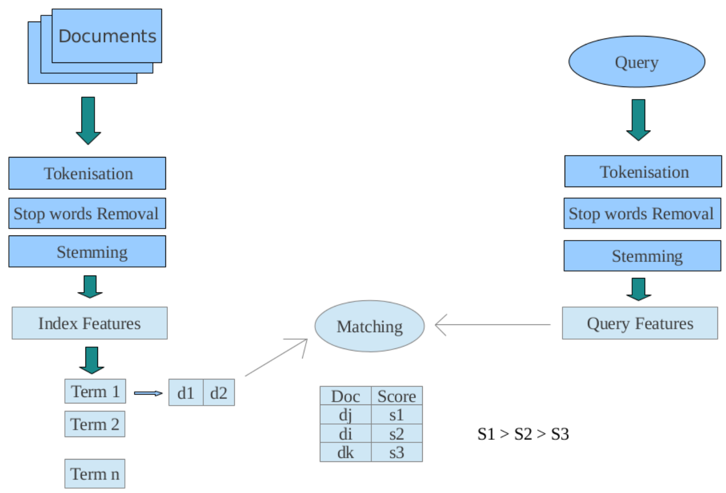


Figure 2.4: Retrieval Process

PL2

PL2 is a model from the divergence from randomness framework, based on a poisson distribution [19]. This project uses PL2 weighting model to calculate document scores. This model is also a probabilistic approach. Given a document d and a query $\vec{q}$, the formula of PL2 [34, P. 23-24] is given below:

$$score(d,\vec{q}) = \sum_{t\in\vec{q}} qtw \cdot \frac{1}{tfn+1}(tfn \cdot log_2 \frac{tfn}{\lambda} + (\lambda - tfn) \cdot log_2 e + 0.5 \cdot log_2(2\pi \cdot tfn)) \tag{2.3}$$

where λ is the mean and variance of a Poisson distribution, and the query weight term qtw is given by qtf/qtf_{max}. qtf is the query term frequency. qtf_{max} is the maximum query term frequency among the query terms. From equation 2.3,

$$tfn = tf \cdot log_2(1 + c \cdot \frac{avgl}{l}), c > 0$$

where tf is the actual term frequency of the term t in document d and l is the length of the document in tokens. $avgl$ is the average document length in the whole collection. c is the hyper-parameter that controls the normalisation applied to the term frequency with respect to the document length. The proof and explanation are out of the scope of the project.

2.1.3 Retrieval Process in Information Retrieval

Section 2.1.1 and Section 2.1.2 explained basic indexing process and a few retrieval models respectively. In this section, we will see how documents are retrieved. Figure 2.4 shows retrieval process of IR system. In IR, there are 2 phases in general: online and offline phases. The offline phase is the phase that all documents in the corpus are indexed, all features are extracted and index structure is built (Section 2.1.1).

On the other hand, online phase begins after a user submits a query into the system. After that, tokenisation, stopwords removal and stemming processes are performed as same as the offline phase. Features can also be extracted from query terms as well. At this point, it comes to the process of matching and assigning scores to documents. This process can make use of one of the retrieval models explained in Section 2.1.2. In this project, PL2 weighting model and voting technique which will be explained in Section 2.4.2 are used to compute scores. Once scores have been assigned to relevant documents, the system ranks all documents in order of decreasing scores and show to the user. Figure 2.4 gives a graphical representation of retrieval process.

2.1.4 Evaluation

This section is devoted to backgrounds of evaluation of IR systems. It is very important as it is a background for Evaluation Chapter. Since IR is research-based, understanding how evaluation is carried out will enable readers to determine whether this project is achieved or not. There are 3 main reasons for evaluating IR systems [30, P. 3]:

1. Economic reasons: If people are going to buy the technology, they want to know how effective it is.
2. Scientific reasons: Researchers want to know if progress is being made. So they need a measure for progress. This can show that their IR system is better or worse than someone else's.
3. Verification: If an IR system is built, it is necessary to verify the performance.

To measure information retrieval effectiveness in the standard way, a test collection is required and it consists of 3 things [17]:

1. A document collection.
2. A test suite of information needs, expressible as queries.
3. A set of relevance judgments, usually a binary assessment of either relevant or nonrelevant for each query-document pair.

The standard approach to information retrieval system evaluation revolves around the notion of relevant and nonrelevant documents. With respect to a user's information need, a document in the test collection is given a binary classification as either relevant or nonrelevant [17]. However, this can be extended by using numbers as an indicator of the degree of relevancy called **graded relevance value**. For example, documents labelled 2 is more relevant than documents labelled 1, and documents labelled 0 is not relevant. In IR, a binary classification as either relevant or nonrelevant and graded relevance value are called **relevance judgement**. There are a number of test collection standards. In this project, Text Retrieval Conference (TREC) [15] is used since it is widely used in the field of IR.

Precision and Recall

The function of an IR system is to [30, P. 10]:

- retrieve all *relevant documents* measured by **Recall**
- retrieve *no non-relevant documents* measured by **Precision**

Precision (P) is the fraction of retrieved documents that are relevant

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

Recall (R) is the fraction of relevant documents that are retrieved

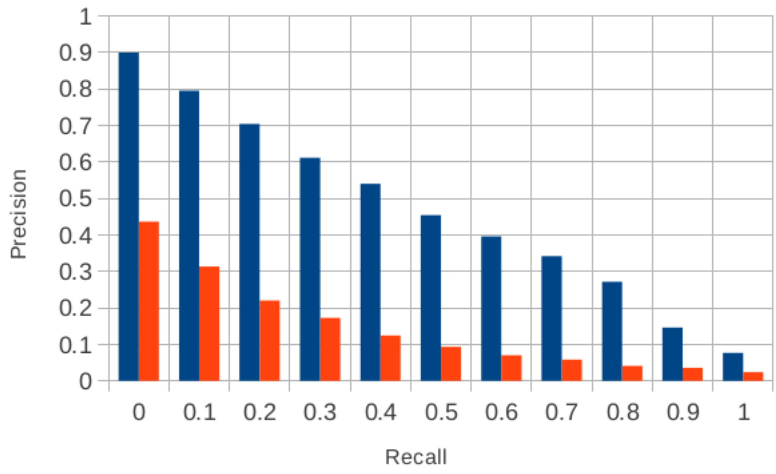


Figure 2.5: Precision-Recall Graph

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

If a system has high precision but low recall, the system returns relevant documents but misses many useful ones. If a system has low precision but high recall, the system returns most relevant documents but includes lots of junks. Therefore, the ideal is to have both high precision and recall. To give a good example, consider Figure 2.5, since overall IR system A (blue) has higher precision than IR system B (red), system A is better than system B.

However, in certain cases, precisions of system A may have higher values than system B in some recall points or vice versa. Therefore, Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG) and Mean Reciprocal Rank (MRR) are used to address this problem. Each of them has different behaviours of evaluation.

Mean Average Precision (MAP)
MAP for a set of queries is the mean of the average precision scores for each query [2]. The equation below is a formula for MAP.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP(q)}}{Q}$$

where Q is the number of queries.

Mean Reciprocal Rank (MRR)
MRR is a statistic measure for evaluating a list of possible responses to a set of queries, ordered by probability of correctness [21]. The equation below is a formula for MRR.

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

where $rank_i$ is the position of the correct result and Q is the number of queries.

Normalized Discounted Cumulative Gain (NDCG or nDCG)
To understand NDCG, first of all, we have to understand **Discounted Cumulative Gain (DCG)**. The premise of DCG is that highly relevant documents appearing in lower position in a search result list should be penalized as the graded relevance value (discussed above) is reduced logarithmically proportional to the position of the result. [20]. The discounted CG accumulated at a particular rank position p is defined as:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

where rel_i is the graded relevance value of the result at position i.

From DCG, we can formulate NDCG as follows:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

where $IDCG_p$ is the maximum possible (ideal) DCG for a given set of queries, documents, and relevances.

2.2 Learning to Rank (LETOR) In Information Retrieval

Learning to rank or machine-learned ranking (MLR) is a type of supervised or semi-supervised machine learning problem in which the goal is to automatically construct a ranking model from training data [35]. Employing learning to rank techniques to learn the ranking function is important as it is viewed as a promising approach to information retrieval [35]. In particular, many learning to rank approaches attempt to learn a combination of features (called the learned model) [27, P. 3]. The resulting learned model is applied to a vector of features for each document, to determine the final scores for producing the final ranking of documents for a query [27, P. 3]. In LETOR, there are typically 2 types of dataset: training and testing dataset. They both consist of queries and documents matching them together with relevance degree of each match [4] and are typically prepared manually by human assessors, who check results for some queries and determine relevance of each result or derived automatically by analyzing clickthrough logs (i.e. search results which got clicks from users). However, they differ in the purpose of usage.

- *Training data* is used by a learning algorithm to produce a ranking model or learned model [4]. This will discuss in more details in section 2.2.5
- *Testing data* is applied with a learned model and used to evaluate the effectiveness of the system [19].

2.2.1 Query Dependent Feature

Figure 2.1 shows a simple search process in IR. After a user submits a query into an IR system, the system ranks the documents with respect to the query in descending order of the documents' scores and returns a documents set to the user. It can be clearly seen that the documents obtained with respect to the query depends on the query the user submitted. In other words, document A can have 2 different degrees of relevancy if a user changes a query. In learning to rank, this is called query dependent feature [19]. To give a good example, consider an IR system using tf-idf retrieval model (see Section 2.1.2). From equation 2.1, given a document d, query $\vec{q} = \{database\}$, $f_{database,d} = 5$, $N = 120$, and $D_{database} = 23$, the score of document d is 3.51. If a query is changed to $\vec{q} = \{information\}$, then the score of document d is possibly different depending on the parameters in the formula.

2.2.2 Query Independent Feature

In contrast to Query Dependent Feature, a feature that does not depend on a user query is called query independent feature [19]. This feature is fixed for each document. To give a good example, consider an algorithm used by Google Search to rank websites in their search engine results. This algorithm is called *PageRank* [6]. According to Google, how PageRank works is given below [6]:

> *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.*

In IR, the link of a website is actually a document. It is obvious that PageRank takes query independent features into account which are number of links and quality of links to a website.

2.2.3 Learning to Rank File Format

From the previous 2 sections, we now know what query dependent and query independent features are. In this section, we will talk about how these features are formatted in a file called **LETOR file**. The LETOR file is then learned to obtain a model file (see next section). The following is the format of LETOR file:

```
#feature_num1:feature_name1
#feature_num2:feature_name2
graded_relevance_value qid:query feature_num1:feature_name1_score feature_num2:feature_name2_score
#docno = document_id
```

In this LETOR file, there are 2 features, *feature_name1* and *feature_name2*. In LETOR file, items after a hash symbol are ignored. The lines not preceded by hash symbol are learned. They represent documents with scores of each feature. *graded_relevance_value* is a label which indicates the degree of relevancy (see last section). If the label is 0, it means the document is irrelevant with respect to a query. If it is 1, the document is relevant. In this project, *graded_relevance_value* range from 0 to 2 in order of the degree of relevancy. To the left of the label is a *query* preceded by qid : and scores of each feature associated to a document. There can be any number of documents in the LETOR file. The example of a real LETOR file is given below:

```
#1:publicationQuery
#2:projectQuery
2 qid:Information_Retrieval 1:3.4 2:1.1 #docno = GLA_000001
```

In this example, there are 2 features: publicationQuery and projectQuery. The document GLA_000001 with respect to the query $InformationRetrieval$ is perfectly relevant with graded relevance value of 2. It has 2 feature values for publicationQuery and projectQuery which are 3.4 and 1.1 respectively.

2.2.4 Learning to Rank Model File Format

After a LETOR file is learned, we obtain a *Learned Model*. This section aims to show some learned model file formats.

```
## Coordinate Ascent
## Restart = 2
## MaxIteration = 25
## StepBase = 0.05
## StepScale = 2.0
## Tolerance = 0.001
## Regularized = false
## Slack = 0.001
1:0.64 2:4.34 3:4.3 4:2.0 5:1.2 6:9.1 7:3.3
```

The lines preceded by hash symbols are ignored. The above learned model applies *Coordinate Ascent* LETOR technique. Knowing what parameters preceded by hash symbols mean is out of the scope of the project. We only take lines not preceded by hash symbols into account. According to the above learned model, there are 7 features (1 to 7). The floating values associated to each feature indicate a score of the feature. However, some LETOR algorithms might produce duplicate features such as *1:0.64 2:4.34 3:4.3 4:2.0 5:1.2 6:9.1 7:3.3 1:1.2 5:3.2*. In this case, we sum up values corresponding to the feature, giving *1:1.84 2:4.34 3:4.3 4:2.0 5:4.4 6:9.1 7:3.3*. AdaRank is one of those algorithms.

2.2.5 Obtaining and Deploying a Learned Model

The general steps for obtaining a learned model using a learning to rank technique are the following [27, P. 4]:

> *1. Top k Retrieval: For a set of training queries, generate a sample of k documents using an initial retrieval approach.*
>
> *2. Feature Extraction: For each document in the sample, extract a vector of feature values.*
>
> *3. Learning: Learn a model by applying a learning to rank technique. Each technique deploys a different loss function to estimate the goodness of various combination of features. Documents are labelled according to available relevance assessments.*
>
> Once a learned model has been obtained from the above learning steps, it can be deployed within a search engine as follows [27, P. 4]
>
> *4. Top k Retrieval: For an unseen test query, a sample of k documents is generated in the same manner as step (1).*
>
> *5. Feature Extraction: As in step (2), a vector of feature values is extracted for each document in the sample. The set of features should be exactly the same as for (2).*
>
> *6. Learned Model Application: The final ranking of documents for the query is obtained by applying the learned model on every document in the sample, and sorting by descending predicted score.*

Figure 2.6 illustrates an architecture of a machine-learned IR system. How this architecture links to our approach will be discussed in section 4.

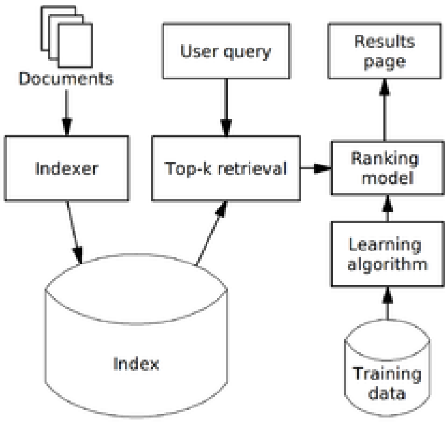


Figure 2.6: An architecture of a machine-learned IR system from `http://en.wikipedia.org/wiki/Learning_to_rank`

2.2.6 Applying a Learned Model

Once a learned model has been generated, then for each document retrieved after retrieval, the scores of each feature are multiplied by ones in the learned model and accumulated. Finally, the accumulated scores for each document are sorted in descending order and documents with the high scores are ranked before those with low scores.

