Evaluating Medical Information Retrieval Systems

Luke Henry - 14467548

Final Year Project – 2018/19
B.Sc. Computer Science and Software Engineering

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc. Computer Science and Software Engineering.

Supervisor: Dr. Liadh Kelly

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of Computer Science and Software Engineering qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Luke Henry Date: 8th March 2019

Acknowledgements

I would like to thank my project supervisor, Dr. Liadh Kelly, for providing valuable knowledge and support necessary to conduct research in this domain. I would also like to thank the University for the provision of supplementary materials and resources that made this project possible, as well as my peers for their continued encouragement and support that contributed greatly to the completion of this project.

Abstract

Medical information retrieval systems are a subset of information retrieval systems that are designed to locate and deliver relevant medical or health related material in response to some search query. These systems primarily deal with unstructured textual information in the form of documents. A relevance score is calculated and assigned for each document for a query that determines the order in which they are displayed. These relevance scores are calculated by information retrieval models, also called ranking or similarity algorithms. Common benchmark retrieval models utilized by information retrieval systems include Okapi Best Matching 25 (BM25) and the divergence from randomness (DFR) model. This is a TREC-style study based on a task proposed by CLEF eHealth in 2015. TREC (Text REtrieval Conference) refers to a series of workshops focusing on the many different information retrieval research areas. CLEF eHealth is an evaluation challenge in the domain of medical information that provides researchers with datasets, evaluation frameworks, and events, with the objective to foster research and development of medical information retrieval technologies. This study utilizes a collection of medical documents, queries and human generated relevance scores provided by CLEF eHealth to evaluate the effectiveness of information retrieval models in the domain of medical information retrieval. This was accomplished by indexing the document collection using a suitable search technology, running queries against it under many baseline information retrieval models, and using evaluation software to compare the calculated relevance scores for each document with those generated by humans. Potential strong baseline models were identified for use in this domain, and a tool and interface were developed that provide a full, automated solution for evaluating information retrieval systems, allowing for further exploration in this space.

List of Figures

List of Tables

Chapter one: Introduction

1.1 Topic addressed in this project

Information Retrieval (IR) is a concept of computer science that involves finding material, typically in the form of documents, of an unstructured nature (usually text) that satisfies an information need from within large collections of data. For effectively retrieving relevant documents by IR strategies, the documents are typically transformed into a suitable representation referred to as an index. IR models/algorithms are then used to select relevant documents from this index. This project focuses on the comparison and evaluation of IR models in relation to Medical Information Retrieval, a subset of IR that refers to the methodologies and technologies used to access medical information archives such as medical journals and medical content found on the web. The CLEF eHealth evaluation challenges [44] were created to support the development and improvement of medical information systems.

1.2 Motivation

Rapid advancements in the ever-changing medical field have led to a growing demand for access to medical information online. As our understanding of medical conditions becomes more specific and complex, health consumers often turn to search engines to support their information needs. All these users require relevant medical information and with there being copious amounts of documents online referencing their illness or condition, not all are advisable or admissible to the general user. Often varying medical knowledge and diction arises as a problem for these individuals querying the web, making it difficult to return the most relevant information which in turn restricts their ability to access the information they seek. Many medical documents uncovered by these users are often written by medical practitioners intended for other medical practitioners, and as such, laypeople are obfuscated when trying to find the medical information they seek. Some of the retrieved information may be outdated, suffer from journalistic bias, or fails to refer to primary sources of medical information, therefore proving to be unreliable. It is evident that better information retrieval systems are required to satisfy this need for relevant medical information when searching for documents online.

1.3 Problem statement

As outlined above, the collection of people querying for medical information is diverse, and with medical diction being thinly skewed to medical practitioners, this aspect of information retrieval is of foremost importance. This project aims to evaluate the effectiveness of information retrieval systems when searching for health content on the web, and to create a tool that allows for repeatable evaluation in this field. This is carried out with the objective to foster research and development of search engines tailored to health information seeking.

1.4 Approach

This project uses the CLEF eHealth 2015 collection of approximately 1.1 million medical documents sourced through the Khresmoi project [1]. CLEF eHealth also provides a set of 67 medical queries, as well as a corresponding set of qrels (query relevances), which are judgments made by humans as to whether a document is relevant to an information need. To evaluate the effectiveness of different medical information retrieval approaches/algorithms, a suitable IR technology capable of processing the documents and queries and implementing the retrieval algorithms was required. One such technology is Apache Lucene, a free and open source information retrieval library. While Lucene was a suitable candidate for this task, a relatively newer search technology, Elasticsearch [45], was selected. Elasticsearch is a free distributed web server built upon Apache Lucene that offers several advantages over the Lucene library. One such advantage is the HTTP web interface it provides which allows users to run queries from any machine without having to download and index the entire set of documents themselves. All interaction with Elasticsearch, including indexing documents and running queries, is mediated through a JSON-based REST API. This provides a standardized, flexible approach that allows users to run their own queries and evaluations independent of the underlying Java code. Elasticsearch is part of a trifecta of search tools – Logstash [22], Elasticsearch and Kibana [23], known as the Elastic stack. Logstash is a data processing pipeline that parses data before indexing it into Elasticsearch, providing additional options for parsing and indexing the document set. Kibana is a data visualization tool that provides a clean user interface for browsing Elasticsearch indices, running single queries quickly and easily, and generating various graphs and metrics from the indexed data. Kibana proved to be an invaluable debugging tool, however Logstash was ultimately not used in this project as the same functionality could be provided by Java without dependence on additional external software. Other alternatives to Elasticsearch that were considered include Apache Solr, Sphinx, and the terrier library. These were viable candidates, however the web server implementation of Elasticsearch accompanied by the strong documentation it provides made it the most suitable choice for this project.

1.5 Metrics

To evaluate the effectiveness of various IR models and ranking functions a TREC-Style evaluation is used. TREC-Style studies involve evaluation of the IR system itself. The evaluation is based on two files: The first, known as "qrels" (query relevances) lists the relevance judgements for each query. The second, known as “result files” or “runs”, contains the rankings of documents returned by the IR system under a specific retrieval algorithm or model. These files are run through TREC_EVAL [35], an industry standard IR evaluation tool that provides an analysis of the results under a set of evaluation metrics. These metrics are compared for the retrieval algorithms used with a focus on precision (P) – the fraction of retrieved documents that are relevant.

1.6 Project

This Java-based research project automates the entirety of the evaluation process described above, and provides a transparent user interface allowing a user with limited understanding of the underlying system to index, query and evaluate the dataset provided by CLEF eHealth (or their own user specified custom dataset) under many of the most significant information retrieval models, providing them with valuable comparative metrics. Noteworthy achievements to accomplish this include the deployment of a robust, distributed search engine as a RESTful web service, the ability to index a large collection of documents and run queries against it by sending JSON payloads via HTTP POST requests, the automated parsing and formatting of documents to be indexed and of the retrieved results, the automated running of the evaluation tool TREC_EVAL, and a simple user interface that mediates the above process and allows users to compare and contrast the performance of many information retrieval models against a specified data set.

From the research conducted on the document set and relevance assessments provided by CLEF eHealth, a statistical language model that employs Dirichlet smoothing to the ranking function, known as LM Dirichlet, proved to be the best performing retrieval model with considerable statistical significance. Other models such as the popular probabilistic relevance model Okapi BM25 and the relatively new divergence from independence model (DFI) [2] were strong performers and could prove valuable in the field of medical information retrieval.

Chapter two: Technical Background

2.1 Topic material

2.1.1 Information Retrieval

Information retrieval (IR) as a concept of computer science refers to the activity of obtaining resources from some information system relevant to some information need. According to Ralf Bierig [3], the four key characteristics of IR are as follows:
“1. Information is unstructured.” That is, documents and text as opposed to relational databases or lists of numbers.
“2. There are no right answers.” Queries are typically lists of words or Boolean expressions that lack clear semantics (compared to the likes of SQL queries), and a list of ‘ideal’ results are judged to be relevant as opposed to any one ‘correct’ result.
“3. Separation of indexing and query time processing.” IR processes very large collections of data that cannot be searched directly on demand. There is a requirement to separate the process into offline (i.e. indexing) time processing and online (i.e. querying) time processing.
“4. A strong empirical method is employed.” There is a need to find out whether one system performs better than another. Better systems produce more relevant information. Results must be reproducible, and evaluation is required.

The word information can be misleading in the context of information retrieval, where information, in the technical meaning given in Shannon's theory of communication [4], is not readily measured. In most cases, the data retrieved by IR systems can be described adequately by simply substituting 'document' for 'information', where documents are heterogenous and mostly textual information sources. Lancaster [5] provides an apt definition for IR in this context: 'Information retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request.’ It should be noted that information retrieval differs significantly from information extraction (IE), a principle that involves the extraction and processing of data to provide additional structure and detail, where IR is involved only in the retrieval of the data itself [6].

2.1.2 Search Engines

In the twenty-first century, the use of information retrieval systems has become widespread; end-user access to massive amounts of information in databases and on the web is routine [7]. A new name that is increasingly used to describe IR, *search*, has been popularized by IR technologies such as Google, self-described *search engines* that index the contents of the World Wide Web and attempt to deliver relevant documents in response to simple keyword-based search queries. A number of open source search engines such as Apache Lucene, Elasticsearch [45], and Sphinx have become freely available, providing transparent and easy-to-use IR software suitable for both research and enterprise use cases. These search engines employ a wide range of retrieval strategies, each strategy incorporating a specific mathematical model for its document representation.

2.1.3 Information Retrieval Models

Information retrieval models provide a basis for determining what documents are most relevant to a user's information needs. 'IR model' is a broad term used to describe mathematical and probabilistic ranking functions, often referred to as *similarity algorithms*, that are responsible for calculating relevance scores for large sets of documents. These documents are then displayed to the end-user based on the calculated relevancy score. There is need for use of stronger baselines for comparison to new retrieval models being developed by researchers [43]. This project looks at all IR models supported by the search engine Elasticsearch and evaluates their effectiveness in the context of medical information retrieval. The principle similarity algorithms assessed were Okapi BM25, the divergence from randomness framework (DFR), the divergence from independence model (DFI), the information-based model (IB), the language model with Dirichlet smoothing (LM Dirichlet similarity), and the language model with Jelinek Mercer smoothing method (LM Jelinek Mercer similarity). We are most interested in the models BM25, DFI and LM Dirichlet due to their observed high levels of precision in the retrieval of relevant medical documents. TF-IDF, often used as a weak baseline in IR systems, is included to provide a comparison to strong baseline retrieval models.

