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 17, 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 engines to obtain the most effective search results. Most of the project's aims and requirements were satisfied. However, various difficulties were encountered in the late stages of the development life-cycle which ultimately led to a reduction of the system's scope.

Acknowledgements

Thanks to Dr. Craig Macdonald for his advice and supervision throughout this project.

Contents

0.1 Introduction

0.1.1 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" [8]. 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 below.

- 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

0.1.2 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" [20, P. 388]. In an expert search task, the users' need, expressed as queries, is to identify people who have relevant expertise to the need [20, 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 [20, P. 388]. The profiles represent the system's knowledge of the expertise of each candidate, and they are ranked in response to a user query [20, 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 evdience of expertise to rank candidate persons with respect to their predicted expertise about the query.

0.1.3 Definition of Mining Academic Expertise from Funded Research and Aims

http://experts.sicsa.ac.uk/ [7] 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 [6]. To integrate academic funded projects and publications together, Learning to Rank Algorithms for Information Retrieval (IR) are applied in this project.

0.1.4 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 performance of http://experts.sicsa.ac.uk/ [7]

0.1.5 Overview

In this dissertation, Section **??** discusses about Requirements Specification which is grouped into functional and non-functional requirements. Section 0.2 aims to explain the backgrounds of the project to readers. This section is necessary for readers to understand subsequent sections. Section 0.4 discusses about Designs and Implementations of the system which also references to deployed SICSA search system [7]. Section 5 provides results and analysis of the techiques used. This section can be viewed as the most important part of the whole project since it analyses adding expertise evidence improves the relevance of the search or not. The last section is the conclusion of the project.

0.2 Background

0.2.1 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 1 illustrates search process. As information resources were not originally intended for access (Retrieval of unstructured data)[P. 7] [23]. 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 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 0.2.1.

A search engine is an information retrieval system designed to help find information stored on a computer system [16]. The search results are usually presented in a list 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 [16]. 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 [19].