2.2.7 Learning to Rank Algorithms

In this project, we use only 2 LETOR algorithms which are *AdaRank* [33, P. 60] and *Coordinate Ascent* [36]. Full detailed explanations of the algorithms are out of the scope of this project. The algorithms are chosen because of its simplicity and its effectiveness.

Coordinate Ascent

Coordinate Ascent Algorithm is an adaptation of Automatic Feature Selection Algorithm [36]. It is also a linear model [38, P. 1-4] based LETOR algorithm as same as AdaRank.

Let M_t denote the model learned after iteration t. Features are denoted by f and the weight (parameter) associated with feature f is denoted by λ_f. The candidate set of features is denoted by F. A model is represented as set of pairs of feature/weight. We assume that $SCORE(M)$ returns the goodness of model M with respect to some training data.

The algorithm shown in Figure 2.7 starts with an empty model M_t. The feature f is added to the model. AFS is used to obtain the best λ_f. The utility of feature f ($SCORE_f$) is defined to be the maximum utility obtained during training. This process is repeated for every $f \in F$. The feature with maximum utility is then added to the model and removed from F. The entire process is then repeated until the change in utility between consecutive iterations drops.

begin
  $t := 0$
  $M_t := \{\}$
  while $SCORE(M_t) - SCORE(M_{t-1}) > 0$ do
    for $f \in F$ do
      $\hat{\lambda} := argmax_{\lambda_f} SCORE(M \cup \{(f, \lambda_f)\})$ AFS is used to obtain λ_f
      $SCORE_f := SCORE(M \cup \{(f, \hat{\lambda_f})\})$
    end
    $f^* := argmax_f SCORE_f$
    $M := M \cup \{(f^*, \hat{\lambda_{f^*}})\}$
    $F := F - \{f^*\}$
    $t := t + 1$
  end
end

Figure 2.7: Pseudocode of Coordinate Ascent

AdaRank

AdaRank is a linear model [38, P. 1-4] based LETOR algorithm. It repeats the process of reweighting the training data, creating a weak ranker, and assigning a weight to the weak ranker, to minimise the loss function [33, P. 60]. In this project, a loss function is an evaluation measure such as MAP, NDCG, etc. Then AdaRank linearly combines the weak rankers as the ranking model. One advantage of AdaRank is its simplicity, and it is one of the simplest learning to rank algorithms [33, P. 60]. In Section 2.2.4, we discussed that AdaRank is one of the LETOR algorithms which produce duplicate features such as *1:0.64 2:4.34 3:4.3 4:2.0 5:1.2 6:9.1 7:3.3 1:1.2 5:3.2*. The reason behind it is that AdaRank can select the same feature multiple times [19].

2.2.8 K-fold Cross-validation

In Sections 2.2 and 2.2.5, we talked about 2 mandatory types of data: training and testing data. If our learned model is based too much on training data (pays too much attentions on training data), the predictions for unseen testing data are very poor. This is often the case, when our training dataset is very small [38]. In machine learning, we call it *over-fitting* [38, P. 28]. Determining the optimal model such that it is able to generalise well without over-fitting is very challenging. This is why we need *validation data*. A validation data is typically provided separately or removed from original dataset [19]. K-fold cross-validation splits the original dataset into K equally sized blocks or folds [38]. Each fold takes its turn as a validation data for a training data comprised of the other K - 1 folds [38]. Figure 2.8 describes K-fold Cross-validation graphically.

2.3 Tools

This section will discussed tools used in this project. They include search engine platform Terrier, learning to rank library RankLib and evaluation tool, trec_eval.

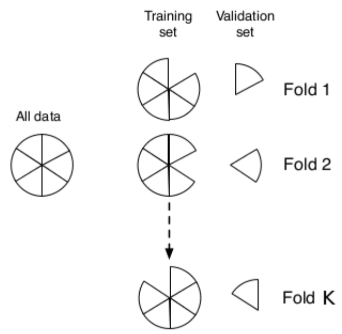


Figure 2.8: K-fold Cross-validation from Machine Learning Lecture 3 - Linear Modelling By Simon Rogers

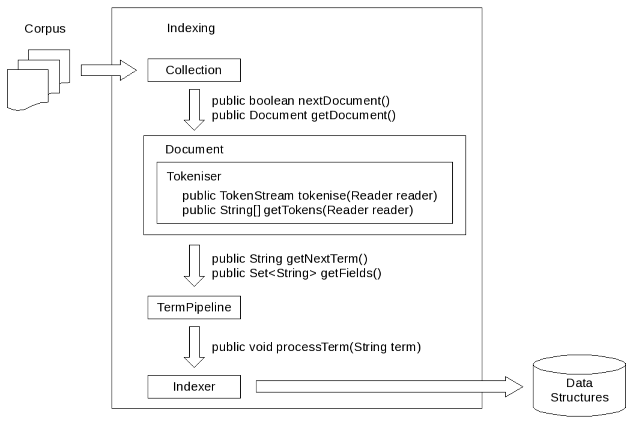


Figure 2.9: Indexing Architecture of Terrier from `http://terrier.org/docs/v3.5/basicComponents.html`

2.3.1 Terrier

Every IR system requires programs that handle indexing, retrieving, ranking, etc. To build everything from scratch, it would be impossible within the 1 year duration. However, there are a number of search engine platforms that deal with IR functionalities effectively. Terrier [14] was chosen because it is a highly flexible, efficient, and effective open source search engine. It is a comprehensive, and flexible platform for research and experimentation in text retrieval. Research can easily be carried out on standard TREC collection [15]. Using Terrier, this project can easily extend from the existing search engine as it used Terrier as a search engine platform and it is written in Java which is the same programming language used in this project.

Terrier Indexing

Figure 2.9 gives an overview of the interaction of the main components involved in the indexing process of Terrier.

1. A corpus will be represented in the form of a ***Collection*** *object. Raw text data will be represented in the form of a* ***Document*** *object. Document implementations usually are provided with an instance of a* ***Tokeniser*** *class that breaks pieces of text into single indexing tokens.*

*2. The indexer is responsible for managing the indexing process. It iterates through the documents of the collection and sends each found term through a **TermPipeline** component.*

*3. A TermPipeline can transform terms or remove terms that should not be indexed. An example for a TermPipeline chain is termpipelines=Stopwords,PorterStemmer, which removes terms from the document using the **Stopwords** object, and then applies Porter's Stemming algorithm for English to the terms.*

4. Once terms have been processed through the TermPipeline, they are aggregated and the following data structures are created by their corresponding DocumentBuilders: DirectIndex, DocumentIndex, Lexicon, and InvertedIndex.

5. For single-pass indexing, the structures are written in a different order. Inverted file postings are built in memory, and committed to 'runs' when memory is exhausted. Once the collection had been indexed, all runs are merged to form the inverted index and the lexicon.

2.3.2 RankLib

RankLib [9] is an open source library of learning to rank algorithms. It provides various retrieval metrics to facilitate evaluation. Currently 8 popular algorithms have been implemented:

- MART (Multiple Additive Regression Trees, a.k.a. Gradient boosted regression tree)
- RankNet
- RankBoost
- AdaRank
- Coordinate Ascent
- LambdaMART
- ListNet
- Random Forests

This library was chosen because it is easy to use, written in Java which can be easily combined with this system as it is also developed using Java, and it implements 8 different learning to rank algorithms which makes it possible for users to try different algorithms. However, AdaRank and Coordinate Ascent were only used in the project because other algorithms are too complex and not part of the scope of this project. RankLib provides k-fold cross validation functionally given only training dataset. This saves us time to split data for validation, training and testing as described in section 2.2.

2.3.3 trec_eval

trec_eval is the standard tool used by the TREC community [15] for evaluating a retrieval run, given the results file and a standard set of judged results. trec_eval was chosen because the data used in this project uses TREC format. It is easy to use and written in C which makes the tool efficient. To use trec_eval, 2 files are required: qrels and result file in TREC format. *qrels* is a file that stores relevant results for queries stored in a queries file [8]. It has the following format [10]:

TOPIC ITERATION DOCUMENT# RELEVANCY

where *TOPIC* is the query, *ITERATION* is the feedback iteration (almost always zero and not used), *DOCUMENT#* is the official document number that corresponds to the "docno" field in the documents (in this project, expert's id), and *RELEVANCY* is a number indicating a degree of relevancy. In this project, 3 numbers: 0, 1, and 2 indicating non-relevant, relevant and perfectly relevant respectively, are used. The result file is the file that contains the following [16]:

TOPIC ITERATION DOCUMENT# RANK SIM RUN_ID

where *RANK* is the position of the corresponding document# in a ranking, *SIM* is a float value indicating a score and *RUN_ID* is a string which gets printed out with the output.

After trec_eval is run, it will give measures of metrics discussed in Section 2.1.4. To obtain NDCG measure, trec_eval requires an additional argument (-m ndcg) when it is run.

2.4 Expert Search

In section 1.2, we discussed about the definition of Expert Search. This section will give 5 scenarios proposed by Yimam-Seid & Kobsa why expert search is needed [26, P. 387] and outline several ways to present query results.

> *1. Access to non-documented information - e.g. in an organisation where not all relevant information is documented.*
>
> *2. Specification need - the user is unable to formulate a plan to solve a problem, and resorts to seeking experts to assist them in formulating a plan.*
>
> *3. Leveraging on another's expertise (group efficiency) - e.g. finding a piece of information that a relevant expert would know/find with less effort than the seeker.*
>
> *4. Interpretation need - e.g. deriving the implications of, or understanding, a piece of information.*
>
> *5. Socialisation need - the user may prefer that the human dimension be involved, as opposed to interacting with documents and computers.*

2.4.1 Presenting Query Results

Figure 2.1 illustrates the process of search. From this figure, the process will start again if a user is not satisfied with the results. In this section, the focus is on how to convince user that a query result is good. Suppose a user who is currently studying software engineering would like to know "how to normalise database tables", first of all, he needs to interpret his information need into a query which is "database normalisation". He then types his query into his preferred search engine. After he submits the query, the search engine gives him a list of results ranked by the degree of relevancy. The question is how does he determine which result is what he is looking for?. Well, he could assume that the ranking provided by the search engine is correct. That is, the first result is what he is looking for. However, this is not always the case. He then explores each result and sees if it is the right one. But without exploring each result, could he be able to determine that which result is likely to satisfy his information need? Perhaps, there has to be some evidence to convince him by just looking at the result. The followings are evidence he could take into account [19]:

- URL

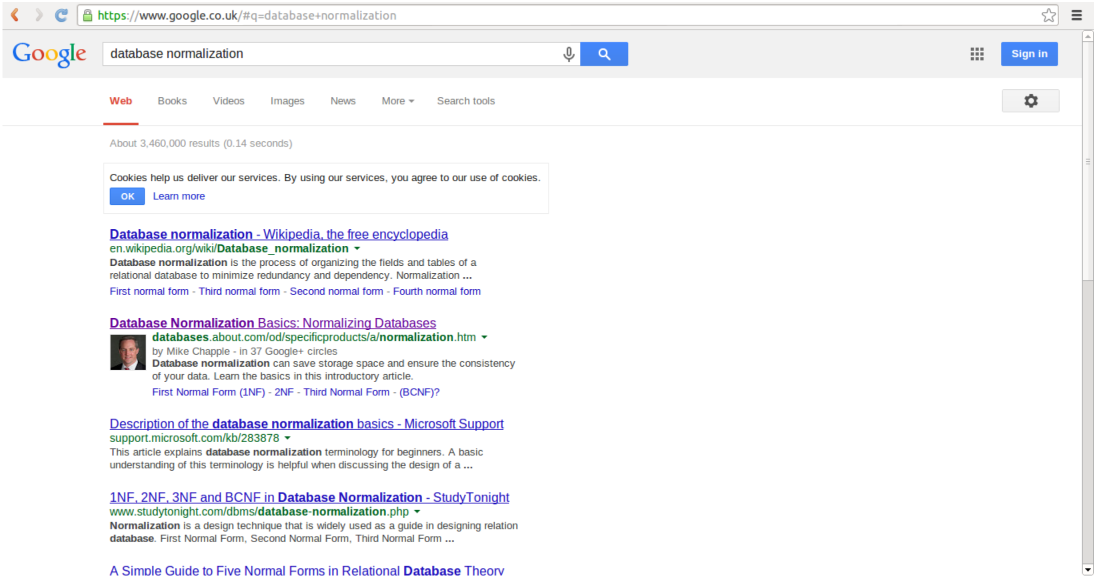


Figure 2.10: Sample Query

- Photo
- Author's Name
- keywords of the article name

If a result in response to a query have all or some of these evidence, it has more credits than ones with no evidence at all. Figure 2.10 shows the results of the query "database normalisation". It is obvious that from the top 4 results, all of the article names include the keywords a user submitted, and the third result does not have query keywords included in the URL. Among all of which, the second result has more evidence than others. It has an author's name, a photo of an author that other results do not. In this project, these evidence of credits also take into account. However, we are unable to take photo into account because the datasource (see Section 1.3) does not provide photos of experts.

2.4.2 Voting Techniques

In section 2.1.3, we very briefly talked about weighting models or retrieval models. In other words, how documents are assigned scores. In expert search, we rank experts (people), not documents directly. In this section, it aims to give an overview of a voting technique used in this project. To understand this section, readers must understand what data fusion technique is. "Data fusion techniques also known as metasearch techniques, are used to combine separate rankings of documents into a single ranking, with the aim of improving over the performance of any constituent ranking" [26, P. 388]. Within the context of this project, expert search is seen as a voting problem. The profile of each candidate is a set of documents associated to him to represent their expertise. When each document associated to a candidate's profile get retrieved by the IR system, implicit vote for that candidate occurs [26, P. 389]. Data fusion technique is then used to combine the ranking with respect to the query and the implicit vote. In expert search task, it shows that "improving the quality of the underlying document representation can significantly improve the retrieval performance of the data fusion techniques on an expert search task" [26, P. 387]. To give a simple example how voting technique works, take a look at this example

Let $R(Q)$ be the set of documents retrieved for query Q, and the set of documents belonging to the profile candidate C be denoted $profile(C)$. In expert search, we need to find a ranking of candidates, given $R(Q)$.

Rank	Docs	Scores
1	D1	5.4
2	D2	4.2
3	D3	3.9
4	D4	2.0

Table 2.4: R(Q)

Profiles	Docs
C1	D3, D4, D2
C2	D1, D2
C3	D3, D2
C4	D5, D6

Table 2.5: Profiles

Consider the simple example in Tables 2.4 and 2.5, the ranking of documents with respect to the query has retrieved documents $\{D1, D2, D3, D4\}$. Using the candidate profiles, candidate $C1$ has accumulated 3 votes, $C2$ 2 votes, $C3$ 2 votes and $C4$ no votes. If all votes are counted equally, and each document in a candidate's profile is equally weighted, a possible ranking of candidates to this query could be $\{C1, C2, C3\}$. There are various expert ranking approaches, however $expCombMNZ$ is used as it is already implemented in Terrier (see Section 2.3.1). The formula of $expCombMNZ$ is as follows:

$$candScore(C, Q) = \|R(Q) \cap profile(C)\| \sum_{d \in R(Q) \cap profile(C)} exp(score_d) \tag{2.4}$$

where $\|R(Q) \cap profile(C)\|$ is the number of documents from the profile of candidate C that are in the ranking $R(Q)$ and $score_d$ is the score of document d obtained using retrieval model of candidate C. It can be seen that this voting technique is not equally weighted as illustrated in the example before since it considers the score obtained using retrieval model of candidate C ($score_d$) as well.