BM25

Okapi BM25[8], sometimes referred to as Best Match 25, is based on the probabilistic retrieval framework devised by S. Robertson and K. Jones in the late 1970s [9]. BM25 is considered a bag-of-words retrieval model as it is not concerned with the inter-relationship between query terms, such as their order and proximity to one another [10]. It is widely considered a standard method across the TREC (Text REtrieval Conference) community [11] and is well supported by most popular search technologies. The most prominent instantiation of the function is shown in Figure 1.

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)},$$

Figure 1 – BM25

Where $q_1,\dots,q_n$ refer to the terms composing the search query Q,
D refers to the document being scored and |D| to the length of the document,
$f(q_i,D)$ denotes the frequency of the term q_i in document D,
and *avgdl* is the average length of the document.
In the absence of advanced optimisation, k_i is chosen as 1.2 or 2 and b is typically 0.75.
IDF is the *inverse document frequency*, a numerical statistic that is intended to reflect how important a word is to a document in a collection [12].

The Okapi BM25 retrieval function tends to overly penalize very long documents, and new state-of-the-art extensions of BM25 are currently being developed in an attempt to solve this issue [13].

DFI

The DFI-based model used by Elasticsearch was developed at TREC 2012 on the basis of Shannon's information theory [14]. This term weighting model stems from the hypothesis of being an ideal retrieval function under the assumption that low term frequency corresponds to a high amount of information, and vice versa. If this assumption holds true, the increase in total information will be higher for smaller term frequencies, and hence the model assigns weights to search terms reflecting this hypothesis [2].

The proposed DFI-based function, as implemented by Elasticsearch, as shown in Figure 2.

$$\Delta(I_{ij}) = \left[(tf_{ij}+1) \times \log_2\left(\frac{(tf_{ij}+1)}{\sqrt{e_{ij}^{+}}}\right)\right] - \left[tf_{ij} \times \log_2\left(\frac{tf_{ij}}{\sqrt{e_{ij}}}\right)\right],$$

where

$$e_{ij}^{+} = \frac{(TF_i+1) \times (D_j+1)}{N+1}.$$

Figure 2 – Novel implementation of Divergence from Independence model

which calculates the increase in total information we would get by observing term *i* in document *j* one more time, given that it occurs tf_{ij} times in document *j*.

LM Dirichlet

LM Dirichlet is a language model with a Dirichlet smoothing function applied. A language model is a probability distribution over sequences of words. The language modelling approach to information retrieval models the idea that a document is a good match to a query if the document model is likely to generate the query, which will happen when the query words occur within a document at a high frequency [16]. It calculates this by assigning a probability $P(t_1,\ldots,t_n)$ to each query term *t.*

Smoothing is a form of *regularisation* for statistical language models, a technique used to address overfitting of query terms (i.e. too many matches to generalise a novel subset of documents). Smoothing is crucial in making language models useful in practical applications. Dirichlet's smoothing function (also known as Dirichlet-prior smoothing) is a form of linear interpolated smoothing that gives less weight to a document as its length increases. This preference for longer documents is highly advantageous on TREC collections such as the one utilized by this project, as discussed by M. Smucker and J. Allan in an investigation of Dirichlet smoothing's performance advantage [17].

Another well-known smoothing function is Jelinek-Mercer smoothing, however it underperformed in the evaluation of medical information retrieval investigated in this project and as such will not be discussed further in this section.

2.1.4 Evaluation Measures

This project employs a TREC-Style study, a system evaluation with a simple user model and minimal user interaction. This study lies on the left, system-focused side of the continuum of information retrieval evaluation studies proposed by Kelly, 2009 [18], shown in Figure 3.

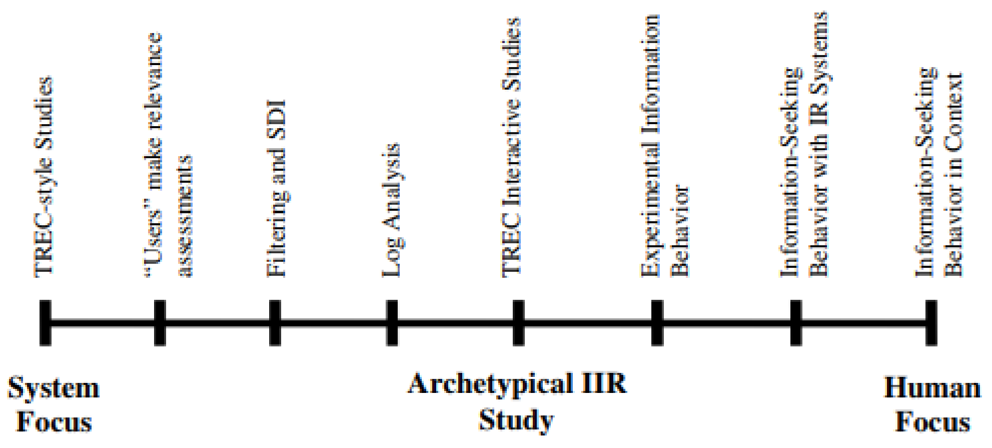

TREC-style Studies
"Users" make relevance assessments
Filtering and SDI
Log Analysis
TREC Interactive Studies
Experimental Information Behavior
Information-Seeking Behavior with IR Systems
Information-Seeking Behavior in Context
System Focus
Archetypical IIR Study
Human Focus

Figure 3 – Kelly Continuum of IR studies [18]

TREC_EVAL [35] is the standard tool used by the TREC community for evaluating an ad hoc retrieval run under NIST (National Institute of Standards and Technology) evaluation procedures. As such, TREC_EVAL provides the basis for all evaluation measures used in this project. The most important of these metrics is precision (P). TREC_EVAL provides three measures of precision: Average precision for all relevant documents, precision at intervals of documents ranging from 5 to 1000 denoted *P@N* (i.e. *precision at N documents*), and the precision at the number of relevant documents, *R*, returned for a query (*R-Precision*).

Table 1 below shows the click-through rate of the top ranked documents on Google search engine results pages based on AOL data released in 2006.

Table 1 – Search engine results page click-through data [http://www.redcardinal.ie/google/12-08-2006/clickthrough-analysis-of-aol-datatgz/]

Rank#	Click Throughs	%	Delta #n-1	Delta #n1
	19,434,540	100%		
1	8,220,278	42.30%	n/a	n/a
2	2,316,738	11.92%	-71.82%	-71.82%
3	1,640,751	8.44%	-29.46%	-80.04%
4	1,171,642	6.03%	-28.59%	-85.75%
5	943,667	4.86%	-19.46%	-88.52%
6	774,718	3.99%	-17.90%	-90.58%
7	655,914	3.37%	-15.34%	-92.95%
8	579,196	2.98%	-11.69%	-92.95%
9	549,196	2.83%	-5.18%	-93.32%
10	577.325	2.97%	-5.12%	-92.98%

This data suggests that approximately 90% of users do not look beyond the first 10 search engine results. For this reason, the metrics we are most interested in are P@5 and P@10 documents as used in the CLEF eHealth Task outlined in Section 1.1 [20].

2.2 Technical material

2.2.1 Elastic Stack

The primary external tool utilized by this project is Elasticsearch, an open source search engine built on Apache Lucene. Elasticsearch is part of the Elastic stack, a cooperative software solution developed by Elastic designed to take in data in any format, parse, search, analyse and visualize it in real time [21]. This software stack consists of three pieces of software – Logstash, Elasticsearch, and Kibana – and is typically used for data management in corporate environments, e.g. central logging systems for continuous integration environments such as Jenkins. Logstash [22] was initially considered as a data processing pipeline for indexing document sets, however a RESTful API provided by Elasticsearch was ultimately used as it allowed the task to be completed purely through Java which was already in use, reducing reliance on external software. Another part of this stack, Kibana [23], was utilized as a development tool only. It provides an interface for browsing Elasticsearch indices and running queries in real time (based on Lucene's query syntax). It also provides a development console that allowed indices to be created, deleted and moved manually, as well as providing an environment where HTTP requests could be constructed with ease and directed at Elasticsearch's REST API endpoints. This eliminated the need for temporary Java methods in the early stages of testing.

2.2.2 Elasticsearch

Elasticsearch is intended to be deployed as a web server allowing easy external access from any machine for indexing and querying. For this reason, Elasticsearch is typically accessed through HTTP requests and provides various RESTful APIs for this purpose. The primary APIs used were the index API, search API, and the reindex API.

The index API allows the use of HTTP Post requests to index new documents and HTTP Put requests to create and configure indices. Use of this API was essential as it was the basis for populating the search engine with the provided 1.1 million medical documents to be queried. Elastic provide excellent up-to-date documentation on the use of this API which proved extremely valuable [24].

The search API provides a HTTP Post endpoint for querying indices [25]. It also provides additional options for returning results, e.g. only the 'x most relevant results', which proved necessary as it would not be computationally viable to retrieve all 1.1 million documents.

The reindex API provides further functionality for indexing, leading to a large improvement in performance that will be discussed in Section 4.3 of this report [26].

All Elasticsearch APIs accept data in the JavaScript Object Notation (JSON) format, and as such a strong understanding of this format was required. The w3schools tutorials [27] on JSON were especially useful for understanding and constructing the JSON payloads required by Elasticsearch.

2.2.3 Java

It was also necessary to construct the JSON payloads mentioned above within Java. The JsonObject Java library [28] proved invaluable for this task as it provided simple methods for constructing JSON objects and traversing nodes when extracting information. It also allowed some conversion between Java objects (not to be confused with JavaScript objects which are already of JSON format) and JSON objects, reducing code complexity. This tutorial [29] on JSON in Java provided useful sample code for encoding

JSON in Java, with a focus on the construction of attribute-value pairs that are read by Elasticsearch APIs.

Other Java libraries that proved useful were the URL and URLConnection library [30], providing clear and succinct methods for connecting to URLs and creating HTTP requests. This tutorial by Yong Mook Kim [31] provided an excellent rundown on data streams, constructing HTTP headers, and collecting HTTP response codes.