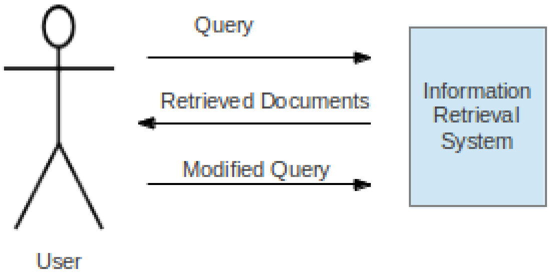


Figure 1: Search Process

```
<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: Document

Brief Overview of Information Retrieval System Architecture

In IR systems, two main objectives have to be met [22] - first, the results must satisfy user - this means retrieving information to meet user's information need, second, retrieving process must be fast. This section is devoted to a brief overiew of the architecture of IR systems which makes readers understand 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 done offline. There are 4 steps in indexing process and each process is performed sequentially [22]:

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

Given a document containing Albert Einstein's quote about life,

> 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.

it can be illustrated in a terms-frequency table.

Table 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.

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 1: Terms and Frequency

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

Table 2: Terms and Frequency After Stopwords Removal

Tokenisation is the process of breaking a stream of text into words called tokens(terms) [18]. The stream of text will be used by other indexing process.

Stopwords Removal is the process of removing stopwords in order to reduce the size of the indexing structure [22, P. 15]. This also results in efficient lookup. Table 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 [22, 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

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

Table 3: Terms and Frequency After Stemming

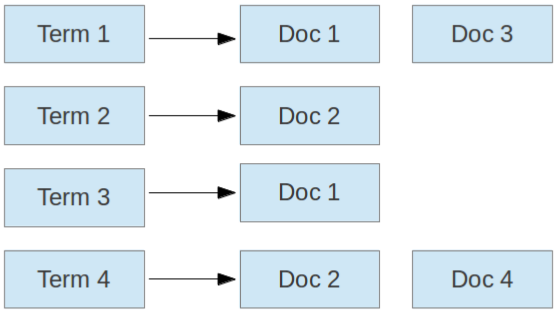


Figure 3: Simple Inverted Index

of word meanings. For instance, the root form of the term "gravitation" is "gravity". But the meaning of "gravitation" is different from "gravity".

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) [12]. 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 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 freqency of the term etc.

Retrieval Models

Last 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.

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 [17]. As the name suggest, 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 [17]. 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 apprear in the corpus. The following is the formula of **tf-idf** weighting model:

$$W_{fk} = f_{fd}(logN/D_k) \tag{1}$$

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 $logN/D_k$ is the idf of the model. This model is a non-probabiltic approach and one of the easiest models in IR.

BM25 - In addition to non-probabiltic approach, BM25 and Language Model are very popular probabiltic retrieval models. We will briefly discuss these models. Why use probabilities? In section 0.2.1, 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 [25, P. 7].

Let x be a document in the collection, $R_{d\vec{q}}$ represent the *relevance* of a document d with respect to query $\vec{q}$ and $NR_{d\vec{q}}$ represent *non-relevance* of a document d with respect to query $\vec{q}$. We will use R and NR to represent the former and the latter for brevity. The aim is to find $P(R|x)$ - the probability that a retrieved document x is relevant.

$$P(R|x) = \frac{P(x|R)P(R)}{P(x)} \tag{2}$$

$$P(NR|x) = \frac{P(x|NR)P(NR)}{P(x)} \tag{3}$$

The above two equations are derived from Bayes theorem. Note that $P(R|x) + P(NR|x) = 1$. If $P(R|x) > P(NR|x)$, then a document x is relevant. This tells us when to stop ranking. To move further, we have to understand **Binary Independence Model** (BIM). In BIM, binary means boolean (0 or 1) and documents are represented as binary vectors of terms:

$$\vec{x} = (x_t, ..., x_n), 1 <= t <= n$$

where x_t is 1 if and only if the term x_t is present in document $\vec{x}$, 0 otherwise. Independence means terms occur in documents independently. Althought this assumption is far from correct, it often gives satisfactory results in practice. To derive a ranking for query terms, we do the following:

$$\frac{P(R|x)}{P(NR|x)} = \frac{\frac{P(x|R)P(R)}{P(x)}}{\frac{P(x|NR)P(NR)}{P(x)}} \tag{4}$$

Since we are only ranking documents, there is thus no need for us to estimate $P(R)$ and $P(NR)$. We now end up with

$$\frac{P(R|x)}{P(NR|x)} = \frac{P(x|R)}{P(x|NR)} \tag{5}$$

It is at this point that we make the **Naive Bayes conditional independence assumption** [3, P. 261] that the presence or absence of a word in a document is independent of the presence or absence of any other word (given the query):

$$\frac{P(R|x)}{P(NR|x)} = \prod_{t:x_t=1}^{M} \frac{P(x_t = 1|R)}{P(x_t = 1|NR)} \prod_{t:x_t=0}^{M} \frac{P(x_t = 0|R)}{P(x_t = 0|NR)} \tag{6}$$

where $P(x_t = 1|R)$ is a probability of a term x_t appearing in a document relevant to a query, $P(x_t = 1|NR)$ is a probability of a term x_t appearing in a document irrelevant to a query, $P(x_t = 0|R)$ is a probability of a term

	Document	Relevant (R)	Irrelevant (NR)
Term present	$x_t = 1$	p_t	u_t
Term absent	$x_t = 0$	$1 - p_t$	$1 - u_t$

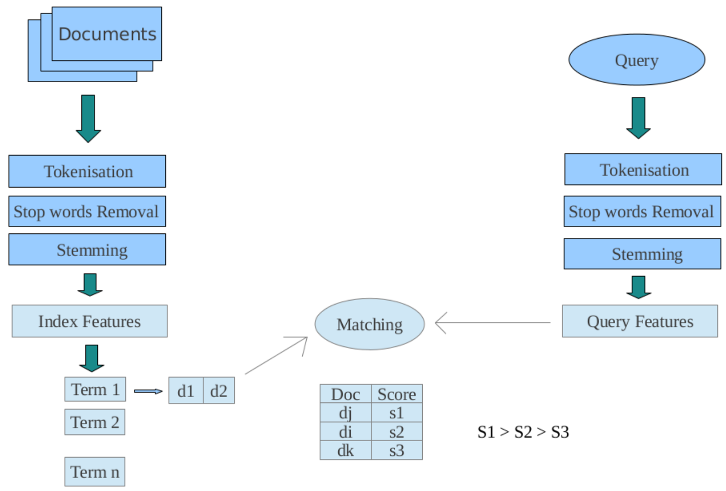


Figure 4: Retrieval Process

x_t not appearing in a document relevant to a query and $P(x_t = 0|NR)$ is a probability of a term x_t not appearing in a document irrelevant to a query.

For brevity, let $p_t = P(x_t = 1|R)$ be the probability of a term appearing in a document relevant to the query, and $u_t = P(x_t = 1|NR)$. These quantities can be visualised in table 0.2.1.

From equation 6, we now have the following:

$$\frac{P(R|x)}{P(NR|x)} = \prod_{t:x_t=1}^{M} \frac{p_t}{u_t} (\prod_{t:x_t=1}^{M} \frac{1-u_t}{1-p_t} \prod_{t:x_t=1}^{M} \frac{1-p_t}{1-u_t}) \prod_{t:x_t=0}^{M} \frac{1-p_t}{1-u_t} \tag{7}$$

Let us make an additional simplifying assumption that terms not occurring in the query are equally likely to occur in relevant and nonrelevant documents: that is, if $q_t = 0$ then $p_t = u_t$. We now end up with the equation as follows:

$$\frac{P(R|x)}{P(NR|x)} = \prod_{t:x_t=1}^{M} \frac{p_t(1-u_t)}{u_t(1-p_t)} \tag{8}$$

From equation 8, we can apply logarithm to avoid zero product and the resulting quantity used for ranking is called the Retrieval Status Value (RSV).

Not done

Retrieval Process in Information Retrieval

Section 0.2.1 and Section 0.2.1 explained basic indexing process and a few retrieval models respectively. In this section, we will see how documents are retrieved. Figure 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 and all features are extracted (section 0.2.1).

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 makes use of retrieval model explained in section 0.2.1.

... Not done...

In this project, special model is used for expert search system. It will be discussed in section 0.2.4. Once scores have been assigned to relevant documents, the system rank all documents in order of decreasing scores and show to the user. Figure 4 gives a graphical representation of retrieval process.

Evaluation

This section is devoted to backgrounds of evaluation of IR systems. It is very important as it is a background for Evaluation Section. Since IR is research-based, understanding how evaluation is carried out will give a background for the readers to determine whether this project is achieved or not. In IR, there are 3 main reasons for evaluating IR systems [24, 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 [11]:

1. A document collection.
2. A test suite of information needs, expressible as queries.
3. A set of relevance judgments, standardly 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 information need, a document in the test collection is given a binary classification as either relevant or nonrelevant [11]. However, this can be extended by using numbers as an indicator of the degree of relevancy. For example, documents labelled 2 is more relevant than documents labelled 1, or documents labelled 0 is not relevant. There are a number of test collection stardards. In this project, Text Retrieval Conference (TREC) is used since it is widely used in the field of IR.