2.5 Summary

So far, we have introduced some backgrounds required to understand the rest of the chapters in this project. Next chapter, we will discuss design/architecture of the current system and proposals to the new system.

Chapter 3

Current System and Proposals, Requirements and Design of New System

In this chapter, Section 3.1 is aimed to discuss the design/architecture and user interface of the current system and Section 3.2 is aimed to introduce expert search indicators of the new system and its design/architecture.

3.1 Current System

This section is intended to give some statistics and briefly discuss design of the current system. Table 3.1 gives statistics of current system.

Figure 3.1 is the home page of SICSA. It provides a set of sample queries categorised in 4 different categories on the right of the search panel. The top panel shows links to different SICSA pages. The middle panel is the search panel. It is independent of other panels. Experts (results) with respect to a query are independently shown in this panel without reloading other panels. Figure 3.2 shows the interface when experts with respect to a query, "information retrieval" are shown. Also, the system demonstrates faceted search for academics by presenting refinement options using university, location and total publication range categories. Below the refinement options is popular tags (terms) appear in expert's profiles the system retrieved. Each result in the search panel includes expert's details: university, number of publications, publication period and total number of coauthors.

Figure 3.3 illustrates a profile page of an expert. This page introduces most collaborated coauthors facet on the right of the page, a facet that shows popular terms appearing in an expert's profile and publications and related academics facet.

Design and Architecture of Current System

Figure 3.4 shows design and architecture of the current system. It makes use of publication as expertise evidence. From the figure, after a user submits a query to the system, the system sends a query to Terrier. Terrier

Number of experts	1569
Number of publications	22225
Number of all documents	24823
Number of tokens (terms)	6876832

Table 3.1: Statistics of Current System

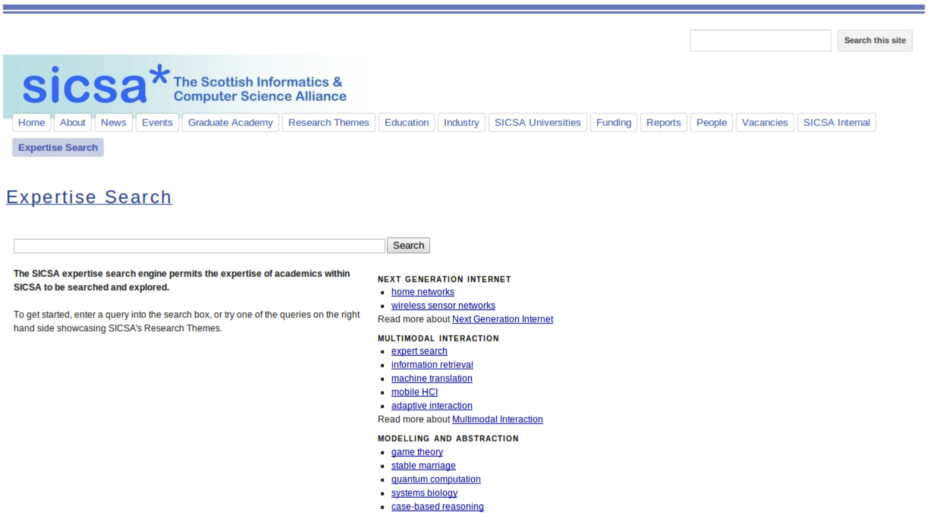


Figure 3.1: Home Page

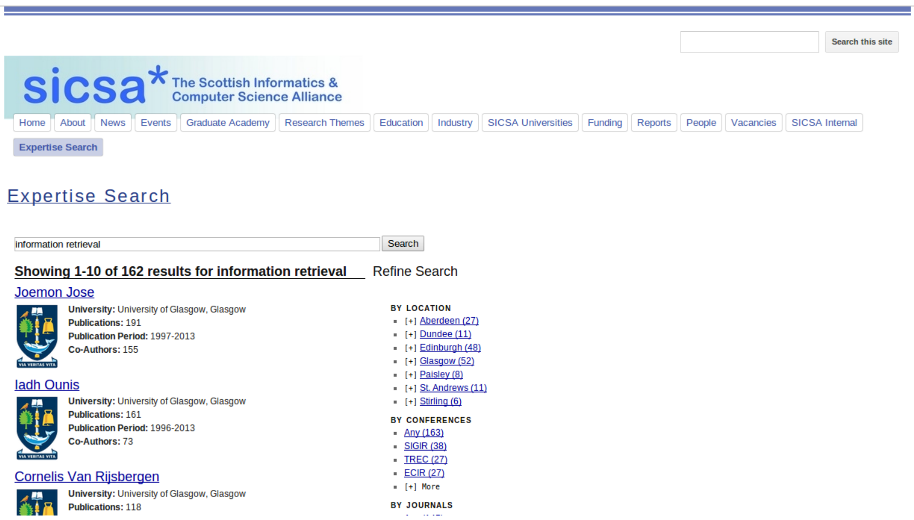


Figure 3.2: Experts In Response to “information retrieval” Query

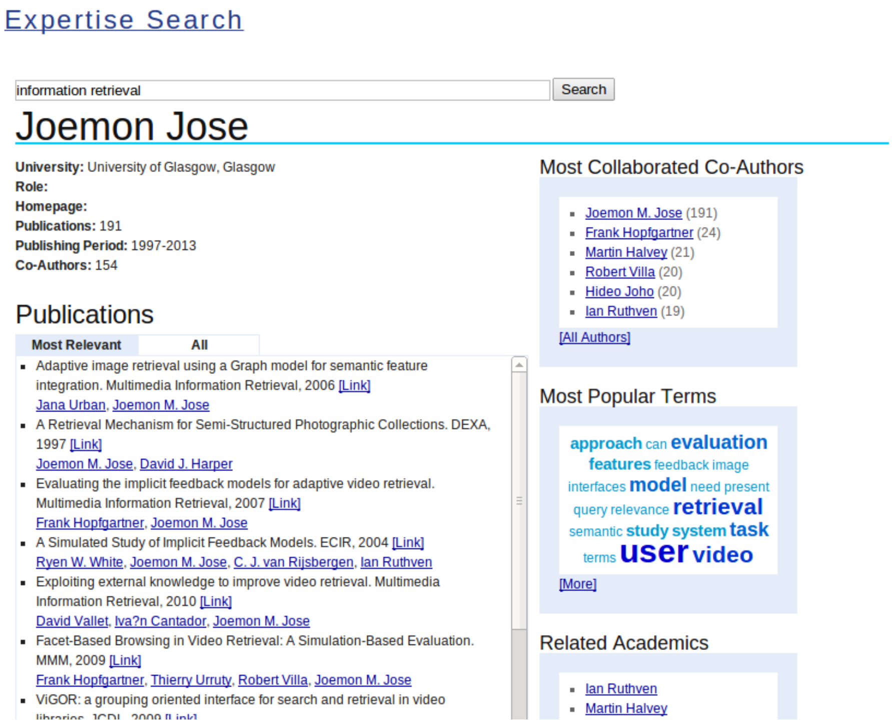


Figure 3.3: Current Expert's Profile Facet

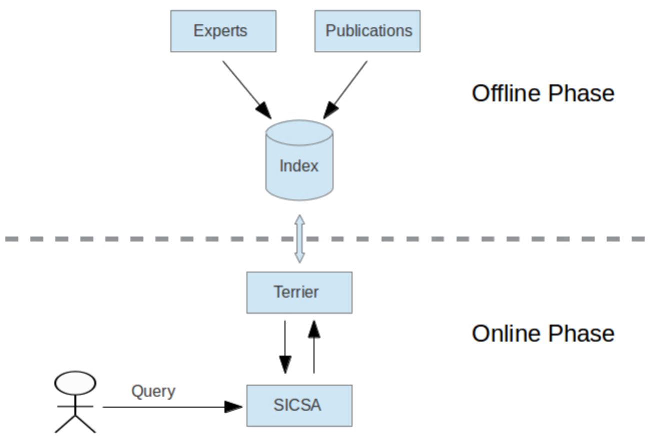


Figure 3.4: Design of Current System

then processes the query using publication as expertise evidence. During this process, $PL2$ retrieval model and voting technique $expCombMNZ$ are applied to calcuate the score of each expert. The score of each expert is computed using equation 2.4 where $score_d$ is the score of expert using $PL2$ retrieval model. Finally Terrier returns relevant experts to the user.

3.2 Expert Search Indicators of the New System

In Section 2.4.1, it was obvious that some evidence could be used to convince users how reliable a document (link) is. In this section, the evidence within the context of this project will be discussed. As discussed in Section 1.3, this project makes use of publications and funded projects as expertise evidence. Based on these evidence, what makes an expert a good expert? we propose several intuitions about a relevant expert as follows

- he has published a lot of publications.
- he has co-authored with a lot of other experts in publications.
- he has co-authored with a lot of other experts in funded projects.
- he has received a lot of funding.
- he has involved in a lot of projects.

Reasons why such intuitions are chosen are because experts are likely to be professional if they have all intuitions described above. To give an example, If an expert is looking for co-workers for a project related to information retrieval, he of course would look for co-workers who have experience (published lots of publications, involved in lots of projects, etc) and obtained lots of funding in the field of interest. It can be seen clearly that these assumptions or features in learning to rank are independent of the query a user submits. These features are query independent (Section 2.2.2). However, query dependent features (Section 2.2.1) should take into account as well. In this project, there are 2 query dependent features:

- An expert has published publications on the topic of query (Current System)
- An expert has been involved in projects on the topic of query

To sum up, a good expert should have high scores in both query dependent and independent features.

3.2.1 Requirements Specification

Due to the scope of the project it would be impossible to start without a clear vision of how the end product should function. These requirements were decided on during the early stages of the project. The reasoning behind them comes primarily from a number of sources:

- Research into a previous attempt at current SICSA system (Section 3.1).
- Research into presenting query results (Section 2.4.1).
- Research into learning to rank techniques to enhancing the performance of the retrieval system by integrating different kinds of expertise evidence (Section 2.2).

- Discussion with Dr. Craig Macdonald.

The requirements have been split into 2 sections depending on their necessity: functional requirements and non-functional requirements

Functional Requirements

Must Have

- Extracting funded projects from 2 different sources: Grant on the Web [1] and Research Councils UK [11].
- Integrating publications and funded projects as expertise evidence.
- Utilizing learning to rank techniques with an attempt to enhance the performance of the retrieval system using AdaRank and Coordinate Ascent algorithms.

Should Have

- Refinement options for users to filter results by funded projects, publications and both.
- Refinement options for users to filter funded projects and publications of each expert.

Could Have

- Applications of more LETOR algorithms such as LambdaMART, Random Forests etc.

Would Like to Have

- Ability to upload funded projects manually by members of the system.

Non-functional Requirements

- Functional 24 hours a day.
- Update data regularly.
- Able to load all evidence into main memory for efficient lookup.
- Fully functional for the purpose of the final demonstration.

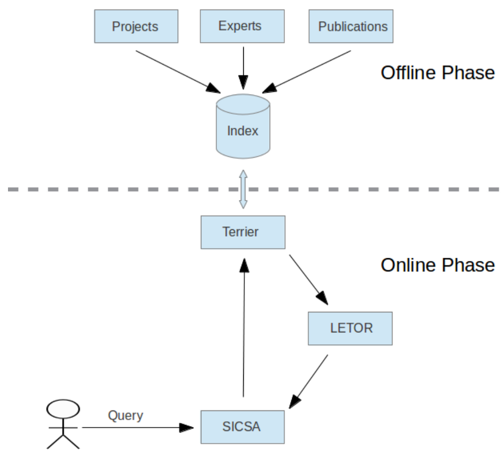


Figure 3.5: Design of Current System

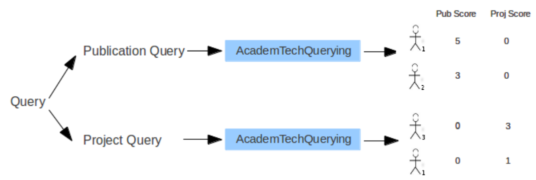


Figure 3.6: Querying

3.2.2 Design and Architecture of New System

Figure 3.5 shows design of new system. It makes use of learning to rank (LETOR) technique to integrate both publication and funded projects as expertise evidence. From the figure, in contrast to the current system, LETOR is applied to documents retrieved from Terrier. This LETOR reranks the documents to get the optimal ranking.

Retrieving Documents (Experts) with Respect to a Query

In the previous section, we gave a broad view of the new system retrieval scheme. In this section, we will be more specific on how we are going to obtain results from user query before applying them to learning to rank technique to obtain optimal ranking. Figure 3.6 shows the process of obtaining documents (experts) with respect to a query.

Given a query $\vec{q}$, it is transformed into publication query $\vec{q}_{pub}$, using only publications as expertise evidence

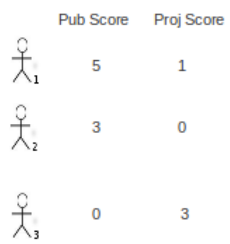


Figure 3.7: Results After Union

and project query $\vec{q}_{proj}$, using only funded projects as expertise evidence. AcademTechQuerying Class then processes both $\vec{q}_{pub}$ and $\vec{q}_{proj}$. For each query, the system applies $PL2$ retrieval model and voting technique $expCombMNZ$ (see Section 3.1). The results of both queries are then unioned, denoted $R_{cand}(\vec{q})$ as shown in Figure 3.7. The results obtained are based on query dependent features as discussed in Section 2.2.1. This process is depicted mathematically as follows:

$$\vec{q}_{pub} \rightarrow R_{cand}(\vec{q}_{pub}), \vec{q}_{proj} \rightarrow R_{cand}(\vec{q}_{proj}) \tag{3.1}$$

$$R_{cand}(\vec{q}) = R_{cand}(\vec{q}_{pub}) \cup R_{cand}(\vec{q}_{proj}) \tag{3.2}$$

where $R_{pub}(\vec{q}_{pub})$ and $R_{proj}(\vec{q}_{proj})$ are the rankings of candidate with respect to $\vec{q}_{pub}$ and $\vec{q}_{proj}$ respectively.

Applying a Learned Model and Producing Final Ranking

In the previous section, we discussed how ranking of candidates is obtained ($R_{cand}(\vec{q})$). In this section, we will propose how to apply the ranking with a learned model (see Sections 2.2.5 and 2.2.6).