The CLEF eHealth collection of documents to be indexed were compiled from web pages and as such differed greatly from the JSON format required by Elasticsearch. These documents were typically HTML web pages containing impurities including HTML tags and inline JavaScript. This data required parsing and restructuring to extract the document content in a plain text to be indexed and queried. This was done via a Java implementation of regular expressions. An online tool called regex 101 [32] served as an excellent test environment for creating regular expressions that would correctly parse these documents, and also helped with the identification of edge cases where expressions may fail. This book on regular expressions by Jan Goyvaerts [33] was beneficial in the writing of robust, well-defined regular expressions while also demonstrating various strategies for their implementation in Java.

Another challenge was the building of a graphical user interface and controller that would give end-users the freedom to run individual operations repeatedly or under different combinations of retrieval models. It was also desirable to give users the option to run evaluations under their own document collections and/or query relevance assessments. Eclipse WindowBuilder, a plugin for the Eclipse IDE, was utilized to aid in the development of this user interface. This extension combines two Java toolkits for developing user interfaces; Java SWT and Java Swing. The Eclipse foundation offers excellent video tutorials [34] that demonstrate the use of this plugin and the development of a linked controller that triggers Java operations in response to a user's interaction.

2.2.4 TREC_EVAL

The only external tool used (aside from Elasticsearch) was TREC_EVAL, a command line tool for the evaluation of IR systems. The TREC_EVAL binary is packaged with a useful README that details input parameters and evaluation metrics, also available in the official GitHub repository [35]. In order to achieve a transparent runtime environment, it was necessary to automate the running of this tool through Java. This required a good understanding of the Java Process class, a component of the Java Class Library that provides functionality for running external executable programs. This article [36] supplied clear demonstrations on the usage of this class.

Chapter three: The Problem

3.1 Summary

The problem presented in this report is that there is a strong need for improvement in the field of medical information retrieval, particularly in the domain of information retrieval models. This is discussed in a study on medical information retrieval as an instance of domain-specific search by Allan Hanbury, where he states "Due to an explosion in the amount of medical information available, search techniques are gaining importance in the medical domain" [37]. As research into information retrieval continues, new models for identifying the relevant documents to display in search engine results lists are being developed. We see these models applied in other areas, such as in speech recognition as noted by C. Zhai and J. Lafferty in this 2017 study [38]. "Language modeling approaches to information retrieval are attractive and promising because they connect the problem of retrieval with that of language model estimation, which has been studied extensively in other application areas such as speech recognition." However, these models are still under-utilized in the ranking of medical information, which typically employs the same baselines used by generic IR systems. There is a growing need for studies comparing and evaluating the effectiveness of these models to optimize the performance of these retrieval systems, and to identify benchmark retrieval algorithms that provide precise and consistent results when searching for medical information and which can be used as strong baselines for comparison in new retrieval model development.

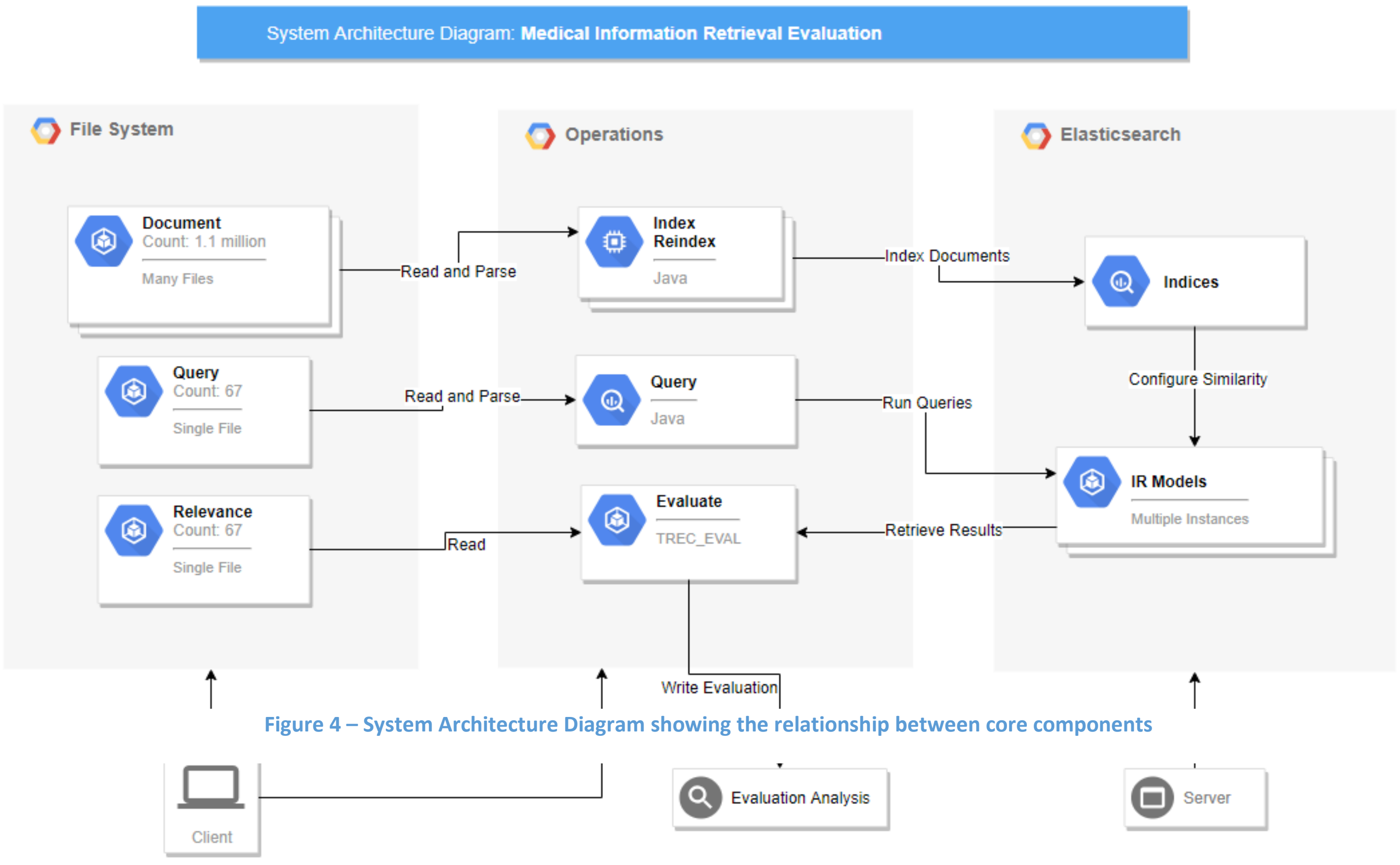


Figure 4 – System Architecture Diagram showing the relationship between core components

Figure 4 describes the architecture of the medical IR system developed and the relationship between each component. The box on the left-hand side describes the necessary files for running evaluations; the document set, queries file and qrels (relevance) file. The centre box describes the primary operations that run during an evaluation, three of which are Java operations; indexing, reindexing and querying. The fourth, 'evaluate', refers to the running of the TREC_EVAL tool to generate metrics for evaluation. The box on the right-hand side describes the Elasticsearch environment, where 'indices' are a component of search engines that contain organized data and 'IR models' are the retrieval algorithms utilized by Elasticsearch to be evaluated. The file system and operations are contained and run on some local 'client' machine, and the Elasticsearch environment is run as a HTTP web server that can be accessed from external machines. The arrows describe the relationship and interactions between the components of this system.

3.2 Problem analysis

Considering this, this project focuses on the different strong, standard retrieval models available for use, such as Okapi BM25, the divergence from randomness framework (DFR), models based on the divergence from randomness framework (DFI), the information-based model (IB), and the language modelling approaches that apply Dirichlet smoothing (LM Dirichlet) and Jelinek Mercer smoothing methods (LM Jelinek Mercer). Results of runs using a baseline IR system show that there is room for improvement in medical information retrieval, and that perhaps separation of medical IR strategies from those employed by traditional search technologies is necessary to identify new areas for improvement.

There is also the problem that full solution tools (covering the entire pipeline from indexing, to retrieval, to evaluation) didn’t exist, and if they did exist it would allow for quick and easy conducting of IR evaluations. This project provides a full solution tool for this problem, and future work can allow for the easy plugin of new retrieval models developed by researchers and investigators into this tool.

Chapter four: The Solution

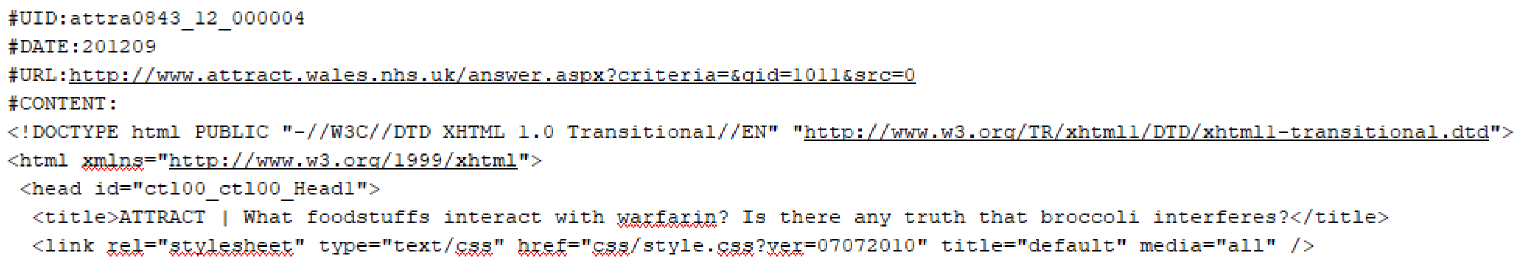
4.1 Summary

For this report, the CLEF eHealth 2015 Information Retrieval Task 2 test collections were used. This collection consists of approximately 1.1 million medical and health related documents targeted at both the general public and healthcare professionals. The documents are provided in the dataset in their raw HTML format along with uniform resource locators such as date and URL. CLEF eHealth have also composed a sample set of 67 queries based on methodology investigated by Zuccon et al. [39] and Staton et al. [40], including human generated relevance assessments for each query. The project aims to utilize these resources to evaluate the effectiveness of various retrieval models in the field of medical IR. This chapter provides an in-depth explanation of the information retrieval and evaluation techniques used.