Precision and Recall

The function of an IR system is to [24, P. 10]:

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

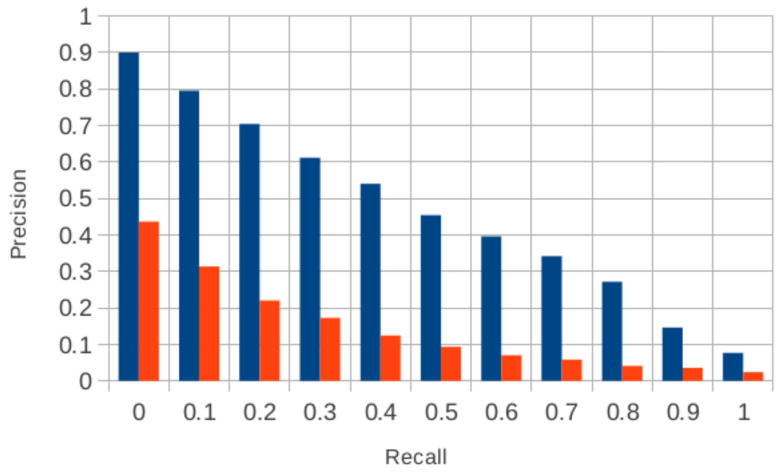


Figure 5: Precision-Recall Graph

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

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

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 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 be 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 [15]. 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) To understand nDCG, first of all, we have to **understand Cumulative Gain (CG)** and and **Discounted Cumulative Gain (DCG)**. CG is the predecessor of DCG and it is the sum of the graded relevance values of all results in a search result list [14]. The CG at a particular rank position p is defined as:

NOT DONE..

is a measure for ranking quality and the usefulness (gain) of an item based on its relevance and position in the provided list.

0.2.2 Learning to Rank

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 [26]. Employing learning to rank techniques to learn the ranking function is viewed as a promising approach to information retrieval [26]. In particular, many learning to rank aproaches attempt to learn a combination of features (called the learned model) [21, 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 [21, P. 3]. In learning to rank, a feature is a binary or numberical indicator representing the quality of a document, or its relation to the query [21, P. 4]. This will be discussed in more details in later section.

Query Dependent Feature

Figure 1 shows a simple search process in IR. After a user submits a query into an IR system. The system ranks the results with respect to the query and returns a result set to the user. It can be clearly seen that the results 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.

Query Independent Feature

In contrast to Query Dependent Feature, a feature that does not depend on a user query is called query independent feature. This feature is fixed for each document. For now, it is better to not dig into great details about this because this will be focused in later section.

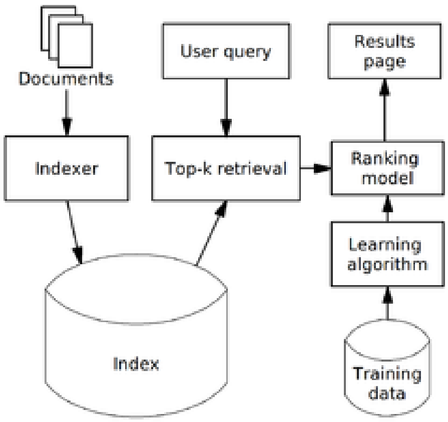


Figure 6: An architecture of a machine-learned IR system from http://en.wikipedia.org/wiki/Learning_to_rank

Obtaining and Deploying a Learned Model

The general steps for obtaining a learned model using a learning to rank technique are the following [21, 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 [21, 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 6 illustrates an architecture of a machine-learned IR system. The architecture will be discussed in more details in Design and Implementation Section.

0.2.3 Tools

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 [9] 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 [10]. 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.

RankLib

RankLib [5] is an open source library of learning to rank algorithms. It also implements many retrieval metrics as well as provides many ways to carry out evaluation. Currently eight 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.

trec_eval

trec_eval is the standard tool used by the TREC community [10] 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.

0.2.4 Expert Search

In section 0.1.2, we discussed about the definition of Expert Search. This section will give 5 scenarios proposed by Yimam-Seid & Kobsa why expert search is needed [20, P. 387].

> *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.*

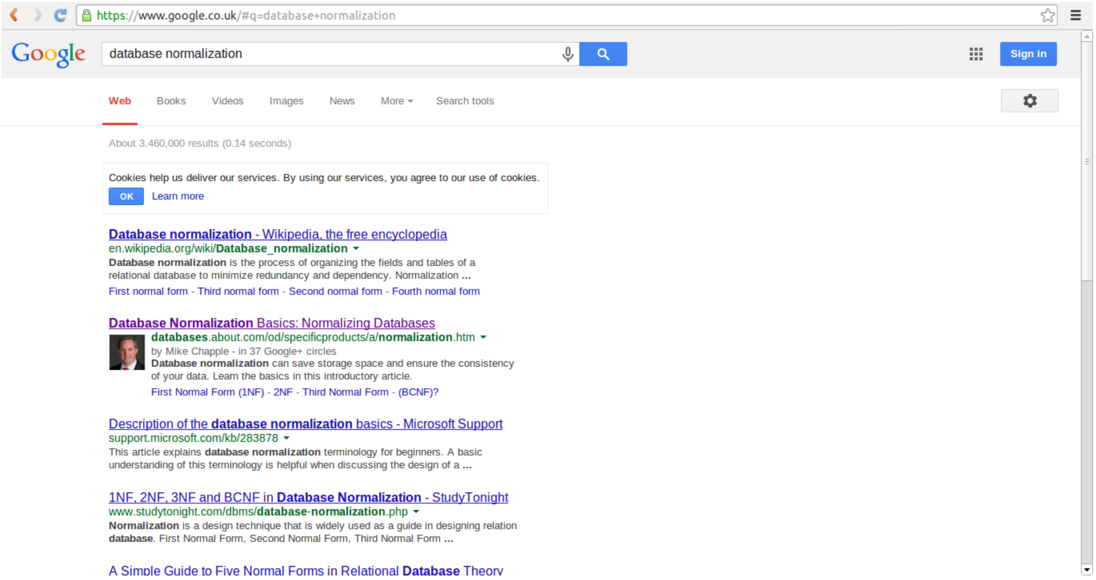


Figure 7: Sample Query

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. Socialisation need - the user may prefer that the human dimension be involved, as opposed to interacting with documents and computers.