Given a ranking of candidates $R_{cand}(\vec{q})$, the score of each candidate C can be computed as follows:

$$score(C, \vec{q}) = \sum_{f_{qi}} w_{f_{qi}} \cdot feat_{f_{qi}}(C) + w_{f_{q,pub}} \cdot score_{\vec{q}_{pub}}(C) + w_{f_{q,proj}} \cdot score_{\vec{q}_{proj}}(C) \tag{3.3}$$

where f_{qi} is a query independent feature discussed in Section 3.2, $w_{f_{qi}}$ is a value of query independent feature f_{qi} in a learned model, and $feat_{f_{qi}}(C)$ is a value of query independent feature of candidate C. $w_{f_{q,pub}}$ and $w_{f_{q,proj}}$ are values of query depenent features in a learned model of publications and projects respectively (see Section 3.2). $score_{\vec{q}_{pub}}(C)$ and $score_{\vec{q}_{proj}}(C)$ are publication and project scores of candidate C respectively (discussed in the previous section).

Once each candidate has its final score computed, we sort all the candidates' score in decreasing order. This is a final ranking R_{final}. Sorting can be done efficiently using $O(n \cdot logn)$ sorting algorithm such as $QuickSort$, or $MergeSort$.

$$R_{final} = sort(\hat{R}_{cand}(\vec{q})) \tag{3.4}$$

where $\hat{R}_{cand}(\vec{q})$ is the ranking of cadidates for which each candidate's final score is already computed.

Chapter 4

Implementation

In this chapter, we introduce how the system was implemented. Only important aspects of the implementation will be discussed. Section 4.1 will discuss the relationships between Expert, Project, Publication, and University. Section 4.2 will give an overview about the components involved in the querying process. Components in this section will be refered by Section 4.5 and 4.6 which will discuss components related to producing a learned model and applying a learned model to obtain a final ranking respectively. Section 4.3 will explain components about Data Extraction from both Grant on the Web and Research Councils UK datasources and show statistics about the number of funded projects extracted from each datasource. Section 4.4 will describe important files in TREC format which are indexed by Terrier and show statistics after indexing process. Section 4.7 illustrates the behaviour of the expert search system using Sequence Diagram. Finally, Section 4.8 shows user interface of the new system.

4.1 Relationships between Expert, Project, Publication, and University

Before we discuss about the implementations of the system, it is important to understand the relationships between Expert, Publication, Project, and University. Figure 4.1 shows the relationship between Candidate (Expert), Project, Publication and University classes. Each component in the figure shows only important attributes and methods. The Candidate class represents an expert who lectures at a university and has a set of publications and projects. Note that experts do not have to involve in any project or publish any publication but they must be in of one of the universities in SICSA. A list of members of SICSA can be found in Chapter 1.1.

4.2 Querying

This section is devoted to introduce classes used to handle querying process. Figure 4.2 shows class diagrams of *TableFactory*, and *Search*. These two components can also be viewed as the core part of the system. The responsibility of each class is given below:

- *Search* extends Java Servlet which handles search request submitted by a user.
- *TableFactory* loads Experts, Publications, and Projects stored in a file and send them to Search Servlet.

Search contains 4 important methods as follows:

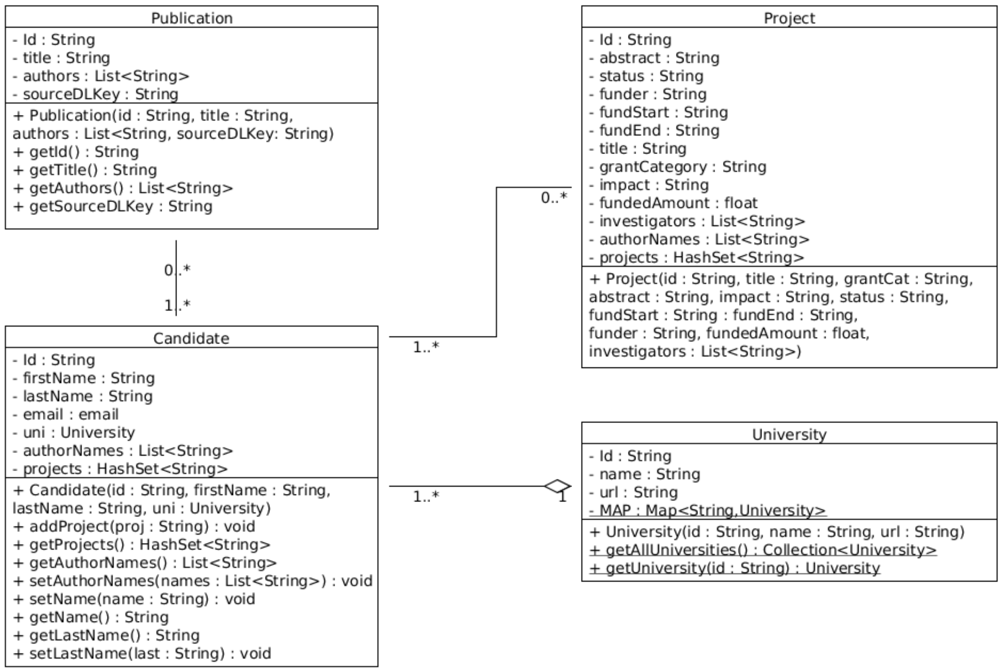


Figure 4.1: Class Diagram

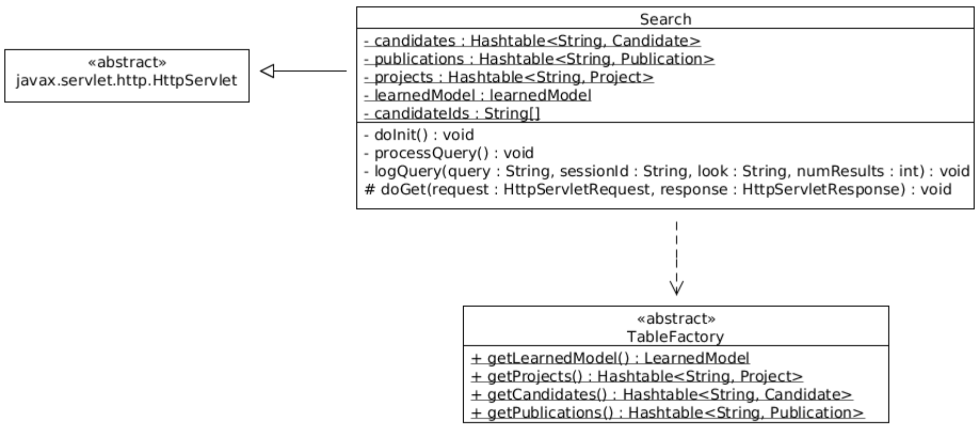


Figure 4.2: TableFactory and Search Servlet Class Diagrams

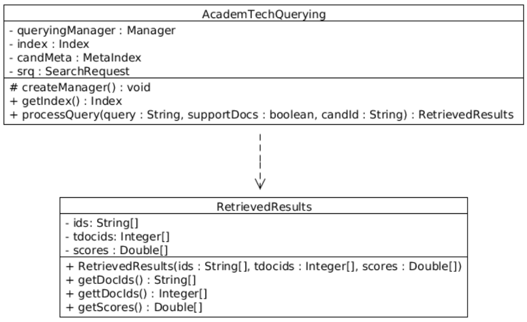


Figure 4.3: AcademTechQuerying Class and RetrievedResults Class

- *doInit()* loads AcademTechQuerying component for querying.
- *processQuery()* processes a query obtained by a user. More details are given in section 4.6.
- *logQuery()* saves a query to a log file.
- *doGet()* handles a GET request when user submits a query. It also generates HTML page of the query results.

As stated in section 3.2, the features extracted from funded projects and publications of each expert will be used in Learning to Rank to improve the retrieval performance of the system. But before we get to Learning to Rank, first of all, we need to understand AcademTechQuerying Class and RetrievedResults Class which are components used in retrieving results with respect to a query.

Figure 4.3 shows class diagrams of AcademTechQuerying and RetrievedResults. As stated in section 2.3.1, the search engine platform used in this project is Terrier. It handles indexing and retrieval processes discussed in section 2.1.1. The AcademTechQuerying Class makes use of 2 Terrier components as follows:

- *queryingManager* of type Manager, responsible for handling/co-ordinating the main high-level operations of a query.
- *srq* of type SearchRequest, responsible for retrieving search result from Manager.

The most important method of AcademTechQuerying is processQuery(). It returns RetrievedResults component. This method takes a query string, supportDocs and candId as arguments. The first argument tells the system that a query is used to retrieve the result, the second and third arguments are used in case the system wants documents associated to a candidate with respect to a query.

The RetrievedResults Class has 3 attributes as follows:

- *ids*, an array of String, which is the ids of the documents(in this case, ids of experts) retrieved by the system.
- *tdocids*, an array of Integer, which is the ids used by Terrier
- *scores*, an array of Double, which is the scores of each document retrieved by the system.

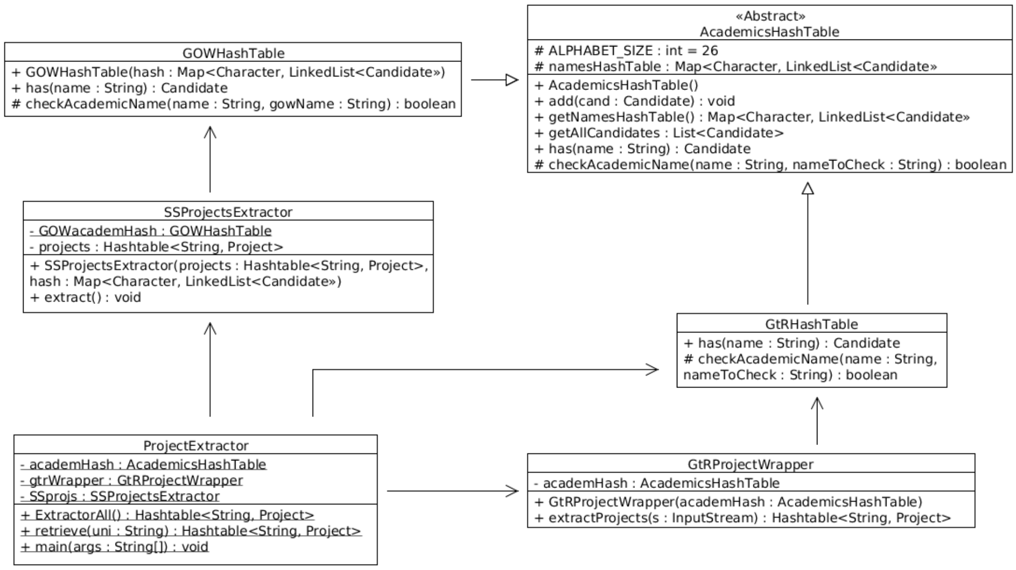


Figure 4.4: Projects Extraction Class Diagrams

4.3 Data Extraction

In section 1.3, it was stated that funded projects are obtained from Grant on the Web [1] and Research Councils UK [11]. This section shows classes used to extract funded projects from each source and gives statistics regarding the number of funded projects from each source. Figure 4.4 shows class diagram related to projects extraction task.

- **AcademicsHashTable** is an abstract class which makes use of HashMap data structure, $Map < Character, LinkedList < Candidate >>$, for efficient look up when matching candidate to our known candidates. It is keyed by the first character of candidate's name. There are 2 abstract methods in this class: has() and checkAcademicName() methods. Both of them are used together to check if the candidate is matched to our known candidates.
- **GtRHashTable** extends from the abstract class AcademicsHashTable. The purpose of this class is the same as AcademicsHashTable Class but implements has() and checkAcademicName() methods which are applicable in Research Councils UK [11] data source.
- **GOWHashTable** extends from the abstract class AcademicsHashTable. Its purpose is similar to AcademicsHashTable Class but has different implementations of has() and checkAcademicName() methods to GOWHashTable's which are applicable in Grant on the Web [1] spreadsheet data source.
- **SSProjectsExtractor** is a class that makes use of GOWHashTable Class to extract funded projects from Grant on the Web [1] spreadsheet.
- **GtRProjectWrapper** is a class used GOWHashTable Class to extract funded projects from Research Councils UK [11].
- **ProjectExtractor** is a class that makes use of SSProjectsExtractor and GtRProjectWrapper Classes to extract funded projects from both sources.

Since each source records expert's name in different formats. For example, Prof. Joemon Jose may be recorded Jose JM in one source and Jose J in another, Polymorphism [7] (in this case, different implementations

Source	Number of Funded Projects
Grant on the Web	32
Research Councils UK	337
	369 total

Table 4.1: The number of funded projects extracted from each source

of has() and checkAcademicName() methods between GtRHashTable and GOWHashTable) is required. However, pattern matching between expert's names in Grant on the Web [1] spreadsheet is still impossible as this source records in the following format: [lastname], [role] [the first and/or second letter of name] such as Rogers, Dr. Simon. The problem occurs with experts, Dr. Craig Macdonald lecturing at University of Glasgow and Prof. Catriona Macdonald lecturing at Glasgow Caledonian University as the source records Macdonald, Dr C for the former and Macdonald, Professor C for the latter. Although, role might be used to distinguish between them but for some experts the role is not known and there might be the situation that the role is also the same. Therefore, university is used as part of the matching as well.

Figure 4.1 shows the number of funded projects extracted from each source. There are 1569 known candidates in the system.

4.4 Indexing

Before indexing process begins, the system has to generate data in TREC format and an association file between TREC files. **FileGenerator** class handles this. It includes only 1 method main(). The list below is files generated by the class:

- ContactsList.trec.gz
- Homepages.trec.gz
- Publications.trec.gz
- Projects.trec.gz
- association.txt

The implementation of FileGenerator already handled ContactsList.trec.gz, Homepages.trec.gz, Publications.trec.gz, and association.txt file generations. However, the new system altered the implementation of FileGenerator class to produce Projects.trec.gz and change the content in association.txt.

ContactsList.trec.gz
This file compressed in gz format stores a list of experts in TREC format. Each expert's data is recorded in the following format:

```
<DOC>
<DOCNO>expert's id</DOCNO>
<name>expert's name</name>
<snippets>description of expert</snippets>
<keyTerms></keyTerms>
<uni>expert's university</uni>
<uniId>university's id</uniId>
<location>location of the university</location>
```

```
<role>role of the expert</role>
<pubGroup>group of publication</pubGroup>
<conference>conference's name involved</conference>
<journal>type of journal</journal>
<coauthorGroup>group of coauthor</coauthorGroup>
</DOC>
```

Homepages.trec.gz
This file compressed in gz format stores a list of home pages in TREC format. Each home page associated to one expert is recorded in the following format:

```
 <DOC>
        <DOCNO>homepage's id</DOCNO>
        <candidate>expert's id</candidate>
        <content>content of the home page</content>
</DOC>
```

Publications.trec.gz
This file compressed in gz format stores a list of publications in TREC format. Each publication is recorded in the following format:

```
 <DOC>
        <DOCNO>publication's id</DOCNO>
        <TYPE>type of evidence</TYPE>
        <type>inproceedings</type>
        <pubtitle>title</pubtitle>
        <author>author's name 1</author>
        <author>author's name 2</author>
        <conference>conference's name involved</conference>
        <year>year</year>
        <snippets>description of the publication</snippets>
        <keyTerms></keyTerms>
        <citationContexts></citationContexts>
        <uni>university's name</uni>
        <uniId>university's id</uniId>
        <location>location of the university</location>
        <role>role of the project</role>
        <pubGroup>group of the publication</pubGroup>
        <journal>type of journal</journal>
        <coauthorGroup>group of coauthor</coauthorGroup>
</DOC>
```

There are 2 <*type*> tags whose values are always set to *pub* and *inproceedings*. The idea is similar one's in funded project which is for filter modes in Terrier.