4.2 Data Preparation

4.2.1 Document Parsing

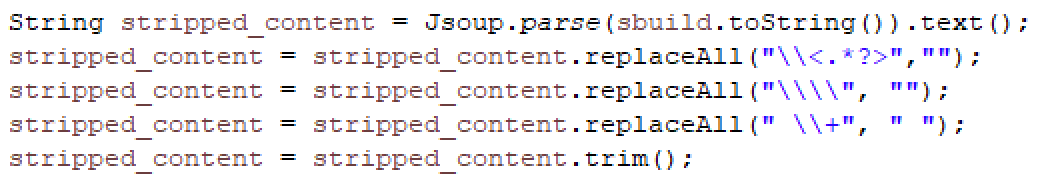
Before discussing parsing methods, it is necessary to clarify the structure of the document data in its raw form. The document collection consisting of approximately 1.1 million medical documents is contained within 1,570 DAT files (a generic format for data storage). As this document set was compiled from web pages, it contains many elements that could interfere with query results and thus the final evaluations. These elements include the HTML data that web browsers use when displaying web pages, such as content headers, formatting including paragraphs and tables, links, references to stylesheets, and inline JavaScript and PHP. The documents also contain locators represented by '#' symbols, occurring in the four lines prior to each document. Figure 5 shows the beginning lines of a document prior to parsing, including the resource locators mentioned above.



```
#UID:attra0843_12_000004
#DATE:201209
#URL:http://www.attract.wales.nhs.uk/answer.aspx?criteria=&qid=1011&src=0
#CONTENT:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head id="ctl00_ctl00_Head1">
  <title>ATTRACT | What foodstuffs interact with warfarin? Is there any truth that broccoli interferes?</title>
  <link rel="stylesheet" type="text/css" href="css/style.css?ver=07072010" title="default" media="all" />
```

Figure 5 – Beginning lines of a document prior to parsing

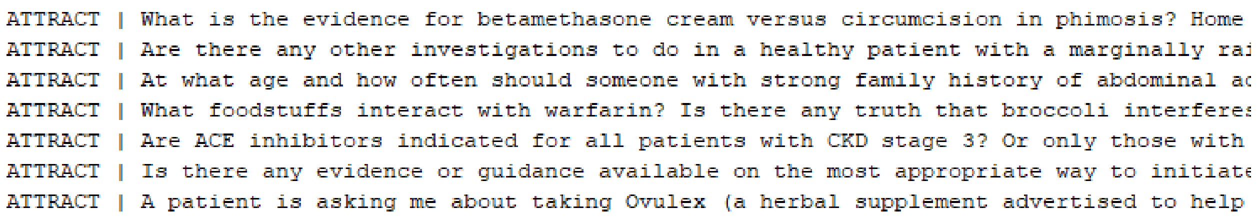
The search technology used, Elasticsearch, requires raw data to be structured into JSON format. This created an additional concern where any special symbols utilized by the JSON notation would likely break this format, causing errors during the indexing procedure. For this reason, a robust parsing solution was necessary. It was also necessary to retain the document identifier (UID) as this is required by the TREC_EVAL tool for running evaluations (detailed in Section 4.4). The document location (URL) was also extracted as it did not require additional effort to do so and could provide opportunities for future work in this area, however this was not utilized by this project. This data was contained within uniform resource locators denoted by *#URL* and *#UID*, and its extraction would require an addition parsing. This first pass of parsing was of low complexity and used simple line-by-line Java operations to identify and store this data. The second parse, with an aim to convert documents into unstructured, textual form, ultimately used a combination of regular expressions and an open source Java HTML parsing library called jsoup. The second parse and the usage of the jsoup library is shown in Figure 6.



```
String stripped_content = Jsoup.parse(sbuild.toString()).text();
stripped_content = stripped_content.replaceAll("\\<.*?>","");
stripped_content = stripped_content.replaceAll("\\\\", "");
stripped_content = stripped_content.replaceAll(" \\+", " ");
stripped_content = stripped_content.trim();
```

Figure 6 – Java implementation of second pass parsing

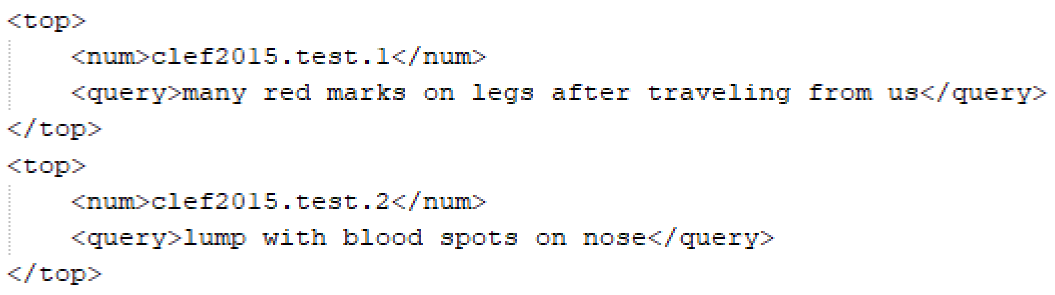
The post parse data shown in Figure 7 is now ready for indexing.



```
ATTRACT | What is the evidence for betamethasone cream versus circumcision in phimosis? Home
ATTRACT | Are there any other investigations to do in a healthy patient with a marginally rai
ATTRACT | At what age and how often should someone with strong family history of abdominal ac
ATTRACT | What foodstuffs interact with warfarin? Is there any truth that broccoli interferes
ATTRACT | Are ACE inhibitors indicated for all patients with CKD stage 3? Or only those with
ATTRACT | Is there any evidence or guidance available on the most appropriate way to initiate
ATTRACT | A patient is asking me about taking Ovulex (a herbal supplement advertised to help
```

Figure 7 – A sample of parsed document content

The second set of data that required parsing is the supplied query set. This data was provided in XML (Extensible Markup Language), another standard data format similar to JSON with a focus on human readability. Due to the well-defined structure of the query set, it proved significantly easier to parse than the collection of documents. All queries were contained within a single text file with each query separated by *query* tags, as shown in Figure 8.



```
<top>
    <num>clef2015.test.1</num>
    <query>many red marks on legs after traveling from us</query>
</top>
<top>
    <num>clef2015.test.2</num>
    <query>lump with blood spots on nose</query>
</top>
```

Figure 8 – Query data prior to parsing

Parsing the queries involved a simple Java function that would check each line for this *query* tag, remove the tags and extract and store the query contained within. As the queries were hand crafted for this purpose, they were absent from any special characters that presented an issue when parsing the document set.

4.2.2 Building JSON Payloads

A modern search technology called Elasticsearch was used as a platform for IR model evaluation. All aspects of Elasticsearch, from configuration to indexing and querying, utilize the open-standard file format JSON. It was therefore necessary to restructure the newly parsed documents into a JSON format readable by Elasticsearch. This proved extremely difficult with native Java datatypes due to the nested nature of the JSON format. The solution to this was to first construct a hashmap of Java Objects containing three pieces of data about each document: URL (location of document on the web), UID (document identifier equivalent to name of containing file) and content (the parsed document content to later be indexed and queried). Then, an external Java library *JsonObject* was used to convert each

Java Object into immutable, verifiably correct JSON objects. These new JSON objects, shown in Figure 9, were stored in Java's implementation of the *Stack* data structure until ready to be indexed.

```
{"UID":"att...","URL":"http://www.at...","content":"I have a patient exposed last week..."}
{"UID":"att...","URL":"http://www.at...","content":"Is there any published evidence on..."}
```

Figure 9 – JSON representation of document payload

A similar process was used for the construction of JSON payloads representing queries, however with an added element of complexity. The JSON format for running queries as required by Elasticsearch utilizes nested name-value pairs, where the first pair tells Elasticsearch that it is to run a query, the second pair tells it what matching function to use (e.g. Boolean matching), and the final nested pair contains the actual query string. Figure 10 shows the JSON structure of a query payload.

```
{"query":{"match":{"content":"many red marks on legs after traveling from us"}}}
{"query":{"match":{"content":"lump with blood spots on nose"}}}
{"query":{"match":{"content":"dry red and scaly feet in children"}}}
```

Figure 10 – JSON representation of query payload

Creating this payload required the construction of three JSON objects, *content* (the parsed query string), *match* (the matching function to be used), and *query* (the keyword that tells Elasticsearch to run as a query). The *content* object was assigned as the value of the second object, *match.* The *match* object, containing the nested *content* object, was then assigned as the value of the final *query* object, creating the desired nested structure. This implementation is shown in Figure 11.

```java
JSONObject query = new JSONObject();
JSONObject match = new JSONObject();
JSONObject content = new JSONObject();
content.put("content", queryset.pop());
match.put("match", content);
query.put("query", match);
jsonqueries.add(query);
```

Figure 11 – Java implementation of JSON query structure

4.3 Indexing

Elasticsearch, as a web-based search engine, utilizes numerous RESTful APIs (application programming interfaces that accept HTTP requests such as GET, PUT, POST, and DELETE) for communication. This is excellent for providing distributed access to many machines and as such it is often used for data management in corporate environments. For this study, Elasticsearch offers the advantage of being searchable from any machine without requiring a copy of the initial document set and without needing to create and populate a new index (which can be rather time consuming). This makes it an ideal framework for group contribution, as well as for conducting future work in this area of research. There are some downsides to this, however. One such downside is that it is highly dependant on network availability. There is also a slight hit to performance compared to locally stored indices implemented by technologies such as Lucene.

Two Elasticsearch APIs are used in this project; the *index API* and the *reindex API.* The index API is used for the initial indexing of the document collection. This is very time consuming (taking approximately 26 hours for 1.1 million documents). The reindex API tells Elasticsearch to copy the contents of an existing index to a new index, empty or otherwise. This function takes approximately 33 minutes to copy the 1.1 million documents in the existing index; a large improvement over the 26 hours taken otherwise. The reindex API was a necessary workaround for reasons explained in Section 4.3.2 of this study.