Presenting Query Results

Figure 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 normalize database tables", first of all, he needs to interpret his information need into a query which is "database normalization". 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 [13]:

- URL
- Photo
- Author
- 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 7 shows the results of the query "database normalization". 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.

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

Table 4: R(Q)

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

Table 5: Profiles

Voting Technique

In section 0.2.1, we very briefly talked about weighting model. In other words, how documents are assigned scores using tf-idf. In this section, it aims to give an overview of voting technique used in this project. To understand this section, readers must understand what data fusion technique is. "Data fustion 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" [20, 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 [20, 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" [20, P. 387]. To give a simple example how data fusion 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). Consider the simple example in Tables 4 and 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$\}$. However, in this project, the technique used is expCombMNZ and the formular is as follows

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

where $|R(Q) \cap profile(C)|$ is the number of documents from the profile of candidate C that are in the ranking R(Q).

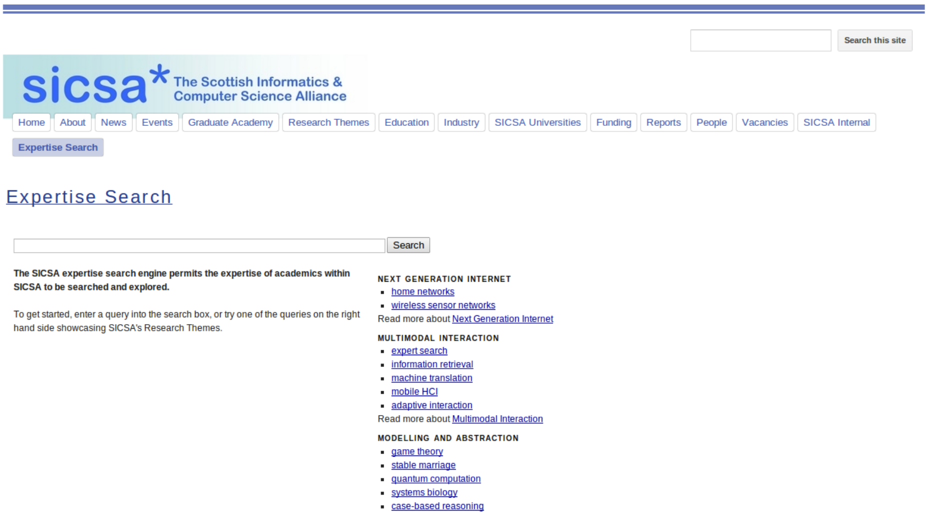


Figure 8: Home Page

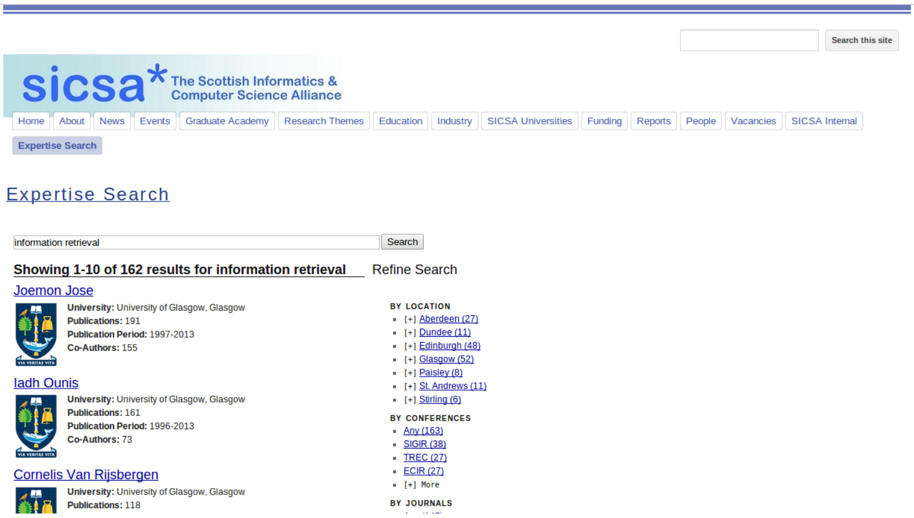


Figure 9: Experts In Response to “information retrieval” Query

0.3 I dont know

Current System

Figure 8 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 differnet 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 9 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 retrieved. Each result in the search panel includes expert’s details: university, number of publications, publication period and total number of coauthors.

Figure 10 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.

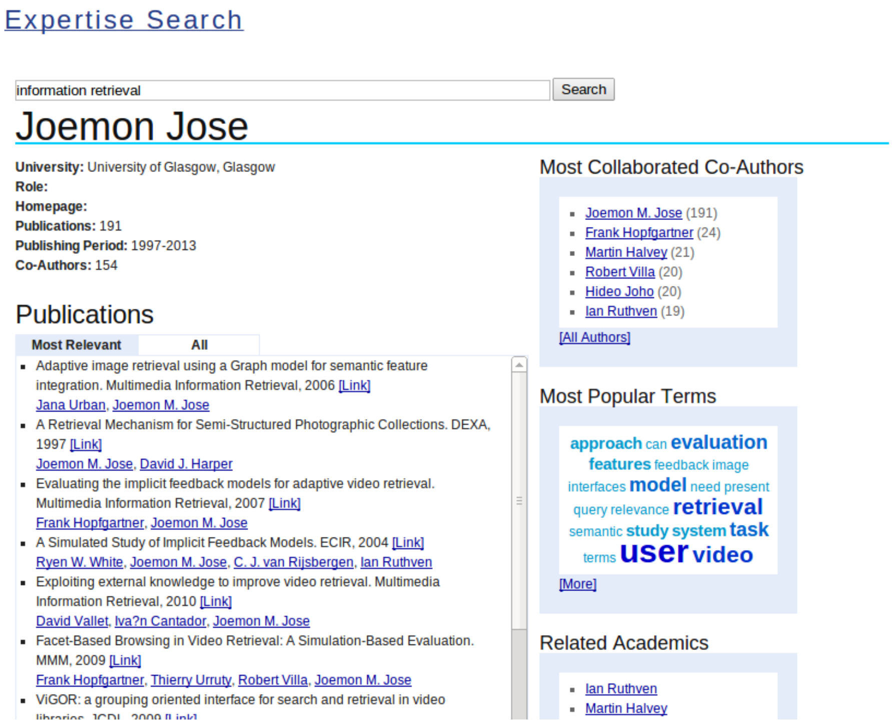


Figure 10: Old Expert's Profile Facet

Proposals

In section 0.2.4, it was obvious that some evidence could be used to convince users how reliable a document (link) is. In this secion, the evidence within the context of this project will be discussed. As discussed in 0.1.3, this project makes use of publications and funded projects as expertise evidence. Based on these evidence, what makes an expert a good expert? Well, it is common sense to propose that an expert is professional if

- 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.

It can be seen clearly that these assumptions or features in learning to rank are independent on the query a user submits. These features are query independent (section 0.2.2). However, query dependent features (section 0.2.2) should take into account as well. In this project, there are 2 query dependent features: funded project and publication features. To sum up, a good expert should have high scores in both query dependent and independent features.

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 old SICSA system (section 0.3).
- Research into presenting query results (section 0.2.4).
- Research into learning to rank techniques to enhancing the performance of the retrieval system by integrating different kinds of expertise evidence (section 0.2.2).
- Discussion with Dr. Craig Macdonald.

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

Functional Requirements

Must Have

- Extracting funded projects from different 2 sources: Grant on the Web [1] and Research Councils UK [6].
- Integrating publications and funded projects as expertise evidence.