Projects.trec.gz
This file compressed in gz format stores a list of funded projects in TREC format. Each funded project is recorded in the following format:

Number of experts	1569
Number of all documents	25311
Number of tokens (terms)	6976895

Table 4.2: Statistics of New System After Indexing Process

```
<DOC>
<DOCNO>funded project's id</DOCNO>
<type>type of evidence</type>
<title>title of the project</title>
<grantCategory>type of grant</grantCategory>
<start>start date</start>
<end>end date</end>
<abstract>abstract of the project</abstract>
<impact>impact of the project</impact>
<people>
<person>expert's name</person>
</people>
</DOC>
```

where *<type>* tag's value is always *proj* which tells Terrier that this document is of type *proj*. This tag is very useful when the system transforms the query to retrieve documents using only funded projects as expertise evidence (see section 3.2.2).

association.txt
This file tells Terrier about the associations between experts, publications, and funded projects. It is stored in the following format:

```
expert_id expert_homepage pub_id1 pub_id2 ... pub_idn proj_id1
proj_id2 ... proj_idn expert_homepage_id
```

where one line in the file corresponds to only one expert.

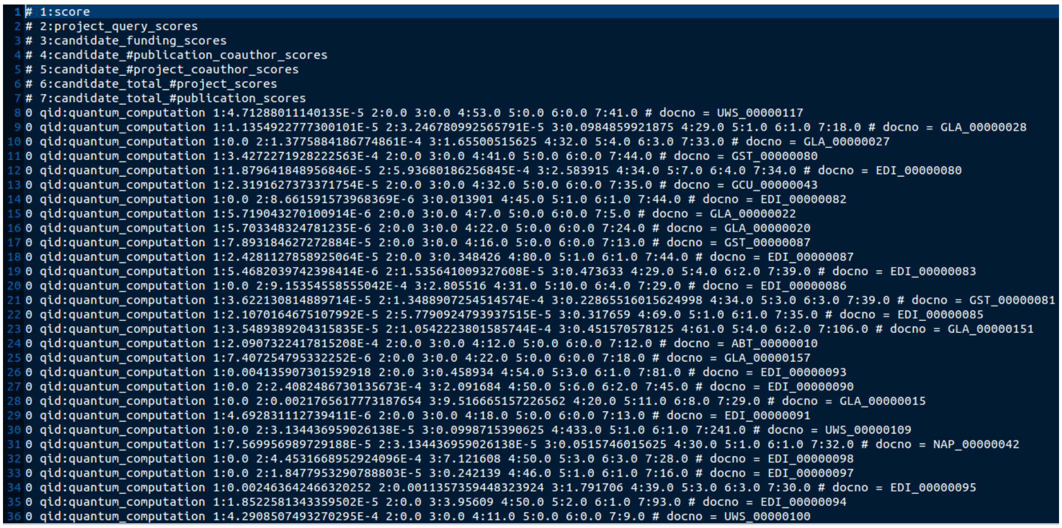
Once all the files have been generated, Terrier indexes those files. Table 4.2 shows statistics of new system after indexing.

4.5 Producing a Learned Model

Section 2.2.5 described general steps in producing a learned model. This section aims to discuss classes involved in producing a learned model. As section 2.2.5 suggested, the first step in producing a learned model is to generate a set of training queries. Then all features described in Section 3.2 for each document (expert) with respect to each training query are extracted and saved in a file. This file in learning to rank is called LETOR file (see section 2.2.3). Figure 4.5 is a LETOR file. There are 7 features as described section 3.2.

Figure 4.6 shows a class diagram that is used to produce a learned model. There are 5 important methods as follows

- *loadQueries()* is a method that loads all training queries from a file.



```
# 1:score
# 2:project_query_scores
# 3:candidate_funding_scores
# 4:candidate_#publication_coauthor_scores
# 5:candidate_#project_coauthor_scores
# 6:candidate_total_#project_scores
# 7:candidate_total_#publication_scores
0 qid:quantum_computation 1:4.71288011140135E-5 2:0.0 3:0.0 4:53.0 5:0.0 6:0.0 7:41.0 # docno = UWS_00000117
0 qid:quantum_computation 1:1.1354922777300101E-5 2:3.246780992565791E-5 3:0.0984859921875 4:29.0 5:1.0 6:1.0 7:18.0 # docno = GLA_00000028
0 qid:quantum_computation 1:0.0 2:1.3775884186774861E-4 3:1.65500515625 4:32.0 5:4.0 6:3.0 7:33.0 # docno = GLA_00000027
0 qid:quantum_computation 1:3.4272271928222563E-4 2:0.0 3:0.0 4:41.0 5:0.0 6:0.0 7:44.0 # docno = GST_00000080
0 qid:quantum_computation 1:1.879641848956846E-5 2:5.93680186256845E-4 3:2.583915 4:34.0 5:7.0 6:4.0 7:34.0 # docno = EDI_00000080
0 qid:quantum_computation 1:2.3191627373371754E-5 2:0.0 3:0.0 4:32.0 5:0.0 6:0.0 7:35.0 # docno = GCU_00000043
0 qid:quantum_computation 1:0.0 2:8.661591573968369E-6 3:0.013901 4:45.0 5:1.0 6:1.0 7:44.0 # docno = EDI_00000082
0 qid:quantum_computation 1:5.719043270100914E-6 2:0.0 3:0.0 4:7.0 5:0.0 6:0.0 7:5.0 # docno = GLA_00000022
0 qid:quantum_computation 1:5.703348324781235E-6 2:0.0 3:0.0 4:22.0 5:0.0 6:0.0 7:24.0 # docno = GLA_00000020
0 qid:quantum_computation 1:7.893184627272884E-5 2:0.0 3:0.0 4:16.0 5:0.0 6:0.0 7:13.0 # docno = GST_00000087
0 qid:quantum_computation 1:2.4281127858925064E-5 2:0.0 3:0.348426 4:80.0 5:1.0 6:1.0 7:44.0 # docno = EDI_00000087
0 qid:quantum_computation 1:5.4682039742398414E-6 2:1.535641009327608E-5 3:0.473633 4:29.0 5:4.0 6:2.0 7:39.0 # docno = EDI_00000083
0 qid:quantum_computation 1:0.0 2:9.15354558555042E-4 3:2.805516 4:31.0 5:10.0 6:4.0 7:29.0 # docno = EDI_00000086
0 qid:quantum_computation 1:3.622130814889714E-5 2:1.3488907254514574E-4 3:0.22865516015624998 4:34.0 5:3.0 6:3.0 7:39.0 # docno = GST_00000081
0 qid:quantum_computation 1:2.1070164675107992E-5 2:5.7790924793937515E-5 3:0.317659 4:69.0 5:1.0 6:1.0 7:35.0 # docno = EDI_00000085
0 qid:quantum_computation 1:3.5489389204315835E-5 2:1.0542223801585744E-4 3:0.451570578125 4:61.0 5:4.0 6:2.0 7:106.0 # docno = GLA_00000151
0 qid:quantum_computation 1:2.0907322417815208E-4 2:0.0 3:0.0 4:12.0 5:0.0 6:0.0 7:12.0 # docno = ABT_00000010
0 qid:quantum_computation 1:7.407254795332252E-6 2:0.0 3:0.0 4:22.0 5:0.0 6:0.0 7:18.0 # docno = GLA_00000157
0 qid:quantum_computation 1:0.004135907301592918 2:0.0 3:0.458934 4:54.0 5:3.0 6:1.0 7:81.0 # docno = EDI_00000093
0 qid:quantum_computation 1:0.0 2:2.4082486730135673E-4 3:2.091684 4:50.0 5:6.0 6:2.0 7:45.0 # docno = EDI_00000090
0 qid:quantum_computation 1:0.0 2:0.0021765617773187654 3:9.516665157226562 4:20.0 5:11.0 6:8.0 7:29.0 # docno = GLA_00000015
0 qid:quantum_computation 1:4.692831112739411E-6 2:0.0 3:0.0 4:18.0 5:0.0 6:0.0 7:13.0 # docno = EDI_00000091
0 qid:quantum_computation 1:0.0 2:3.134436959026138E-5 3:0.0998715390625 4:433.0 5:1.0 6:1.0 7:241.0 # docno = UWS_00000109
0 qid:quantum_computation 1:7.569956989729188E-5 2:3.134436959026138E-5 3:0.0515746015625 4:30.0 5:1.0 6:1.0 7:32.0 # docno = NAP_00000042
0 qid:quantum_computation 1:0.0 2:4.4531668952924096E-4 3:7.121608 4:50.0 5:3.0 6:3.0 7:28.0 # docno = EDI_00000098
0 qid:quantum_computation 1:0.0 2:1.8477953290788803E-5 3:0.242139 4:46.0 5:1.0 6:1.0 7:16.0 # docno = EDI_00000097
0 qid:quantum_computation 1:0.002463642466320252 2:0.0011357359448323924 3:1.791706 4:39.0 5:3.0 6:3.0 7:30.0 # docno = EDI_00000095
0 qid:quantum_computation 1:1.8522581343359502E-5 2:0.0 3:3.95609 4:50.0 5:2.0 6:1.0 7:93.0 # docno = EDI_00000094
0 qid:quantum_computation 1:4.2908507493270295E-4 2:0.0 3:0.0 4:11.0 5:0.0 6:0.0 7:9.0 # docno = UWS_00000100
```

Figure 4.5: LETOR file of Training Queries

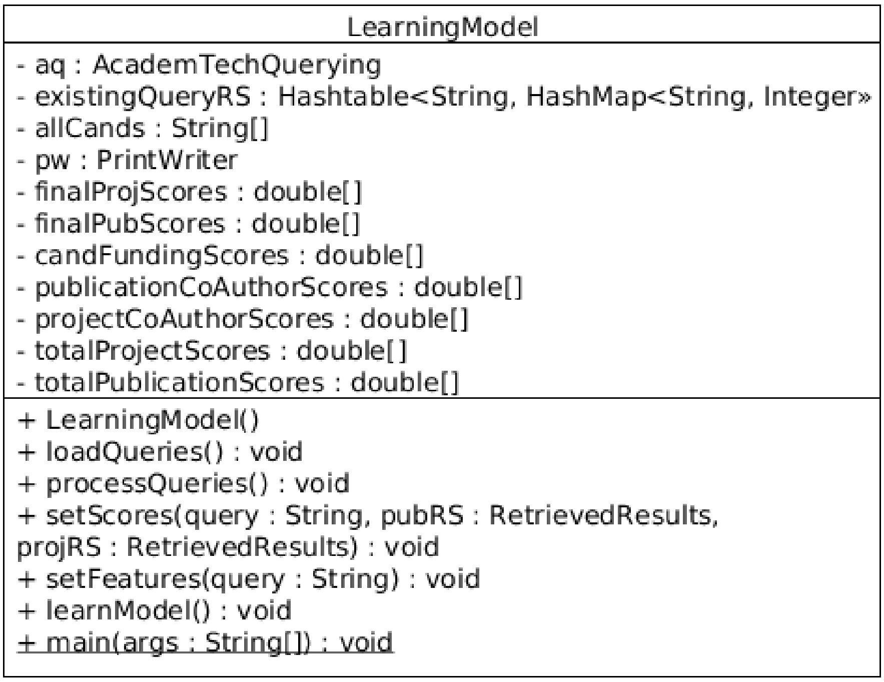


Figure 4.6: Class Diagram of LearningModel

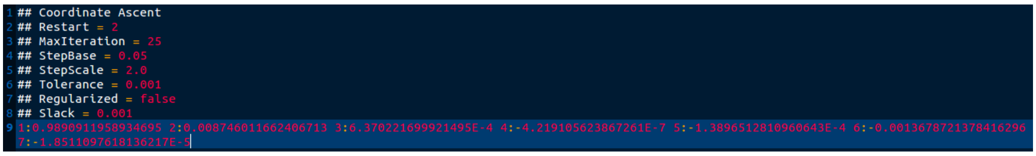


Figure 4.7: Learned Model Using Coordinate Ascent Algorithm

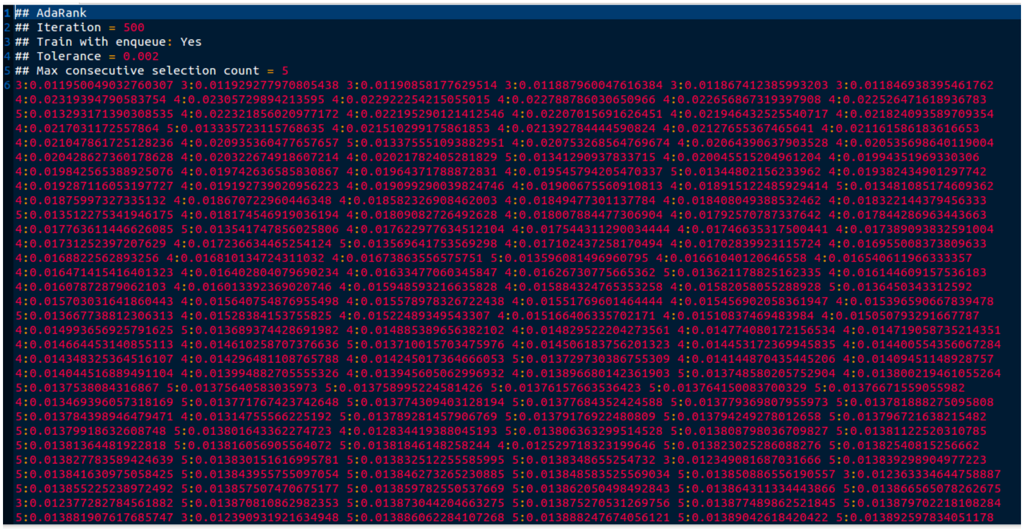


Figure 4.8: Learned Model Using AdaRank Algorithm

- *processQuery()* is a method that retrieve documents (experts) with respect to publication query and project query.
- *setScores()* is a method that performs a union between results with respect to publication query and project query as discussed in Section 3.2.2
- *setFeatures()* is a method that extracts features for each document (expert) and write into a LETOR file.
- *learnModel()* is a method that produces a learned model using RankLib.

As stated in Section 2.3.2, learning to rank algorithms used in this project are AdaRank and Coordinate Ascent. The performance of each of them will be discussed in Evaluation Section. Figures 4.7 and 4.8 show learned models using *Coordinate Ascent* and *AdaRank* Algorithms. Section 4.6 will explain a component used to apply a learned model.