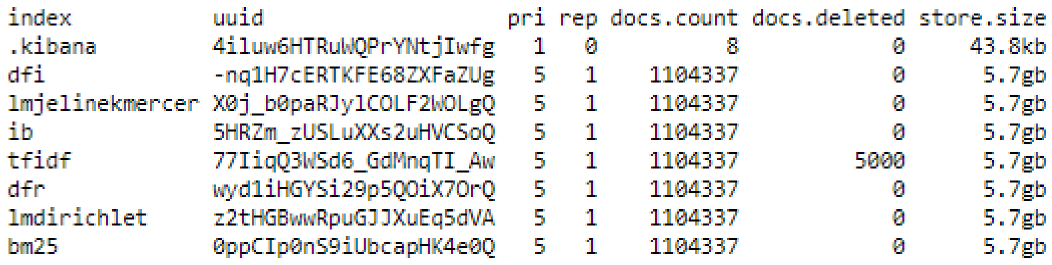
4.3.1 Initial Indexing

To connect to the Elasticsearch index API, a standard Java library *URLConnection* was used. This library provides functionality allowing a connection to the index API endpoint of our Elasticsearch deployment. It provides a range of useful methods that simplify the process of creating HTTP headers, setting request properties such as *Content-Type*, and triggering the various HTTP request methods. In this case HTTP post requests are used to send the JSON payloads constructed in the previous section. It was also necessary to indicate the media type of the resource being sent (*application/json*) and the character set in use (*UTF-8*). Utilizing the URLConnection library, this task was a simple one-line operation (line 3 in Figure 11 below).

```
URLConnection connection = new URL(url).openConnection();
connection.setDoOutput(true); //Triggers HTTP POST by default
connection.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
```

Figure 12 – Java implementation of *URLConnection* library

Figure 13 shows a snapshot of the resulting indices as provided by Elasticsearch's web interface. The left-hand side lists each index, named for their respective IR model. The number of documents is listed under the *docs.count* heading, and is the same for each index as they all utilize the same set of documents. Kibana, detailed in Section 5.1, stores its objects as documents in the .kibana index – this index is not part of the evaluation procedure.



```
index            uuid                   pri rep docs.count docs.deleted store.size
.kibana          4iluw6HTRuWQPrYNtjIwfg   1   0          8            0     43.8kb
dfi              -nq1H7cERTKFE68ZXFaZUg   5   1    1104337            0      5.7gb
lmjelinekmercer X0j_b0paRJy1COLF2WOLgQ   5   1    1104337            0      5.7gb
ib               5HRZm_zUSLuXXs2uHVCSoQ   5   1    1104337            0      5.7gb
tfidf            77IiqQ3WSd6_GdMnqTI_Aw   5   1    1104337         5000      5.7gb
dfr              wyd1iHGYSi29p5QOiX7OrQ   5   1    1104337            0      5.7gb
lmdirichlet      z2tHGBwwRpuGJJXuEq5dVA   5   1    1104337            0      5.7gb
bm25             0ppCIp0nS9iUbcapHK4e0Q   5   1    1104337            0      5.7gb
```

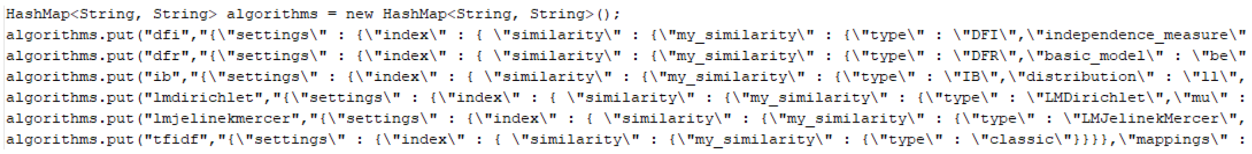
Figure 13 – Snapshot of /_cat/indices?v from Elasticsearch web interface

4.3.2 Reindexing

A core weakness of Elasticsearch that necessitates the use of the reindex API is its inability to change *similarity modules* (term describing IR models used by Elasticsearch). This means that in order to run queries under a different IR model, it is necessary to reindex the entirety of the document set. This was a major setback to this study due to the long runtime required to index the document set (26 hours). Given that the aim of the project is to evaluate the effectiveness of several IR models, it would be impractical to wait 26 hours for each evaluation. As Elasticsearch is such a new technology, it is highly likely (and suggested as a future feature in the official Elasticsearch documentation) that support will be added allowing the change of similarity mappings for existing fields, however for the purposes of this project a workaround was required. This workaround was the reindex API.

The reindex API required the construction of two new, reasonably simple JSON payloads. The requirements of the reindex API are somewhat static, and as such it was reasonable to encode this JSON data by hand. The Java String representation of this data for each similarity algorithm is shown in Figure

14. The indices are created and named after the similarity algorithm configured for use, allowing easy recognition when running queries.



```
HashMap<String, String> algorithms = new HashMap<String, String>();
algorithms.put("dfi","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"DFI\",\"independence_measure\"
algorithms.put("dfr","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"DFR\",\"basic_model\" : \"be\"
algorithms.put("ib","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"IB\",\"distribution\" : \"ll\",
algorithms.put("lmdirichlet","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"LMDirichlet\",\"mu\" :
algorithms.put("lmjelinekmercer","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"LMJelinekMercer\",
algorithms.put("tfidf","{\"settings\" : {\"index\" : { \"similarity\" : {\"my_similarity\" : {\"type\" : \"classic\"}}}},\"mappings\" :
```

Figure 14 – Java Implementation of manual JSON payload construction

An additional complexity of the reindex API is that one such payload, used for creating new indices with a different IR model configuration, required the use of *HTTP put* instead of the post method employed elsewhere. This was relatively simple to implement with the *URLConnection* library mentioned above.

The creation of new (empty) indices was near instant, however the reindexing to populate these indices took approximately 33 minutes. This was the best compromise available for changing IR models, with a significant time improvement from the 26-hour run time of the index API.

4.4 Querying

4.4.1 Running Queries

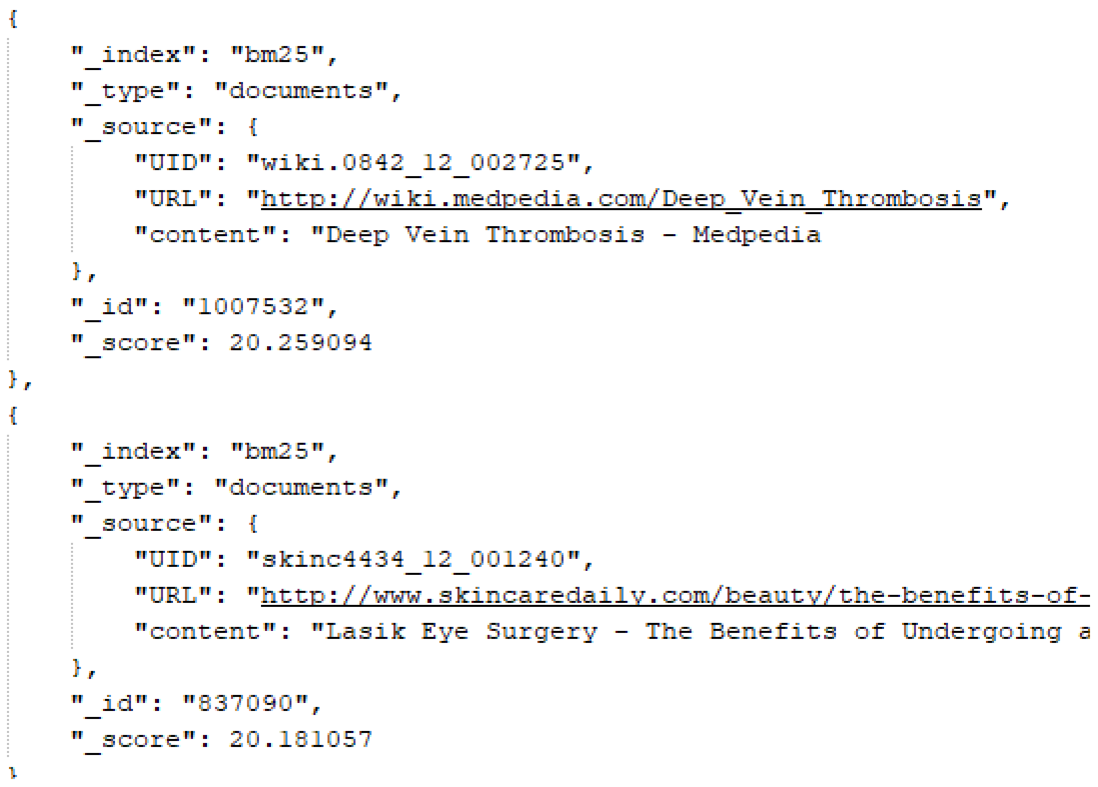
Running the parsed and formatted set of queries again utilizes the *URLConnection* library to send *HTTP post* requests containing JSON payloads. Running queries involves the use of Elasticsearch's *search* API. To access this API, a connection must be made to the appropriate index endpoint followed by the keyword */_search.* The search API is extremely versatile, allowing the running of simple queries through *HTTP get* (the default request made by web browsers) as well as more complicated queries through *HTTP post*. It also supports advanced options such as *size* and multiple attribute searches. An example of a query run via *HTTP get* request is shown below, with a size of 200 (returns the 200 most relevant results). 'Localhost:9200' describes the Elasticsearch host and port number, and 'bm25' is the name of the index being queried. The query being run in this case is "red skin bruise".

http://localhost:9200/bm25/_search?q=content:red+skin+bruise&size=200

URL encoded queries were unsuitable for this project due to the length of some queries and the inclusion of special characters such as question marks, however the implementation of the *post* request is very similar.

4.4.2 Parsing and Formatting Response

When running queries against an Elasticsearch index, we are most interested in the response object returned by the search API. This JSON response contains a plethora of information; time taken in milliseconds, total number of relevant documents, rank and relevancy score for each document, and of course the document content itself (including the URL and UID fields mentioned in Section 4.1). A snapshot of the response object is shown by Figure 15.



```
{
    "_index": "bm25",
    "_type": "documents",
    "_source": {
        "UID": "wiki.0842_12_002725",
        "URL": "http://wiki.medpedia.com/Deep_Vein_Thrombosis",
        "content": "Deep Vein Thrombosis - Medpedia
    },
    "_id": "1007532",
    "_score": 20.259094
},
{
    "_index": "bm25",
    "_type": "documents",
    "_source": {
        "UID": "skinc4434_12_001240",
        "URL": "http://www.skincaredaily.com/beauty/the-benefits-of-
        "content": "Lasik Eye Surgery - The Benefits of Undergoing a
    },
    "_id": "837090",
    "_score": 20.181057
}
```

Figure 15 – Elasticsearch JSON response object

To evaluate these query results we must extract three pieces of information – the relevancy score calculated by the IR model in use, the document ranking (i.e. the position in which the document is placed on a search engine results page), and the document UID (an identifier corresponding to the document source file and position). Extracting these metrics involves the use of the *JsonObject* Java library to traverse the nested JSON results, identifying and isolating them individually and storing them for later formatting. The line of code used to extract the relevancy score is given in Figure 16. The score is parsed from its String representation to the Java primitive *double* type, which would allow for the use of additional mathematical operations in the future that are out of the scope of this project.

```
score = (double)response.getJSONObject("hits").getJSONArray("hits").getJSONObject(q).get("_score");
```

Figure 16 – Extraction of relevance scores from JSON response object

The standard tool for TREC-style studies, TREC_EVAL, will be used to evaluate the query results for each IR model supported by Elasticsearch. The evaluation metrics provided by TREC_EVAL will be discussed in greater detail in Chapter 5 of this project. TREC_EVAL requires a very specific format to run, and as such the parsed out query results must be transformed to satisfy this format. This TREC_EVAL readable format looks as shown in Figure 17.

query-number   Q0   document-id   rank   score   Exp

Figure 17 – Format required by TREC_EVAL

One search engine result is represented by these six parameters, each separated by any whitespace character. Each search engine result appears on its own line. *Query-number* identifies the query being run. *Document-id* is the external ID for the retrieved document, i.e. the UID. *Score* is the relevance calculated by the retrieval system under a particular IR model for that document against the respective query. *Q0* (Q zero) and *Exp* are constants that are used by some evaluation software.