- Utilizing learning to rank techniques with an attempt to enhance the performance of the retrieval system.
- Able to conclude that applying learning to rank techniques helps improve the performance of the retrieval system using evaluation matrices discussed in section 0.2.1.

Should Have

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

Could Have

- not yet

Would Like to Have

- Ability to upload funded projects manually by member 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.

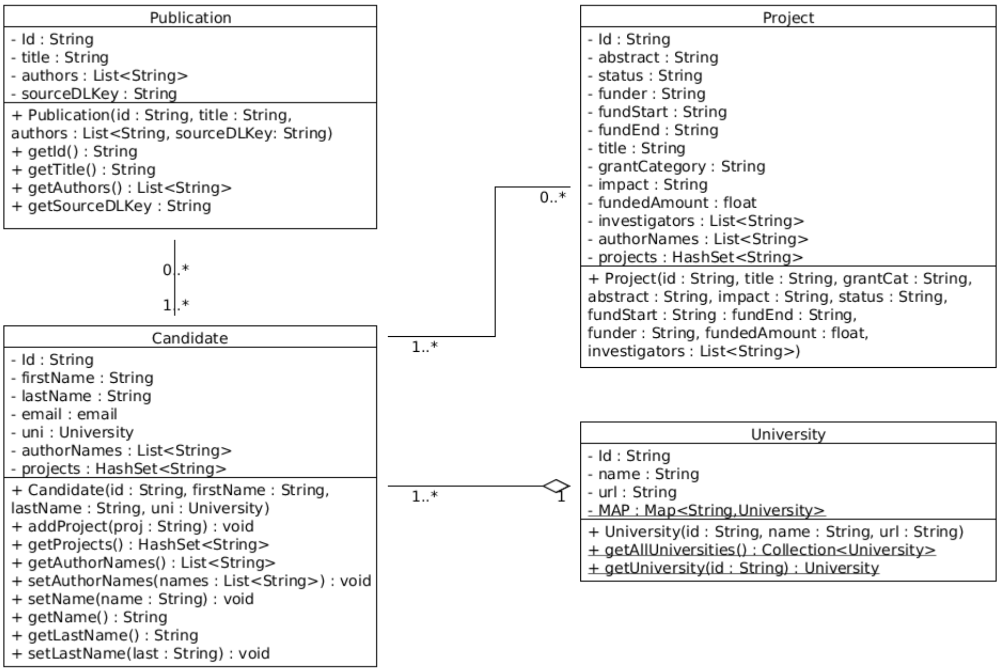


Figure 11: Class Diagram

Design and Architecture

0.4 Implementation

0.4.1 Overview of Important Class Diagrams

As stated in section 0.1.3, this project extends from SICSA project which uses only publications as expertise evidence. The aim of this project is to integrate funded projects with publications to improve the performance of the retrieval. Figure 11 shows the relationship between Candidate, 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.

As stated in section 0.3, 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 12 shows class diagrams of AcademTechQuerying and RetrievedResults. As stated in section 0.2.3, the search engine platform used in this project is Terrier. It handles indexing and retrieval processes discussed in section 0.2.1. 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 important method of AcademTechQuerying is processQuery(). It returns RetrievedResults. This method takes a query string, supportDocs and candId as arguments. The first argument tells the system that a query is

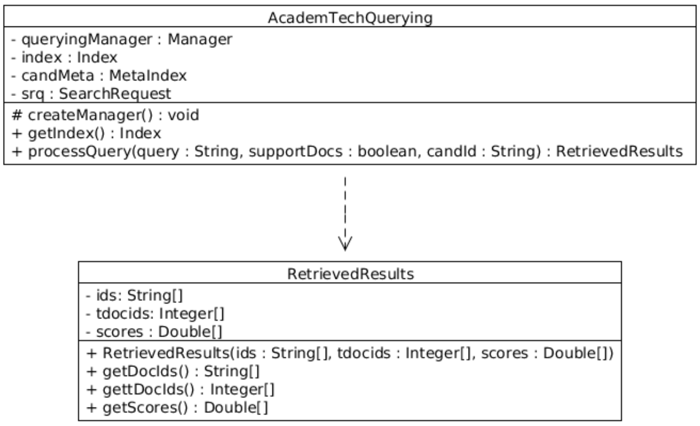


Figure 12: AcademTechQuerying Class and RetrievedResults Class

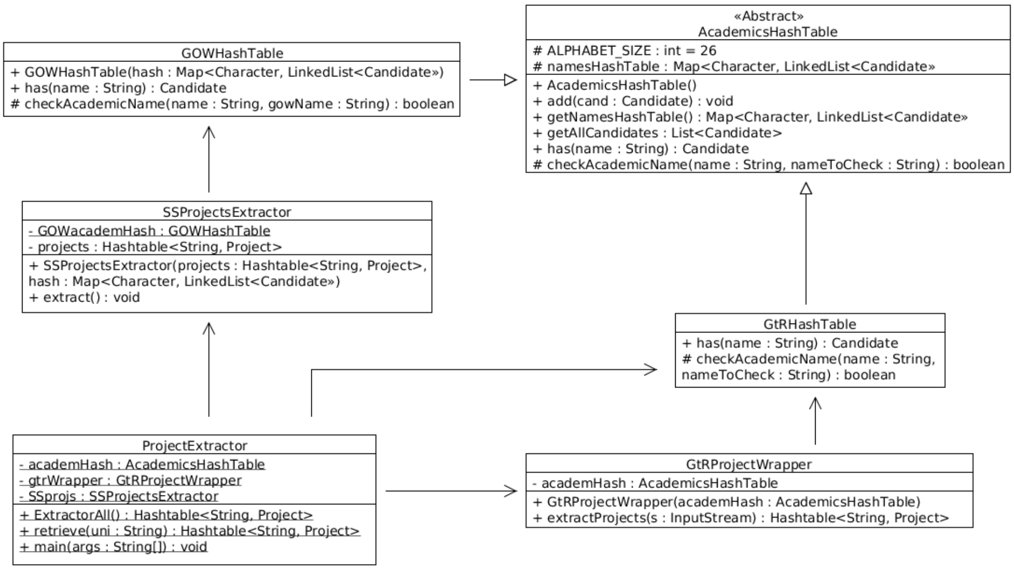


Figure 13: Projects Extraction Class Diagrams

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.

0.4.2 Data Extraction

In section 0.1.3, it was stated that funded project information are obtained from Grant on the Web [1] and Research Councils UK [6]. This section shows classes used to extract funded projects from each source and

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

Table 6: The number of funded projects extracted from each source

gives statistics regarding the number of total funded projects from each source. Figure 13 shows class diagram related to projects extraction.

AcademicsHashTable is an abstract class which makes use of HashMap data structure, Map¡Character, LinkedList¡Candida 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 from a source 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 suitable for data in Research Councils UK [6].

GOWHashTable extends from the abstract class AcademicsHashTable. Its purpose is similar to AcademicsHashTable Class but has different implementations of hash() and checkAcademicName() methods to GOWHashTable's which is suitable for Grant on the Web [1] spreadsheet.

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 [6].

ProjectExtractor is a class that makes use of SSProjectsExtractor and GtRProjectWrapper Classes to extract funded projects from both sources.

The most difficult part in extracting projects is to match the known expert's names with the expert's names in each source. This is because 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. This is why Polymorphism [4] (different implementations 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 letter of name] such as Jose, Professor JM. 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 expert 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 6 shows the number of funded projects extracted from each source. There are 1569 known candidates in the system.

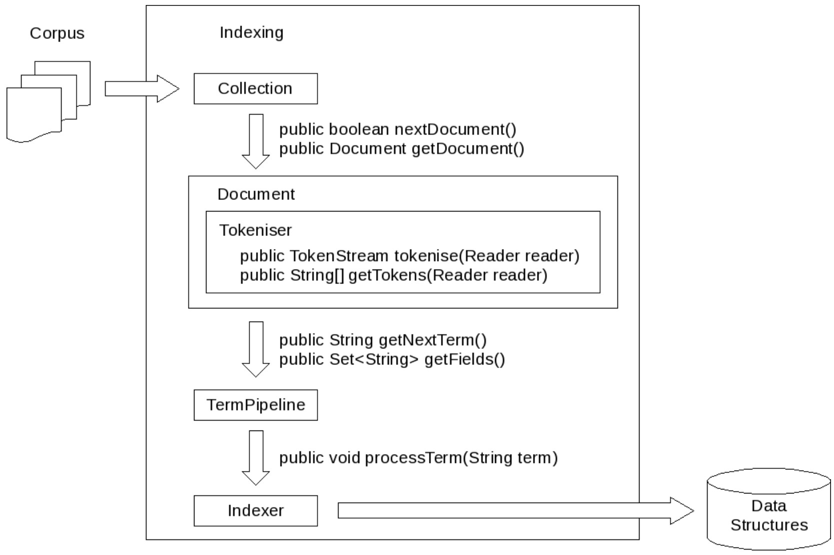


Figure 14: Indexing Architecture of Terrier from `http://terrier.org/docs/v3.5/basicComponents.html`

Indexing

In Section 0.2.1, indexing process is discussed. This process makes the retrieval process much more efficiently. In this section, steps towards indexing using Terrier [9] are discussed.