4.6 Applying a Learned Model and Producing Final Ranking

In Chapter 3.2.2, we explained the process of applying a learned model and generating a final ranking. This section aims to describe components that handle these tasks. The class *CandidateScores* has a method called, getTotalScore() taking a learnedModel component as a parameter. This method applies a learned model as described in section 2.2.6 and returns a score associated with an expert. Figure 4.9 shows class diagrams of *Candidatescores* and *LearnedModel*.

The *CandidateScores* is used for computing scores of candidates based on a learned model. The *CandidateSorting* sorts scores of candidates obtained after applying a learned model in descending score. The sorting algorithm applied in this class is **merge sort** [5], giving time complexity of $O(n \cdot log\, n)$ where n is the number

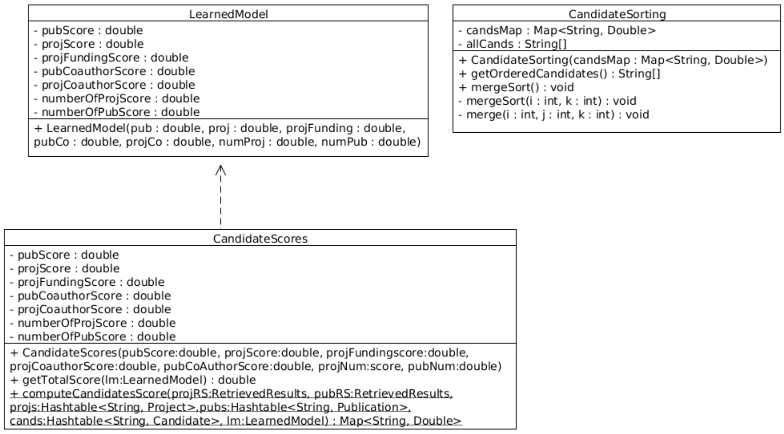


Figure 4.9: Class Diagrams of CandidateScores and CandidateSorting

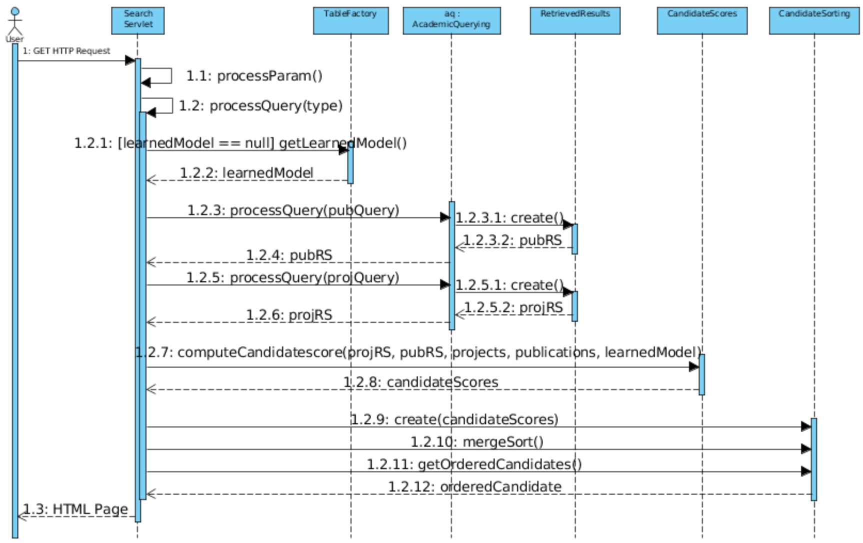


Figure 4.10: Sequence Diagram of Querying

of candidates (experts) to be sorted. The time taked to do merge sort is roughly 0.75 milliseconds on average. Figure 4.9 shows class diagrams of *CandidateScores*, *LearnedModel* and *CandidateSorting*. Important attributes and methods are only shown.

4.7 Sequence Diagram of Querying Process

This section shows the sequence diagram of how the searching works based on discussion in Chapter 3.2.2. It illustrates which components are involved in the process of querying. Figure 4.10 illustrates a sequence diagram of querying. This sequence diagram only shows important processes to retrieve relevant results. Given a query submitted by a user, Search servlet processes parameters sent via GET request by calling processParam() method. This is followed by invoking processQuery() method which takes one parameter, “type”. This parameter is used for filter mode. The method processQuery() performs tasks discussed in Chapter 3.2.2.

First of all, AcademTechQuerying Class is used to produce the ranking of candidates using publications as expertise evidence, $R_{cand}(\vec{q}_{pub})$ and the ranking using $R_{cand}(\vec{q}_{proj})$. Following this, the CandidateScores Class computes the score for each candidate based on a learned model ($\hat{R}_{cand}(\vec{q})$). Finally, the ranking $\hat{R}_{cand}(\vec{q})$

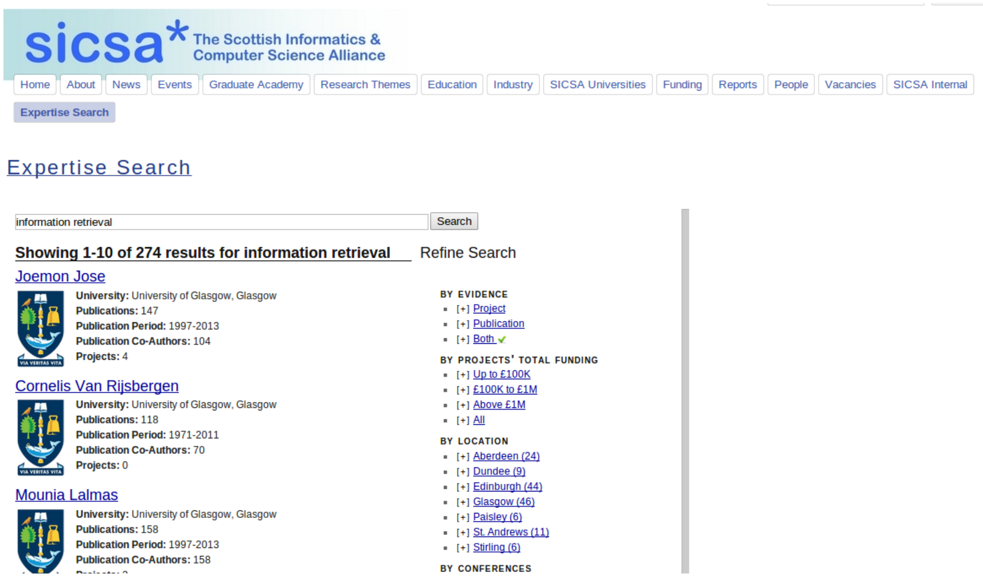


Figure 4.11: New Facets : Experts In Response to "information retrieval" Query

produced by CandidateScores component is sorted by CandidateSorting Class to produce the final ranking R_{final}.

4.8 User Interface of the New System

Figure 4.11 shows a new search facet in response to a query "information retrieval". The number of projects experts have been involved in is displayed. In addition to the current system filter modes, filter options by using projects, publications and both expertise evidence, and by projects' total funding are added on the right panel of the page.

Figure 4.12 shows a new facet of profile page. There are 2 tabs for publications and funded projects and a drop down box for selecting result filter modes. The number of projects, project collaborators, project period and total funding are shown.

Figure 4.13 shows a facet when a project tab is selected. The right hand side of the page shows the most project collaborators and other filter modes: filter projects by amount of funding and by the status of projects.

Figure 4.14 shows a facet when user selects a project link. This shows descriptions of a selected project. The descriptions include:

- Funder
- Title
- Funded Value
- Funded Period
- Status
- Grant Category
- People involved in the project

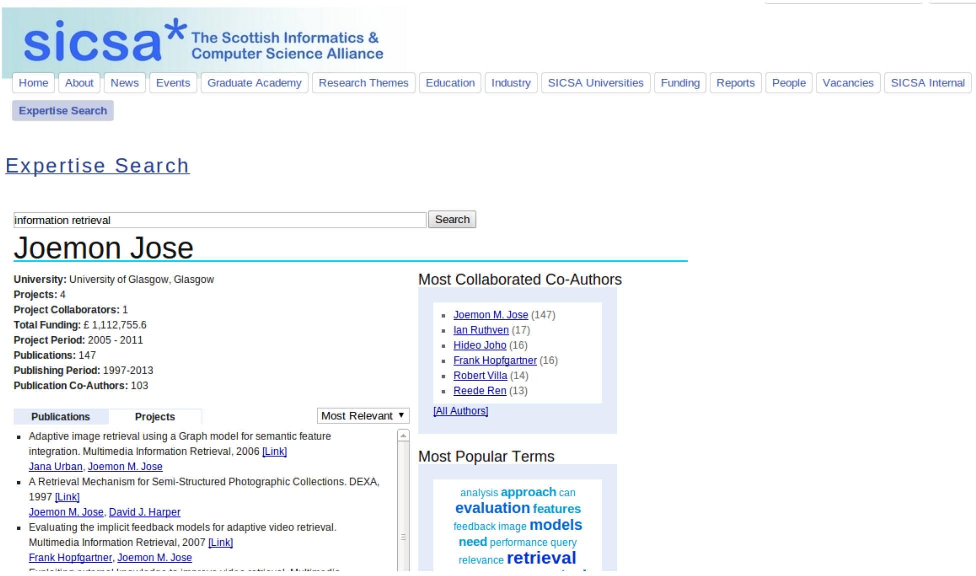


Figure 4.12: New Expert's Profile Facet

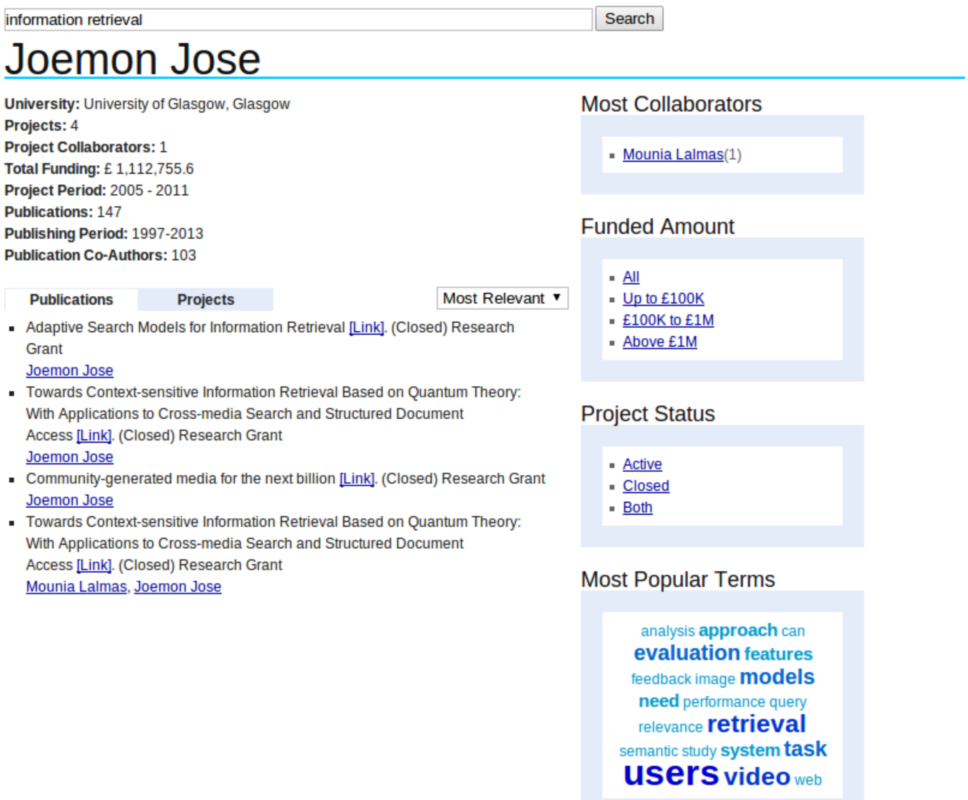


Figure 4.13: Project Tab

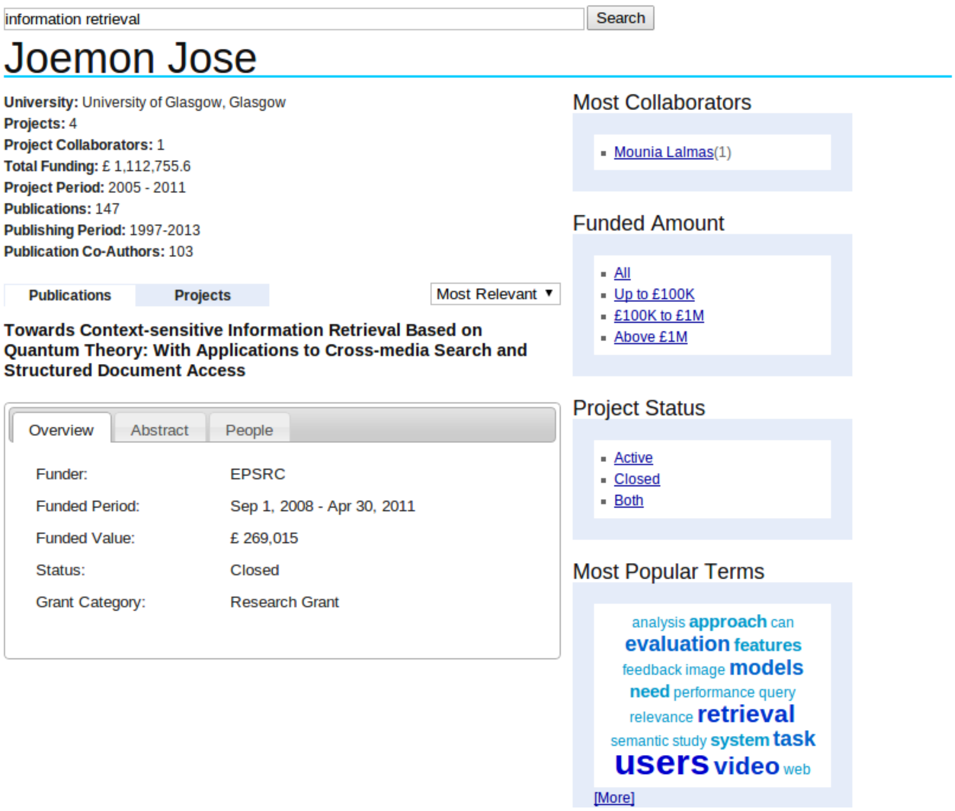


Figure 4.14: Project Page

- Abstract
- Impact

Some projects descriptions are not shown if they are missing.