As TREC_EVAL is an external executable tool, it is necessary to create a file containing these expressions for each IR model used. Java's *Files* class was used to create and write these files. Each file is given a name related to the applied retrieval model for identification by TREC_EVAL. It is also necessary to ensure that the correct character set, *UTF-8*, is in use. This usage of the *Files* library and its containing methods is shown in Figure 18 below.

```
try (Writer writer = new BufferedWriter(
        new OutputStreamWriter(
                new FileOutputStream("G:\\Project\\test\\" + model + ".txt", true), "utf-8"))) {
```

Figure 18 – Implementation of Java *Files* library

The construction of the extracted query results, particularly query number, document id, rank, and score, into the necessary TREC_EVAL format is done on a per-document basis. Each line is created as detailed in Figure 17 above and is separated by a newline character. The Java implementation of this is shown in Figure 19.

```
trec_format = query_number + "\t" + q0 + "\t" + document_id + " " + rank + " " + score + " Exp\n";
```

Figure 19

and the final formatted results file is shown in Figure 20.

```
1    0    babyc2035_12_000647 199 16.261417 Exp
1    0    natio3787_12_000643 200 16.259155 Exp
2    0    mayoc3579_12_004926 1 20.239843 Exp
2    0    mayoc3579_12_017673 2 20.074932 Exp
2    0    stead4562_12_000853 3 19.137026 Exp
```

Figure 20 – Results in TREC_EVAL format

4.5 Running TREC_EVAL

TREC_EVAL is typically run as a command line tool. It accepts two parameters; the formatted query results file and the qrels (human generated query relevances) file provided by CLEF eHealth, as well as optional advanced parameters for more specific use cases. For the sake of transparency, it would be beneficial to shield end users from the inner workings of this tool. This principle will be discussed in further detail in the User Interface section, Section 4.6, of this report.

The best option for achieving transparency is to automate the running of this tool. The *Java.lang.Process* class was used for this task as it provides useful functionality for the running of external executable programs. This class requires three parameters. The first parameter is a String representation of the command to be run, which is identical to the execution at command line. The second parameter is a String array of environment variables; however, system variables are satisfactory for this use-case so this parameter will be set to *null.* The final parameter sets the working directory, which should point to the directory containing the TREC_EVAL binary. The Java implementation is shown in Figure 20 below.

```
Runtime.getRuntime().exec("G:\\...\\TREC_EVAL.exe \"G:\\...\\qrels.txt\" \"G:\\...\\"
        + selectedModels[q] + ".txt"", null, new File("G:\\...\\trec_directory"));
        // selectedModels[q] is the name of the retrieval model to be evaluated
```

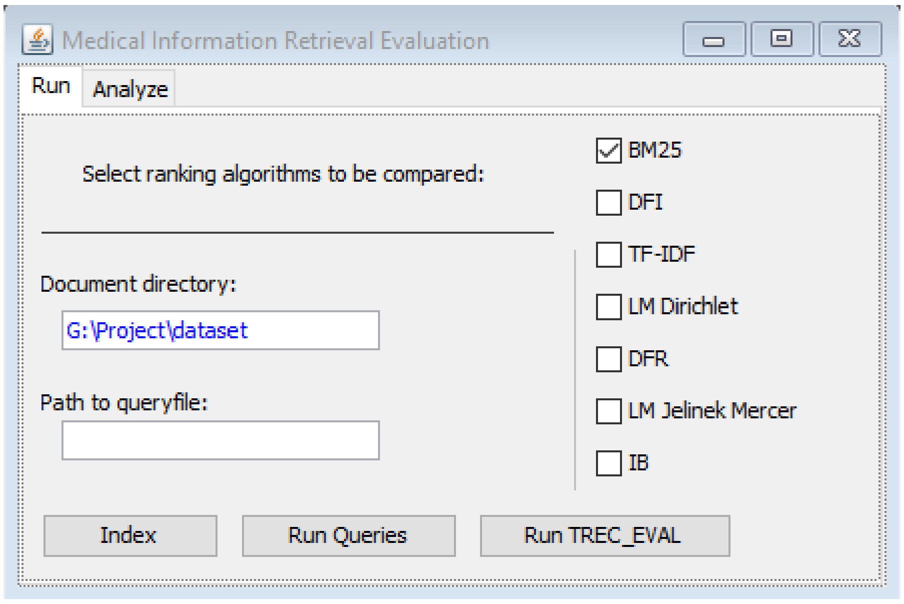
Figure 21 – Automation of TREC_EVAL

The TREC_EVAL output must also be written to files for further comparison and analysis. The same strategy as was used to write the formatted results file was utilized here.

4.6 User Interface

Due to the complexity of this project it was highly desirable to provide a simple user environment. A user interface was created to achieve this. This interface was created using *WindowBuilder*, an extension for the Eclipse development environment. It combines the functionality of two Java-based user interface designers; *Java Swing* and *Java SWT*, providing a convenient drag-and-drop visual designer that automatically generates Java code for the rendering of various user interface elements. This extension makes use of two Java class files. A *UI* class, containing the generated Java representation of the user interface, and a *controller* class where functionality is written for interactive elements such as buttons, tabs and checkboxes.

The user interface solves many of the confusions that may arise for end-users when attempting to run their own evaluations. For example, without some understanding of the underlying code it may be difficult for users to know which Java class performs what function. A minimum of three functions must be run to complete a single evaluation; the index function, the query function, and the TREC_EVAL function. The reindex function adds a further layer of complexity due to its unintuitive nature. The user may also need to be aware of the locations of the various files that are written throughout the process, such as those generated by TREC_EVAL. It would also be necessary to run each function several times to generate the evaluations required to make any meaningful analyses. The user interface, pictured in Figure 22, attempts to solve these issues.


Medical Information Retrieval Evaluation

Run | Analyze

Select ranking algorithms to be compared:

- [✓] BM25
- [ ] DFI
- [ ] TF-IDF
- [ ] LM Dirichlet
- [ ] DFR
- [ ] LM Jelinek Mercer
- [ ] IB

Document directory: G:\Project\dataset

Path to queryfile:

Figure 22 – User Interface

As can be seen in Figure 22 above, the user interface provides three primary functions – the ability to run the index API, the ability to run a set of queries, and the ability to run the TREC_EVAL executable. It also provides a simple analysis tab that loads the output files of TREC_EVAL runs for easy comparison. Finally, it provides selectors for the ranking algorithms (i.e. information retrieval models) supported by Elasticsearch. Multiple retrieval models can be selected at one time, and these functions will be run iteratively for each selected retrieval model, bypassing the need for multiple manual runs.

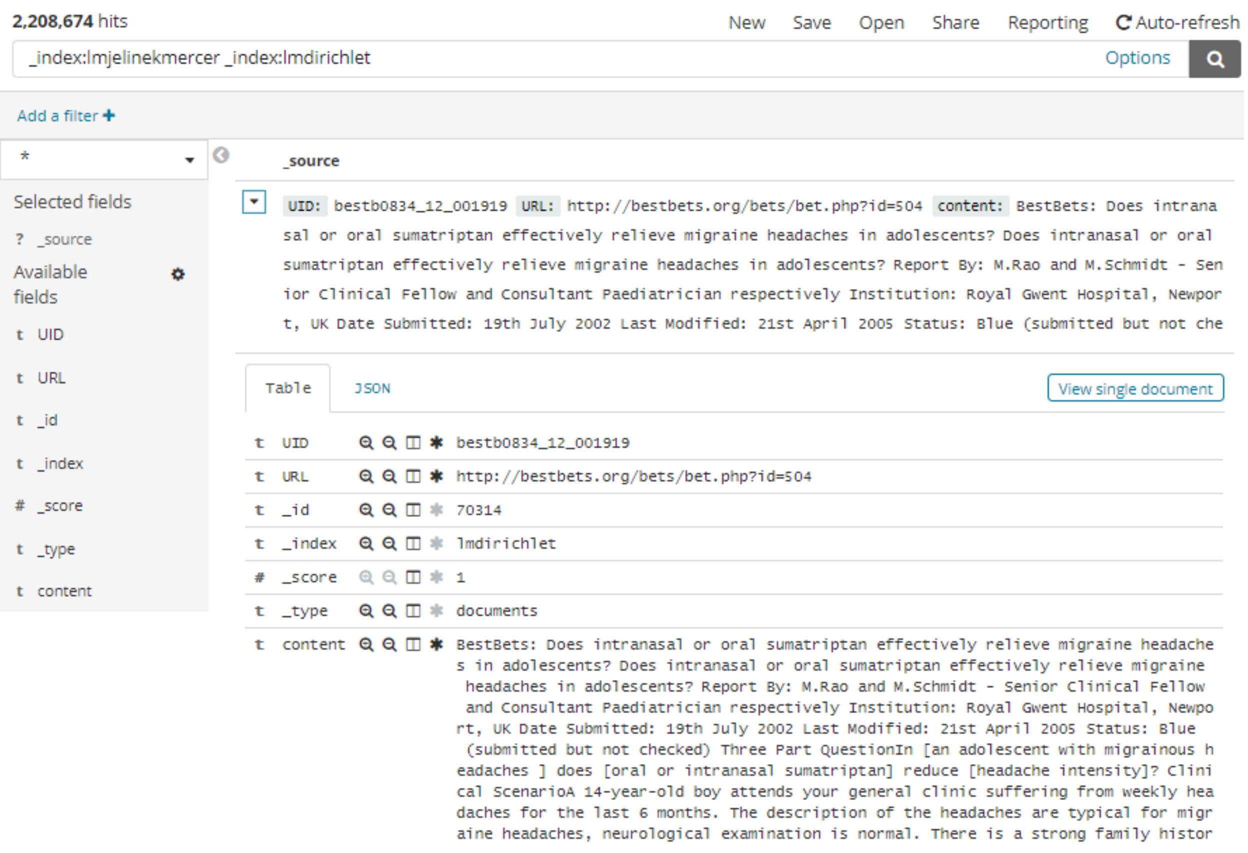
The user interface also employs the software principle of transparency by hiding the added complexity of the reindex function described in Section 4.2. Transparency in the context of computing systems somewhat counterintuitively refers to the disguising of complex systems that end-users do not require knowledge of [41]. This UI creates a layer of abstraction by combining the index and reindex functions under one more easily understood *Index* function. It achieves this by looking for changes in the document directory since the previous index. If no changes are found, it must be the case that the user intends to reindex the same document collection under a different IR model. In this case, the much more time efficient reindex function will be run, saving the user over 25 hours of waiting and requiring no additional understanding of the underlying system.