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.

I will discuss with u about this section today.

0.4.3 Retrieving Documents (Experts) with respect to a Query

Figure 15 shows the process of obtaining documents (experts) with respect to a query. First of all, the user query is transformed into publication query and project query. AcademTechQuerying Class then processes both

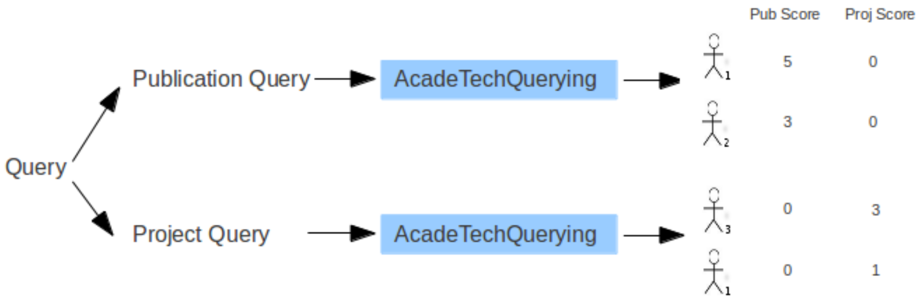


Figure 15: Querying

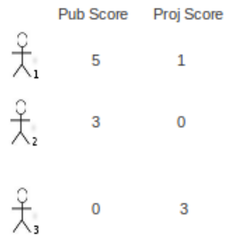


Figure 16: Results After Union

queries to get results with respect to publication query and project query. The results of both queries are then unioned as shown in Figure 16.

0.4.4 Producing a Learned Model

Section 0.2.2 described general steps to producing a learned model. This section aims to discuss classes involved in producing a learned model. As section 0.2.2 suggested, the first step in producing a learned model is to generate a set of training queries. Then all features described in Section 0.3 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. Figure 17 is a sample LETOR file. The lines preceded by hash key are ignored. They are just headers which describe features. There are 7 features as described section 0.3. The numbers after a hash key indicate features id. However, the lines not preceded by hash key are learned. They represent documents (experts) with scores of each feature attached to them. They are preceded by numbers. These numbers in learning to rank are labels which indicate the degree of relevancy. If the label is 0, it means the expert is irrelevant with respect to a query. If it is 1, the expert is relevant. However, the label needs not be binary. It could range from 0 to any positive number. The higher the number, the more relevant the expert is. In this project, labels range from 0 to 2 in order of the degree of relevancy. To the left of the label is query id and scores of each feature associated to that expert. Again, within these lines, anything after hash key is ignored.

34 training queries 7 are trained by the tool called RankLib 0.2.3 to get a learned model. This process is performed only one time.

Figure 18 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:publication_query_scores
# 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
2 qid:quantum_computation 1:1.1354922777300101E-5 2:3.2467809925657 91E-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
# 1:publication_query_scores
# 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:neural_network 1:0.0025657762336962274 2:2.1956492302765645E-4 3:0.901715 4:201.0 5:2.0 6:1.0 7:207.0 # docno = STI_00000017
0 qid:neural_network 1:0.0 2:0.0014732071662231284 3:0.00825125 4:5.0 5:1.0 6:1.0 7:7.0 # docno = STI_00000018
1 qid:neural_network 1:0.0 2:0.0033273449823246555 3:0.23846859375 4:0.0 5:2.0 6:1.0 7:0.0 # docno = STI_00000009
0 qid:neural_network 1:8.91772344534587E-5 2:0.005505326954511566 3:0.45389303515625 4:44.0 5:2.0 6:2.0 7:32.0 # docno = GST_00000052
# 1:publication_query_scores
# 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:parallel_logic_programming 1:8.772427802562802E-5 2:0.0 3:0.0 4:14.0 5:0.0 6:0.0 7:16.0 # docno = GLA_00000029
2 qid:parallel_logic_programming 1:0.0 2:4.5920546854416106E-4 3:0.0984859921875 4:29.0 5:1.0 6:1.0 7:18.0 # docno = GLA_00000028
0 qid:parallel_logic_programming 1:0.0 2:0.0019483794463139668 3:1.65500515625 4:32.0 5:4.0 6:3.0 7:33.0 # docno = GLA_00000027
2 qid:parallel_logic_programming 1:7.148613480987296E-4 2:0.0042986222877033405 3:2.583915 4:34.0 5:7.0 6:4.0 7:34.0 # docno = EDI_00000080
0 qid:parallel_logic_programming 1:0.0 2:1.2250441979824126E-4 3:0.013901 4:45.0 5:1.0 6:1.0 7:44.0 # docno = EDI_00000082
1 qid:parallel_logic_programming 1:3.025779899890064E-4 2:0.0 3:0.0 4:63.0 5:0.0 6:0.0 7:47.0 # docno = NAP_00000053
0 qid:parallel_logic_programming 1:4.347270450240863E-4 2:0.0 3:0.0 4:49.0 5:0.0 6:0.0 7:28.0 # docno = HWU_00000051
```