Chapter 5

Evaluation

This chapter is aimed to demonstrate that applying LETOR technique improves the effectiveness of the expert search system. It separates into 2 experiments. The first experiment does not make use of K-fold Cross-validation technique but the second experiment does. We use LETOR technique proposed in Chapter 2.2. Basically, there are 2 necessary datasets: training and testing datasets and 1 optional dataset: validation dataset. In IR, we refer them as training, testing and validation queries respectively. Some of the queries were provided by Dr. Craig Macdonald and the others were made up by myself. We made an assumption that the current system (using only publication as expertise evidence) is effective. This means that relevance judgements (the experts retrieved in response to a query) can be obtained manually from the current system. To make relevance judgements, top 20 experts retrieved in response to a query are examined if they are truly an expert by visiting their official home page and with the help of Dr. Craig Macdonald. Throughout this chapter, the queries annotated by * denote queries whose relevant experts are restricted to those experts only from the University of Glasgow. It is difficult to judge experts not in the University of Glasgow because we do not know them. However, in practice, this will incur dataset quality [19].

Before we discuss evaluation, we should have some goals set up. This is necessary because the evaluation results can be compared against the goals in order to conclude whether all goals are achieved or not. As stated in Chapter 1.3, our goal is to integrate new kind of expertise evidence, funded projects, with existing expertise evidence, publications, to enhance the effectiveness of the expert search system. At the end of this chapter, we have to be able to answer the following question:

> *Does integrating funded projects with publications using learning to rank technique (LETOR) help improve the performance of the retrieval system?*

5.1 Evaluation Without K-fold Cross-validation

This section is aimed to demonstrate that applying LETOR technique improves the performance of the retrieval system. K-fold Cross-validation technique is not applied in this section.

5.1.1 Experimental Setting

The following is a list of steps in running an evaluation:

#	Query	#	Query
1	*language modelling	18	game theory
2	*manets	19	stable marriage
3	*match	20	quantum computation
4	*multimodal	21	constraint modelling
5	*music as navigation cues	22	home networks
6	networking security	23	wireless sensor networks
7	*neural network	24	distributed systems
8	*older adults use of computers	25	operating system
9	*parallel logic programming	26	*terrier
10	*query expansion	27	*text searching
11	*road traffic accident statistics	28	*trec collection class
12	*shoogle	29	*usability
13	*skill-based behavior	30	*utf support terrier
14	*sound in multimedia human-computer interfaces	31	*visual impairment
15	statistical inference	32	*wafer fab cost
16	*suffix tree	33	database
17	mobile hci	34	programming languages

Table 5.1: Training Queries

#	Query	#	Query
1	*empirical methods	18	*3d human body segmentation
2	*eurosys 2008	19	*anchor text
3	*facial reconstruction	20	*clustering algorithms
4	*force feedback	21	*different matching coefficients
5	functional programming	22	*discrete event simulation
6	*glasgow haskell compiler	23	*discrete mathematics
7	*grid computing	24	*distributed predator prey
8	artificial intelligence	25	*divergence from randomness
9	computer graphics	26	*ecir 2008
10	robotics	27	*group response systems
11	data mining	28	*handwriting pin
12	computer vision	29	*haptics
13	information security	30	*haptic visualisation
14	*3d audio	31	*hci issues in mobile devices
15	expert search	32	*human error health care
16	information retrieval	33	*information extraction
17	machine translation		

Table 5.2: Testing Queries

1. generate a LETOR file for testing queries.
2. from the LETOR file obtained from (1), generate a qrels file as discussed in section 2.3.3.
3. generate 2 learned models (see section 4.5) from training queries using AdaRank, and Coordinate Ascent algorithms.
4. apply each learned model to all testing queries.
5. for each test in (4), generate a scores file of each test using RankLib and generate a result file in TREC format (see section 2.3.3).
6. evaluate the result file in TREC format with the qrels file for each test using trec_eval (see section 2.3.3).

To evaluate without using LETOR (using only publications as expertise evidence), we can set other features apart from publication query feature to 0 and perform step (5) and (6).

Training Queries

34 queries shown in table 5.1 are used for training.

Testing Queries

33 queries are used for testing as shown in table 5.2.

Algorithm	MAP	NDCG	MRR	Number of Relevant Experts
AdaRank	0.0153	0.1872	0.0116	106
Coordinate Ascent	0.5264	0.6618	0.6332	106
Without LETOR	0.5499	0.6871	0.6484	106

Table 5.3: Evaluation Results

Testing Queries	AdaRank		Coordinate Ascent		Without LETOR	
	MAP	MRR	MAP	MRR	MAP	MRR
*3d audio	0.0053	0.0053	1.0000	1.0000	1.0000	1.0000
*3d human body segmentation	0.0127	0.0068	0.1452	0.2000	0.1824	0.2000
*anchor text	0.0263	0.0088	0.7894	1.0000	0.7894	1.0000
artificial intelligence	0.0344	0.0233	0.5944	1.0000	0.5972	1.0000
*clustering algorithms	0.0166	0.0071	0.3048	0.2000	0.3048	0.2000
computer graphics	0.0047	0.0047	0.5000	0.5000	0.5000	0.5000
computer vision	0.0142	0.0130	0.4611	0.5000	0.4611	0.5000
data mining	0.0181	0.0118	0.9167	1.0000	0.9167	1.0000
*different matching coefficients	0.0091	0.0063	0.5833	0.5000	0.5833	0.5000
*discrete event simulation	0.0078	0.0078	0.3333	0.3333	0.3333	0.3333
*discrete mathematics	0.0164	0.0115	0.0306	0.0400	0.1394	0.1250
*distributed predator prey	0.0058	0.0058	0.0037	0.0037	0.1111	0.1111
*divergence from randomness	0.0124	0.0058	0.9167	1.0000	0.9167	1.0000
*ecir 2008	0.0152	0.0070	0.8125	1.0000	0.8125	1.0000
*empirical methods	0.0189	0.0072	0.2092	0.1667	0.2154	0.1667
*eurosys 2008	0.0055	0.0055	1.0000	1.0000	1.0000	1.0000
expert search	0.0107	0.0075	0.5037	1.0000	0.5227	1.0000
*facial reconstruction	0.0039	0.0039	0.0021	0.0021	0.2000	0.2000
*force feedback	0.0073	0.0050	0.6429	1.0000	0.6250	1.0000
functional programming	0.0244	0.0112	0.1687	0.2000	0.2038	0.1429
*glasgow haskell compiler	0.0101	0.0101	0.3333	0.3333	0.5000	0.5000
*grid computing	0.0123	0.0083	0.3095	0.3333	0.3333	0.3333
*group response systems	0.0054	0.0054	0.5000	0.5000	0.5000	0.5000
*handwriting pin	0.0054	0.0038	1.0000	1.0000	1.0000	1.0000
*haptic visualisation	0.0224	0.0102	0.6111	1.0000	0.5861	1.0000
*haptics	0.0407	0.0133	0.4929	0.5000	0.5819	0.5000
*hci issues in mobile devices	0.0268	0.0091	0.5821	1.0000	0.6160	1.0000
*human error health care	0.0064	0.0064	0.5821	1.0000	0.3333	0.3333
*information extraction	0.0069	0.0050	0.3333	0.3333	0.2500	0.2500
information retrieval	0.0308	0.0102	0.7496	1.0000	0.7468	1.0000
information security	0.0163	0.0120	0.7333	1.0000	0.7333	1.0000
machine translation	0.0117	0.0118	1.0000	1.0000	1.0000	1.0000
robotics	0.0404	0.1111	0.5565	1.0000	0.5523	1.0000

Table 5.4: Evaluation Results of Each Testing Query

5.1.2 Experimental Results

Table 5.3 shows the retrieval performance of the proposed LETOR techniques and without LETOR technique. It can be clearly seen that overall without LETOR gives better performance compared to using LETOR considering MAP, NDCG and MRR evaluating metrics. Coordinate Ascent gives second best performance. Therefore, the answer to the question is NO, using LETOR technique (without K-fold Cross-validation) does not enhance the performance of the retrieval system.

Table 5.4 shows the evaluation results of each testing query. Numbers highlighted in green are numbers higher than ones without LETOR. It has shown that applying LETOR technique using Coordinate Ascent algorithm to the testing queries given below outperformed ones without LETOR technique considering MAP/MRR values:

- force feedback
- haptic visualisation
- human error heath care
- information extraction
- information retrieval
- robotics

Training Queries	
*grid computing	*haptics
*hci issues in mobile devices	*neural network
operating system	computer vision
functional programming	*divergence from randomness
quantum computation	*facial reconstruction
*clustering algorithms	*eurosys 2008
*glasgow haskell compiler	*human error health care
*query expansion	*terrier
*different matching coefficients	*distributed predator prey
*parallel logic programming	information security
information retrieval	*group response systems
expert search	distributed systems
*empirical methods	*handwriting pin
machine translation	artificial intelligence
computer graphics	*force feedback
*wafer fab cost	*haptic visualisation
*3d audio	*3d human body segmentation
*discrete event simulation	*anchor text
data mining	robotics
*discrete mathematics	*ecir 2008
programming languages	*information extraction

Table 5.5: Training Queries for Fold 1

Testing Queries
*usability
stable marriage
constraint modelling
mobile hci
*language modelling
home networks
*multimodal
networking security
*suffix tree
*sound in multimedia human-computer interfaces
wireless sensor networks
game theory
*visual impairment

Table 5.6: Testing Queries for Fold 1

5.2 Evaluation With K-fold Cross-validation

In the previous section, the evaluation results have shown that applying LETOR makes the retrieval system perform poorer than not applying LETOR. This section aims to evaluate the performance of the system by applying K-fold Cross-validation technique described in section 2.2.8. We decided to choose $K = 5$. This is fairly normal for learning to rank [19]. Higher K is possible, but that infers that the LETOR needs more training. RankLib provides K-fold Cross-validation functionality. However, we will not use this functionality because some of our datasets results were considered only experts in University of Glasgow. For this reason, the datasets should be split into training, validation and testing queries manually to avoid one of them having only results from University of Glasgow [19].

5.2.1 Experimental Setting

As discussed earlier, we decided to choose $K = 5$ for K-fold Cross-validation. Each of the 5 folds has all of the queries: 60% for training, 20% for validation and 20% for testing [3]. Across the 5 folds, each query will appear 3 times in training, once in validation and once in testing queries. For each fold, training queries are trained and validated by validation queries to obtain a learned model. Then a learned model for each fold is applied to testing queries of that fold. Results of each run across the 5 folds are combined into one results file in TREC format. With the qrels file, this results file is then evaluated with trec_eval. Finally, one of the learned models is used by the system.

Fold 1

Tables 5.6, 5.5 and 5.7 show testing, training and validation queries respectively.

Validation Queries
programming languages
*text searching
*older adults use of computers
*music as navigation cues
*statistical inference
*road traffic accident statistics
*shoogle
*skill-based behavior
database
*manets
*suffix tree
*trec collection class
*matching

Table 5.7: Validation Queries for Fold 1

Training Queries	
*grid computing	*older adults use of computers
operating system	*statistical inference
functional programming	quantum computation
*clustering algorithms	*glasgow haskell compiler
*query expansion	*parallel logic programming
expert search	information retrieval
*suffix tree	machine translation
computer graphics	*discrete event simulation
*wafer fab cost	*discrete mathematics
data mining	*text searching
programming languages	*haptics
*neural network	*road traffic accident statistics
*music as navigation cues	*eurosys 2008
*facial reconstruction	*shoogle
*human error health care	*terrier
*distributed predator prey	database
*skill-based behavior	distributed systems
artificial intelligence	*force feedback
*manets	*anchor text
*trec collection class	robotics
*match	*ecir 2008

Table 5.8: Training Queries for Fold 2

Fold 2

Tables 5.9, 5.8 and 5.10 show testing, training and validation queries respectively.

Fold 3

Tables 5.12, 5.11 and 5.13 show testing, training and validation queries respectively.

Fold 4

Tables 5.15, 5.14 and 5.16 show testing, training and validation queries respectively.

Testing Queries
*different matching coefficients
*haptics
*hci issues in mobile devices
computer vision
*empirical methods
*divergence from randomness
*haptic visualisation
*3d human body segmentation
information security
*group response systems
*handwriting pin
*information extraction
*3d audio

Table 5.9: Testing Queries for Fold 2

Validation Queries
*usability
*stable marriage
constraint modelling
*mobile hci
*language modelling
home networks
*multimodal
networking security
*suffix tree
*sound in multimedia human-computer interfaces
*wireless sensor networks
game theory
*visual impairment

Table 5.10: Validation Queries for Fold 2

Training Queries	
*older adults use of computers	constraint modelling
operating system	*multimodal
*statistical inference	quantum computation
*query expansion	*usability
stable marriage	*parallel logic programming
information retrieval	expert search
home networks	networking security
*suffix tree	*wireless sensor networks
computer graphics	*wafer fab cost
game theory	*visual impairment
programming languages	*text searching
*neural network	mobile hci
*road traffic accident statistics	*facial reconstruction
*music as navigation cues	*shoogle
*terrier	*sound in multimedia human-computer interfaces
distributed systems	skill-based behavior
database	*force feedback
*language modelling	*manets
*trec collection class	*match

Table 5.11: Training Queries for Fold 3

Testing Queries
*discrete event simulation
data mining
*discrete mathematics
*grid computing
*haptics
*eurosys 2008
functional programming
*human error health care
*distributed predator prey
*clustering algorithms
*glasgow haskell compiler
artificial intelligence
expert search
*anchor text
robotics
*machine translation
*ecir 2008

Table 5.12: Testing Queries for Fold 3

Validation Queries
*different matching coefficients
*haptics
*hci issues in mobile devices
computer vision
*empirical methods
*divergence from randomness
*haptic visualisation
*3d human body segmentation
information security
*group response systems
*handwriting pin
*information extraction
*3d audio

Table 5.13: Validation Queries for Fold 3

Training Queries	
*older adults use of computers	*constraint modelling
*hci issues in mobile devices	*multimodal
*statistical inference	*usability
*different matching coefficients	stable marriage
*empirical methods	home networks
networking security	*suffix tree
wireless sensor networks	game theory
*visual impairment	*3d audio
*haptics	programming languages
*text searching	mobile hci
computer vision	*divergence from randomness
*road traffic accident statistics	*music as navigation cues
*shoogle	*sound in multimedia human-computer interfaces
information security	*group response systems
*skill-based behavior	database
*handwriting pin	*language modelling
*manets	*haptic visualisation
*3d human body segmentation	*trec collection class
*match	*information extraction

Table 5.14: Training Queries for Fold 4

Testing Queries
programming languages
*neural network
operating system
*facial reconstruction
quantum computation
*terrier
distributed systems
*query expansion
*force feedback
*parallel logic programming
expert search
information retrieval
computer graphics
*wafer fab cost

Table 5.15: Testing Queries for Fold 4

Validation Queries
*discrete event simulation
data mining
*discrete mathematics
*grid computing
*haptics
*eurosys 2008
functional programming
*human error health care
*distributed predator prey
*clustering algorithms
*glasgow haskell compiler
artificial intelligence
expert search
*anchor text
robotics
machine translation
*ecir 2008

Table 5.16: Validation Queries for Fold 4

Training Queries	
*grid computing	constraint modelling
*hci issues in mobile devices	*multimodal
functional programming	*clustering algorithms
*glasgow haskell compiler	*usability
*different matching coefficients	stable marriage
expert search	*empirical methods
home networks	networking security
*suffix tree	machine translation
*wireless sensor networks	game theory
*visual impairment	*3d audio
*discrete event simulation	data mining
*discrete mathematics	*haptics
mobile hci	computer vision
*divergence from randomness	*eurosys 2008
*human error health care	*distributed predator prey
*sound in multimedia human-computer interfaces	information security
*group response systems	*handwriting pin
artificial intelligence	language modelling
*haptic visualisation	*3d human body segmentation
*anchor text	robotics
*ecir 2008	*information extraction

Table 5.17: Training Queries for Fold 5

Testing Queries
programming languages
*text searching
*older adults use of computers
*music as navigation cues
*statistical inference
*road traffic accident statistics
*shoogle
*skill-based behavior
database
*manets
*suffix tree
*trec collection class
*match

Table 5.18: Testing Queries for Fold 5

Fold 5

Tables 5.18, 5.17 and 5.19 show testing, training and validation queries respectively.