Chapter five: Evaluation

5.1 Validation and Testing

5.1.1 Index Analysis

Once index creation was completed, an analytics and visualization platform developed alongside Elasticsearch called Kibana was used. This tool provides a user interface that allows the browsing and querying of Elasticsearch indices. Kibana queries circumnavigate the typical JSON query method utilized in this project. It instead allows the use of Lucene query syntax [42] to quickly and easily query one or more Elasticsearch indices, and returns a human readable visual representation of the data within. This proved to be a valuable development tool for validation of temporary indices created during the initial proof-of-concept testing of Elasticsearch. This tool was later used to ensure that the first indexing of the CLEF eHealth document collection had completed without error or corruption, as well as to ensure that no special characters or unintended HTML content had been indexed.



2,208,674 hits

_index:lmjelinekmercer _index:lmdirichlet

_source

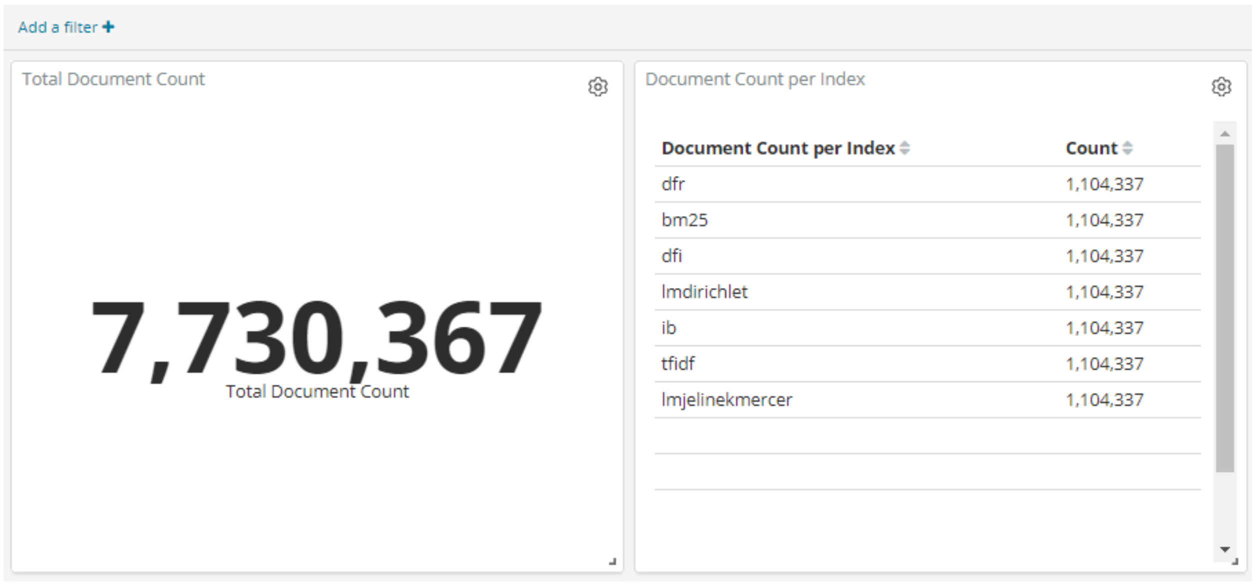
UID: bestb0834_12_001919 URL: http://bestbets.org/bets/bet.php?id=504 content: BestBets: Does intranasal or oral sumatriptan effectively relieve migraine headaches in adolescents? Does intranasal or oral sumatriptan effectively relieve migraine headaches in adolescents? Report By: M.Rao and M.Schmidt - Senior Clinical Fellow and Consultant Paediatrician respectively Institution: Royal Gwent Hospital, Newport, UK Date Submitted: 19th July 2002 Last Modified: 21st April 2005 Status: Blue (submitted but not che

t	UID	bestb0834_12_001919
t	URL	http://bestbets.org/bets/bet.php?id=504
t	_id	70314
t	_index	lmdirichlet
#	_score	1
t	_type	documents
t	content	BestBets: Does intranasal or oral sumatriptan effectively relieve migraine headaches in adolescents? Does intranasal or oral sumatriptan effectively relieve migraine headaches in adolescents? Report By: M.Rao and M.Schmidt - Senior Clinical Fellow and Consultant Paediatrician respectively Institution: Royal Gwent Hospital, Newport, UK Date Submitted: 19th July 2002 Last Modified: 21st April 2005 Status: Blue (submitted but not checked) Three Part QuestionIn [an adolescent with migrainous headaches] does [oral or intranasal sumatriptan] reduce [headache intensity]? Clinical ScenarioA 14-year-old boy attends your general clinic suffering from weekly headaches for the last 6 months. The description of the headaches are typical for migraine headaches, neurological examination is normal. There is a strong family histor

Figure 23 – Kibana *Discover* tool

Figure 23 demonstrates the abilities of the Kibana *discover* tool. The Lucene query run is *_index:lmjelinekmercer _index:lmdirichlet.* This query selects all documents in the LMJelinekMercer and LMDirichlet indices. The hits number in the top left-hand corner indicates that all 1.1 million documents are present in each index, hence the combined total of 2.2 million. It also demonstrates the successful indexing of the UID and URL fields and provides the ability to browse the document content within the *content* field.

Kibana also allows the creation of self-updating visualizations that respond to changes in Elasticsearch indices. The image in Figure 24 below demonstrates a simple dashboard that tracks total document count and document counts for each index.

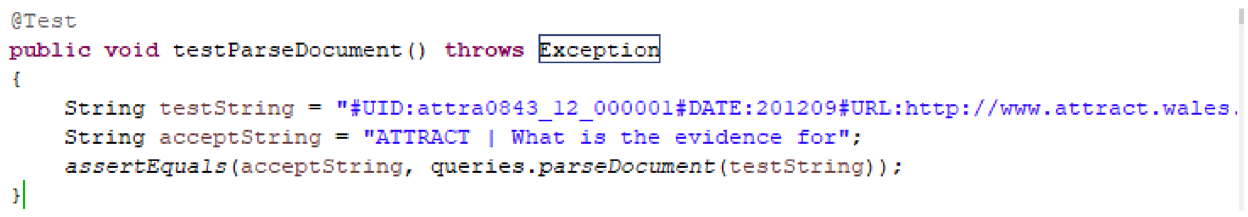


Document Count per Index	Count
dfr	1,104,337
bm25	1,104,337
dfi	1,104,337
lmdirichlet	1,104,337
ib	1,104,337
tfidf	1,104,337
lmjelinekmercer	1,104,337

Figure 24 – Kibana visualizations dashboard

5.1.2 Unit Testing

Unit testing is a form of software testing where small functional units or components are individually tested. Simple unit tests were written for the various parsing functions used throughout this project, as well as for all functions that involve the construction of JSON payloads. These unit tests serve to ensure that all JSON objects are valid, and that parsing functions successfully remove all problem characters. One such unit test is shown in Figure 25.



```
@Test
public void testParseDocument() throws Exception
{
    String testString = "#UID:attra0843_12_000001#DATE:201209#URL:http://www.attract.wales.
    String acceptString = "ATTRACT | What is the evidence for";
    assertEquals(acceptString, queries.parseDocument(testString));
}
```

Figure 25 – Unit test implementation

5.2 Research Evaluation

TREC_EVAL was used to evaluate the human-generated relevancy scores provided by CLEF eHealth with the algorithmically generated scores calculated by the various similarity modules supported by Elasticsearch. This section provides analysis of the results obtained from the TREC_EVAL tool for each information retrieval model, and insight into the implications of these results in the context of medical information retrieval.

Table 2 – Compilation of evaluation results (the best results are highlighted in bold)

Retrieval Model	P@5	P@10	Average Precision	R-Precision
BM25	0.3108	0.2677	0.1475	0.1847
DFI	0.3262	0.2692	0.1870	0.2107
DFR	0.0585	0.0600	0.0278	0.0449
IB	0.1508	0.1538	0.0930	0.1167
LMDirichlet	**0.3354**	**0.3277**	**0.1961**	**0.2356**
LMJelinekMercer	0.2769	0.2538	0.1440	0.1892
TF-IDF	0.1385	0.1323	0.0651	0.1062

Table 2 provides a rundown of the performance of each IR model under four metrics. *P@5* (precision at 5) refers to the precision (i.e. the percentage of retrieved documents that are relevant) after 5 documents have been retrieved. Likewise, *P@10* refers to the precision of the first 10 documents. *Average Precision* provides a non-interpolated average recall over all relevant documents. *R-Precision* measures precision after R documents have been retrieved, where R is the total number of relevant documents for a query. Note that the TF-IDF model is shown for comparison as it is a weak baseline.

From this data, four standout models can be identified; BM25, DFI, LMDirichlet, and LMJelinekMercer. Of the language-based models LMDirichlet and LMJelinekMercer, Dirichlet smoothing outperforms the Jelinek-Mercer smoothing function in all metrics.

Okapi BM25, as a baseline matching function in many IR retrieval systems and the default algorithm utilized by Elasticsearch, performs well as expected.