Figure 17: Sample LETOR file

#	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 7: Training Queries

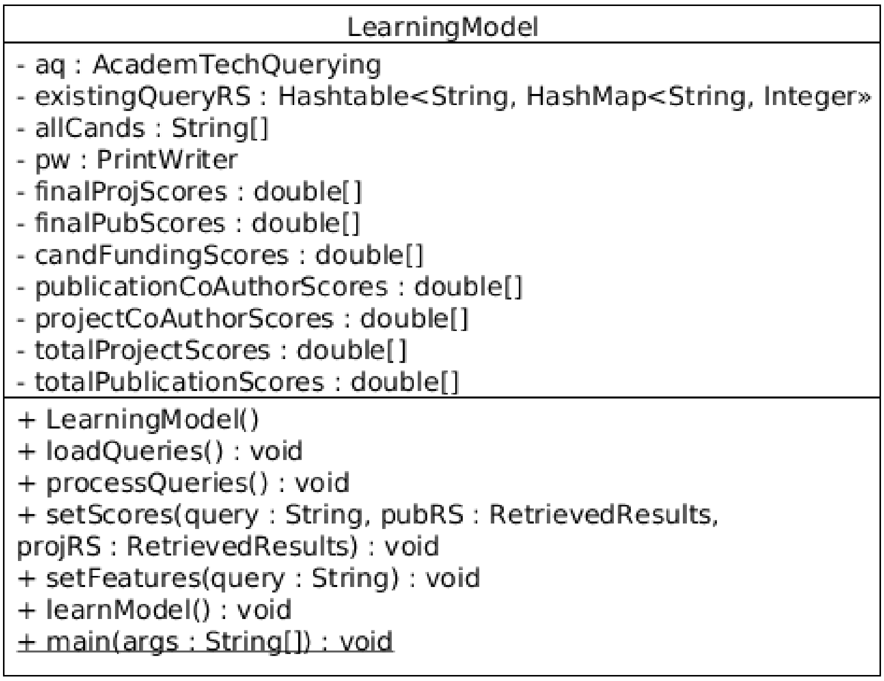
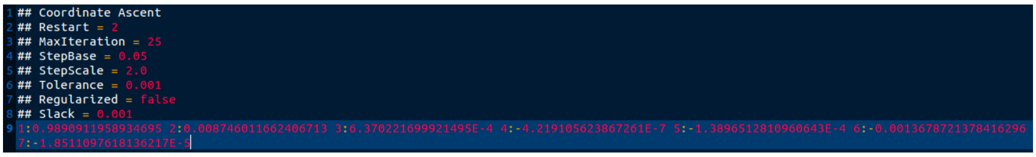


Figure 18: Class Diagram of LearningModel



```
## Coordinate Ascent
## Restart = 2
## MaxIteration = 25
## StepBase = 0.05
## StepScale = 2.0
## Tolerance = 0.001
## Regularized = false
## Slack = 0.001
1:0.9890911958934695 2:0.008746011662406713 3:6.370221699921495E-4 4:-4.219105623867261E-7 5:-1.3896512810960643E-4 6:-0.0013678721378416296
7:-1.8511097618136217E-5
```

Figure 19: Sample Model

- 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 0.4.3
- 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 0.2.3.

As stated in Section 0.2.3, 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. Figure 19 shows a sample model. Similar to LETOR file, lines preceded by hash key are ignored. They are just headers which describe parameters used in the learning to rank algorithm. Knowing what each parameter does is out of the scopre of this project. This model is obtained using Coordinate Ascent Algorithm. The numbers before colons are features id and after colons are scores of each feature. Section 0.4.5 will explain how learned model is applied to get optimal ranking.

0.4.5 Applying a Learned Model

Now that a learned model has been generated, this learned model can be applied to produce optimal ranking. Given a query, scores of query dependent features are computed for each expert as shown in figure 15. After that, scores of query independent feaures of experts obtained from querying are extracted. Then for each expert, the scores of each feature are multiplied by ones in the learned model and accumulated. Finally, the accumulated scores for each expert are sorted in descending order and experts with the high scores are ranked before those with low scores. This was briefly explained in section 0.2.2.

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] An introduction to information retrieval.

[4] Polymorphism.

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

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

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

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

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

[10] Text retrieval conference. http://trec.nist.gov/.

[11] Information retrieval system evaluation, 2009.

[12] Inverted index, 2012.

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

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

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

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

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

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

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

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

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

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

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

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

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

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