5.2.2 Experimental Results

Table 5.20 shows the retrieval performance of the proposed LETOR techniques using K-fold cross-validation techinque and without LETOR techinque. It can be clearly seen that overall the evaluation result is similar to one not using K-fold cross-validation (without LETOR, the performance is better than with LETOR). Coordinate Ascent still gives second best performance. But using K-fold cross-validation technique, the performance is better than not using K-fold cross-validation (see last section). Therefore, the answer to the question is still NO, using LETOR technique with K-fold cross-validation does not enhance the performance of the retrieval system. Table 5.21 shows the evaluation results of each testing query. Numbers highlighted in green show testing queries applying Coordinate Ascent whose evaluation metrics are higher than ones without LETOR. Numbers highlighted in red show that testing queries applying AdaRank algorithm perform better than ones

Validation Queries
programming languages
*neural network
operating system
*facial reconstruction
quantum computation
*terrier
distributed systems
*query expansion
*force feedback
*parallel logic programming
expert search
information retrieval
computer graphics
*wafer fab cost

Table 5.19: Validation Queries for Fold 5

Algorithm	MAP	NDCG	MRR	Number of Relevant Experts
AdaRank	0.0469	0.2397	0.0843	106
Coordinate Ascent	0.5321	0.6703	0.6366	106
Without LETOR	0.5499	0.6871	0.6484	106

Table 5.20: Evaluation Results Using K-fold Cross-validation

Testing Queries	AdaRank		Coordinate Ascent		Without LETOR	
	MAP	MRR	MAP	MRR	MAP	MRR
3d audio	0.0227	0.0227	1.0000	1.0000	1.0000	1.0000
3d human body segmentation	0.0239	0.0222	0.1824	0.2000	0.1824	0.2000
anchor text	0.0263	0.0088	0.7894	1.0000	0.7894	1.0000
artificial intelligence	0.0344	0.0233	0.6064	1.0000	0.5972	1.0000
clustering algorithms	0.0166	0.0071	0.3048	0.2000	0.3048	0.2000
computer graphics	0.1429	0.1429	0.5000	0.5000	0.5000	0.5000
computer vision	0.0430	0.0667	0.4611	0.5000	0.4611	0.5000
data mining	0.0181	0.0118	0.9167	1.0000	0.9167	1.0000
different matching coefficients	0.0076	0.0052	0.5833	0.5000	0.5833	0.5000
discrete event simulation	0.0078	0.0078	0.3333	0.3333	0.3333	0.3333
discrete mathematics	0.0164	0.0115	0.1214	0.1000	0.1394	0.1250
distributed predator prey	0.0058	0.0058	0.0038	0.0038	0.1111	0.1111
divergence from randomness	0.0205	0.0122	0.9167	1.0000	0.9167	1.0000
ecir 2008	0.0152	0.0070	0.8304	1.0000	0.8125	1.0000
empirical methods	0.0316	0.0286	0.2154	0.1667	0.2154	0.1667
eurosys 2008	0.0055	0.0055	1.0000	1.0000	1.0000	1.0000
expert search	0.0106	0.0076	0.5040	1.0000	0.5227	1.0000
facial reconstruction	0.0270	0.0270	0.1429	0.1429	0.2000	0.2000
force feedback	0.5141	1.0000	0.6429	1.0000	0.6250	1.0000
functional programming	0.0244	0.0112	0.0952	0.1111	0.2038	0.1429
glasgow haskell compiler	0.0101	0.0101	0.3333	0.3333	0.5000	0.5000
grid computing	0.0123	0.0083	0.3333	0.3333	0.3333	0.3333
group response systems	0.0047	0.0047	0.5000	0.5000	0.5000	0.5000
handwriting pin	0.0422	0.0400	1.0000	1.0000	1.0000	1.0000
haptic visualisation	0.0328	0.0400	0.5861	1.0000	0.5861	1.0000
haptics	0.0331	0.0104	0.4091	0.5000	0.5819	0.5000
hci issues in mobile devices	0.0319	0.0357	0.6160	1.0000	0.6160	1.0000
human error health care	0.0064	0.0064	0.3333	0.3333	0.3333	0.3333
information extraction	0.0254	0.0217	0.2500	0.2500	0.2500	0.2500
information retrieval	0.2538	1.0000	0.7622	1.0000	0.7468	0.7468
information security	0.0275	0.0476	0.7333	1.0000	0.7333	1.0000
machine translation	0.0117	0.0118	1.0000	1.0000	1.0000	1.0000
robotics	0.0404	0.1111	0.5523	1.0000	0.5523	1.0000

Table 5.21: Evaluation Results of Each Testing Query Using K-fold Cross-validation

applying Coordinate Ascent. The evaluation results of the testing queries applied with LETOR using Coordinate Ascent given below outperform ones without LETOR technique considering MAP/MRR values:

- artificial intelligence
- ecir 2008
- force feedback
- information retrieval

5.3 Causes of Learning to Rank Failure

It has shown in the last section that applying LETOR technique fails. The followings are reasons behind it [19]:

- The size of training/validation dataset is not big enough.
- Selected LETOR Features are not very useful.
- The number of publications and funded projects is not balanced. We have 22225 publications but only 369 funded projects. This results in features related to publications outweighing funded project features in both query dependent and independent features.

For AdaRank, the performance is considerably poorer than Coordinate Ascent because AdaRank requires a large set of validation data to avoid overfitting [19]. Therefore, we decided to apply K-fold Cross-validation technique as it gave better performance compared to not applying K-fold Cross-validation technique (see last section).

Chapter 6

Conclusions

In this project, we applied 2 LETOR algorithms: AdaRank and Coordinate Ascent, to combine different kinds of expertise evidence in order to improve the performance of the system. There are various LETOR algorithms used in industry such as LambdaMART, Random Forests etc. However, AdaRank and Coordinate Ascent algorithms are chosen due to its simplicity and effective performance. Our experimental results (see last chapter) have shown that

- AdaRank does not perform very well compared to Coordinate Ascent.
- Applying LETOR does not improve the performance of the system.

The reasons behind LETOR failure can be found in the previous chapter.

6.1 Requirements Achieved

The following is a summary of the requirements (see chapter 3.2.1) that have been achieved.

- All of the system's Must Have requirements.
- All of the system's Should Have requirements.
- All of the non-functional Requirements.

The system's Would like to Have and Could Have requirements could not be achieved because of time constraint. The Could Have requirements can be difficult for 4th year students [19] as they are too complex and require full understanding of the behaviours of the technique used [19]. However, the aim of the project is to experiment 2 LETOR algorithms: AdaRank and Coordinate, and apply the one performing better.

6.2 Challenges

During the development of the project, we have encountered various challenges. This section will discuss challenges encountered.

6.2.1 LETOR

Learning to rank is a relatively new field in which machine learning algorithms are used to learn this ranking function [37]. It is of particular importance for search engines to accurately tune their ranking functions as it directly affects the search experience of users [37]. Choosing the right LETOR technique is research based and it requires an understanding of the LETOR techniques behaviours which could possibly take weeks or months to come up with the best approach.

6.2.2 Relevance Judgement

Relevance judgement (see chapter 5) by human assessors is considered one of the most difficult parts in this project since it is based on person's experience [32]. For this reason, constructing good quality training, testing and validating datasets is also a challenge as it directly affects the performance of the system [19].

6.3 Problems Encountered

During the development of the project, we have encountered various problems. Some of them can be addressed and some can not. In this section, we list a number of problems encountered during development life cycle.

6.3.1 Expert's Name Pattern Matching

In chapter 4.3, we discussed approaches used to handle differences between expert's name formats in each funded projects datasource. Still, we have problems regarding pattern matching between expert's names from Grant on the Web [1]. Consider Professor Joemon Jose, he is currently a professor in computing science at Glasgow University. Grant on the Web records his name as Jose, Professor JM. This makes it impossible to accurately pattern match the name with our known candidates if there are experts named Professor Joemon Jose and Professor Jasmine Jose because Grant on the Web will record their names in the same format: Jose, Professor JM. Although we could solve the problem by taking the university they are lecturing into account, the problem still exists if they are from the same university.

6.3.2 Lack of LETOR Algorithms Explanation

Clear explanations of the Coordinate Ascent and AdaRank LETOR algorithms are difficult to find. Trying to understand the nature of the algorithms is not an easy task for 4th year student as it is based on mathematics and statistics.

6.3.3 Poor Quality of Relevance Judgements

In chapter 5, we talked about the problems we encountered when constructing relevance judgements. That is, we managed to construct training queries whose results (relevant experts) are only from University of Glasgow (see last chapter). The reason behind this is that it is difficult to judge experts not in University of Glasgow because we do not know them. As a consequence of poor relevance judgements, the dataset used for training and evaluation becomes poor [19].

6.4 How would I have done differently?

In sections 6.2 and 6.3, we discussed challenges and problems encountered during the development of the system. However, these issues could be minimised or eliminated if

- The behaviours of LETOR algorithms is well understood at the early stage of the development - this means that we could experiment only suitable algorithms whose behaviour is applicable to the task.
- Various funded projects datasources should be used to increase the number of expertise evidence - this means that we would have more training, validation and testing datasets.
- Relevance Judgements are prepared by experienced persons - this means that the performance of LETOR techniques could be enhanced as relevance judgements are used to produce training and validation datasets.
- Tools used in this project are familiarised at the early stage of the development - this means that we could focus more on other aspects of the project.

6.5 Future Work

In the previous chapter, the experimental results have shown that applying LETOR techniques using AdaRank and Coordinate Ascent algorithms does not improve the performance of the system. This might not be true however, if the system applies other LETOR algorithms provided by RankLib since each algorithm has its own behaviours. In addition, good training/validation datasets and feature selections play the main roles to the performance of the system [19]. I strongly believe that the intuitions (features) proposed in chapter 3.2 are very useful but the main reason why LETOR fails is due to small training dataset. We had 34 training queries which is very small. Each testing query can be analysed to determine why applying a learned model gives better or worse performance compared to not applying LETOR. This can be done by for example, analysing the number of all terms, the number of funded projects terms and of publications terms relevant to each testing query.

Moreover, for IR systems to get better performance, not only LETOR technique is contributed to the success of the retrieval system, but also other aspects in IR such as tokenisation technique, stemming technique and retrieval models. LETOR is just an optional technique in machine learning used to optimise the ranking based on training/validation datasets. However, all of these aspects should be experimented.

Bibliography

[1] Engineering and physical sciences research council (epsrc). http://gow.epsrc.ac.uk/.

[2] Information retrieval. http://en.wikipedia.org/wiki/Information_retrieval.

[3] K-fold cross-validation. http://en.wikipedia.org/wiki/Cross-validation_(statistics)#K-fold_cross-validation.

[4] Learning to rank. http://en.wikipedia.org/wiki/Learning_to_rank.

[5] Merge sort. http://en.wikipedia.org/wiki/Merge_sort.

[6] Pagerank. http://en.wikipedia.org/wiki/PageRank.

[7] Polymorphism. http://en.wikipedia.org/wiki/Polymorphism_(computer_science).

[8] qrels file. http://www.mpi-inf.mpg.de/~dfischer/manual/manual_qrelformat.html.

[9] Ranklib. http://sourceforge.net/p/lemur/wiki/RankLib/.

[10] Relevance judgement file list. http://trec.nist.gov/data/qrels_eng/.

[11] Research councils uk - gateway to research. http://gtr.rcuk.ac.uk/.

[12] The scottish informatics and computer science alliance expert search system. http://experts.sicsa.ac.uk/.

[13] The scottish informatics and computer science alliance (sicsa). http://www.sicsa.ac.uk/home/.

[14] Terrier. http://www.terrier.org/.

[15] The text retrieval conference (trec). http://trec.nist.gov/overview.html.

[16] Trec result file format. http://ir.iit.edu/~dagr/cs529/files/project_files/trec_eval_desc.htm.

[17] Information retrieval system evaluation. http://nlp.stanford.edu/IR-book/html/htmledition/information-retrieval-system-evaluation-1.html, 2009.

[18] Inverted index. http://en.wikipedia.org/wiki/Inverted_index, 2012.

[19] Conversation of dr. craig macdonald, 2014.

[20] Discounted cumulative gain. http://en.wikipedia.org/wiki/Discounted_cumulative_gain, 2014.

[21] Mean reciprocal rank. http://en.wikipedia.org/wiki/Mean_reciprocal_rank, 2014.

[22] Search engine. http://en.wikipedia.org/wiki/Search_engine_(computing), 2014.

[23] Term frequency–inverse document frequency (tf-idf). http://en.wikipedia.org/wiki/Tf%E2%80%93idf, 2014.

[24] Tokenization. http://en.wikipedia.org/wiki/Tokenization, 2014.

[25] Web search engine. http://en.wikipedia.org/wiki/Web_search_engine, 2014.

[26] Iadh Ounis Craig Macdonald. Voting for candidates: Adapting data fusion techniques for an expert search task. pages 387, 388, 389, 2006.

[27] Rodrygo L.T. Craig Macdonald and Iadh Ounis. About learning models with multiple query dependent features. 2012.

[28] Prof. Joemon M Jose. Architecture of retrieval systems, 2013.

[29] Prof. Joemon M Jose. Information retrieval, 2013.

[30] Prof. Joemon M Jose. Information retrieval - evaluation methodology, 2013.

[31] Prof. Joemon M Jose. Probabilistic retrieval model, 2013.

[32] Professor Joemon Jose. Text classification, 2013.

[33] Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. 2011.

[34] Craig Macdonald. *The Voting Model for People Search*. PhD thesis, Department of Computing Science, Faculty of Information and Mathematical Sciences, University of Glasgow, 2009.

[35] Afshin Rostamizadeh Mehryar Mohri and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[36] Donald Metzler. Automatic feature selection in the markov random field model for information retrieval.

[37] Yi Chang Olivier Chapelle. Yahoo! learning to rank challenge overview. 2011.

[38] Mark Girolami Simon Rogers. *A First Course in Machine Learning*. 2011.