The novel implementation of divergence from independence (DFI) described in Section 2.1.3 of this report performed surprisingly well at this task. There was a significant improvement in precision at 5 documents over BM25, and a negligible improvement in precision at 10 documents. However, DFI managed to retrieve substantially more relevant documents than BM25, hence the jump in average precision. While this metric is not particularly relevant for search engine results pages where there is a focus on precision of the first 5 to 10 results, it could be suitable for other use cases where a high relevant retrieval rate is desirable.

Overall, the language-based model of Dirichlet smoothing has proven to be most effective for the retrieval of medical information, with a significant improvement in all measurements over the next best model. For this reason, there is a strong case for the use of the LM Dirichlet algorithm as a baseline in the medical domain of information retrieval.

Chapter six: Conclusion

6.1 Summary

The aim of this study was to investigate the effectiveness of the various information retrieval models typically utilized in generic retrieval systems with a focus on the subdomain of medical information retrieval. The effectiveness of seven retrieval models was evaluated under four metrics. This chapter discusses the implications of the results obtained from this investigation.

6.2 Contribution to the state-of-the-art

Due to a growing need for medical and health related search applications, search techniques are gaining importance in this field. The evaluations conducted in this study have identified the relative utility of different IR models when attempting to retrieve relevant medical information. This study contributes to the state-of-the-art of information retrieval by demonstrating the strongest baseline retrieval models in the domain, with strong supporting evidence for one such model – LM Dirichlet.

The other big contribution is the generation of a tool which allows for rapid, repeated evaluation in this domain.

6.3 Results discussion

These results are potentially generalizable to the field of medical IR in general due to the large set of medical documents used. However, there are several weaknesses that imply a need for further study. One such weakness is the size of the query set utilized. This query set consists of 67 unique health-related queries, and while a finite number of human-generated relevance assessments were made for each query, the query set itself is a rather small representation of the immense requirement for medical and health related information. Another flaw with this query set is the lack of medical terminology utilized. The queries are typical of those made by laypeople with minimal medical knowledge, and hence the results may not be as applicable for retrieval systems intended for medical professionals. Future studies could look to repeating these experiments on the other years of the CLEF eHealth IR challenges (the queries for these challenges are similarly targeted to laypeople).

It should be noted that obtaining large sets of relevance assessments is extremely laborious, and limitations associated with qrels (results) files generated based on finite relevance assessments are well acknowledged in the IR community. Further relevance assessment although not impossible would likely be expensive and time consuming. The query relevances provided by CLEF eHealth are comprehensive enough to obtain meaningful data.

Despite these weaknesses, LM Dirichlet, a language-based retrieval model that utilizes the Dirichlet Bayesian smoothing algorithm, would likely function as a stronger baseline for medical information retrieval than those typically employed by medical IR retrieval model development research. This report provides strong evidence for this, especially in systems used by laypeople with minimal medical knowledge due to the structure of the assessed queries provided by CLEF eHealth.

6.4 Project Approach

The scalable approach taken by this project attempts to pave the way for future study. This is achieved by the optimization of three principles; ease of use, ease of collaboration, and adaptability.

6.4.1 Ease of Use

This project employs the use of a simple user interface that hides underlying functionality, allowing users to index documents, run queries, and make evaluations without requiring any knowledge of the processes behind these operations. The only user operations required are running Elasticsearch and specifying the locations of their data set and TREC_EVAL executable. All other operations are mediated through the provided user interface.

6.4.2 Ease of Collaboration

Use of the web-based search technology Elasticsearch allows groups of individuals to query and evaluate documents in a shared space. This means that a document set indexed by a single user can be accessed and queried by any number of users from their own machines, supporting rapid changes and/or the use of multiple query sets. As all interaction with Elasticsearch is resolved through simple HTTP requests, it is easy for users to create functionality to support their own use cases. For example, users can make use of Elasticsearch's *update* API allowing the modification of already indexed documents.

Kibana, a tool discussed in the validation section, Section 5.1, of this report can also be deployed to a network location allowing users to access and share real time visualizations of their indexed data. This acts as an excellent management tool for group collaborations.

6.4.3 Adaptability

This project also allows users to input their own custom document collections, query sets and query relevances. This framework is therefore suitable for further research into the area of medical information retrieval, but can also be adapted for the investigation of any other subdomains of IR.

6.5 Future Work

Reflecting on the achievements of this project, there are some considerations for future work that were not possible under the given time allowance. One such possibility for future work would be to provide analysis based on metrics not produced by TREC_EVAL. One such metric that could improve the quality of results is *Normalized Discounted Cumulative Gain* (NDCG), a measure of ranking quality sometimes used in the evaluation of IR systems. This metric attempts to compare a search engine's performance from one query to the next, potentially providing additional insight into the evaluations of precision calculated in this study. Unfortunately this metric is not supported by the Windows version of TREC_EVAL used in this project, and supporting it would require the addition of external software such as Cygwin to emulate a Linux command line environment. As this project attempts to avoid the use of external tools to simplify usage and adaptation by end-users, this complexity would be undesirable. It is possible that future versions of the TREC_EVAL tool will support NDCG natively on Windows systems creating an opportunity for future work.

Another area for future investigation could be to expand the information retrieval models evaluated. As Elasticsearch is relatively new, it supports a somewhat limited selection of similarity modules. It is possible that more potential baseline models could be identified given a larger number of evaluations. Elasticsearch is likely to add support for more similarity modules in the near future, and this project provides the groundwork functionality necessary for their easy inclusion in the developed tool. Also if researchers develop new retrieval models, future work could expand the tool's functionality to allow their easy integration into the tool for testing.

This project could also be expanded to evaluate multilingual queries. This could involve either the conducting of new relevance assessments in different languages, or the utilization of machine translation to represent the existing English query set in many different languages. This would be valuable for assessing the consistency of the results across multiple languages, as it is possible that different retrieval models would be more suitable as baselines in non-English languages.

References

1. The Khresmoi Project [*http://khresmoi.eu*, last accessed 20th March 2019]

2. Bekir, Taner and Dinçer: IRRA at TREC 2012: Divergence From Independence (DFI) (2012). [https://trec.nist.gov/pubs/trec21/papers/irra.web.nb.pdf]

3. Ralf Bierig, Advanced Concepts in Computer Science 2 Lecture Notes, Maynooth University (2019).

4. Shannon, C.E. and Weaver, W., The Mathematical Theory of Communication, University of Illinois Press, Urbana (1964).

5. Lancaster, F.W., Information Retrieval Systems: Characteristics, Testing and Evaluation, Wiley, New York (1968).

6. Jim Cowie, Yorick Wilks: Handbook of Natural Language Processing, Chapter 10 – Information Extraction, University of Sheffield (2000).

7. Information Retrieval: A Health and Biomedical Perspective by William Hersh (2003).

8. Stephen E. Robertson and Hugo Zaragoza: The Probabilistic Relevance Framework: BM25 and Beyond (2009).

9. Stephen E. Robertson and Karen S. Jones: Relevance weighting of search terms (1988).

10. Proceedings of the Third Text REtrieval Conference (TREC 1994). [https://trec.nist.gov/pubs/trec3/t3_proceedings.html]

11. Manning, C. D., Raghavan, P., & Schütze, H, Introduction to Information Retrieval, Cambridge University Press, Cambridge (2008).

12. Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman: Mining of Massive Datasets, Stanford University (2010). [http://infolab.stanford.edu/~ullman/mmds/book.pdf]

13. Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, SIGIR (2011). [https://dl.acm.org/citation.cfm?doid=2009916.2010070]

14. Shannon, Claude; Weaver, Warren: The Mathematical Theory of Communication (1949).

15. Bekir, Taner and Dinçer: IRRA at TREC 2012: Divergence From Independence (DFI) (2012). [https://trec.nist.gov/pubs/trec21/papers/irra.web.nb.pdf]

16. Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze: Introduction to Information Retrieval, Cambridge University Press (2008).

17. Mark D. Smucker and James Allan: An Investigation of Dirichlet Prior Smoothing's Performance Advantage (2006). [http://maroo.cs.umass.edu/getpdf.php?id=694]

18. Diane Kelly: Methods for Evaluating Interactive Information Retrieval Systems with Users (2009).

19. Bhanu Peddi, Huijun Xiong and Noha ElSherbiny: TREC_EVAL: IR Evaluation, Virginia Tech (2010).

20. CLEF eHealth 2019 Information Retrieval Tasks [https://sites.google.com/site/clefehealth/tasks, last accessed 20th March 2019]

21. The Elastic software stack by Elastic [https://www.elastic.co/products, last accessed 20th March 2019]

22. Logstash data processing pipeline by Elastic [https://www.elastic.co/guide/en/logstash/current/index.html, last accessed 20th March 2019]

23. Kibana data visualization tool by Elastic [https://www.elastic.co/guide/en/kibana/6.4/index.html, last accessed 20th March 2019]

24. Elasticsearch official documentation [https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html, last accessed 20th March 2019]

25. Elasticsearch ‘Search API’ [https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html, last accessed 20th March 2019]

26. Elasticsearch ‘Reindex API’ [https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html, last accessed 20th March 2019]

27. JSON tutorial by W3Schools [https://www.w3schools.com/jS/js_json_intro.asp, last accessed 8th March 2019]

28. Java JsonObject library documentation [https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html, last accessed 8th March 2019]

29. JSON in Java tutorial by tutorialspoint [https://www.tutorialspoint.com/json/json_java_example.htm, last accessed 8th March 2019]

30. Java URLConnection library documentation [https://docs.oracle.com/javase/7/docs/api/java/net/URLConnection.html, last accessed 1st February 2019]

31. HTTP requests in Java tutorial by Yong Mook Kim [https://www.mkyong.com/java/how-to-send-http-request-getpost-in-java/, last accessed 1st February 2019]

32. Web application for regular expression testing and verification [https://regex101.com, last accessed 20th March 2019]

33. Jan Goyvaerts: Regular Expressions – The Complete Tutorial (2006).

34. WindowBuilder extension for Eclipse IDE [https://www.eclipse.org/windowbuilder/, last accessed 20th March 2019]

35. TREC_EVAL official GitHub repository [https://github.com/usnistgov/trec_eval, last accessed 30th November 2018]

36. Java Process class tutorial by GeeksforGeeks [https://www.geeksforgeeks.org/java-lang-process-class-java/, last accessed 9th March 2018]

37. Allan Hanbury: Medical information retrieval: an instance of domain-specific search. (2012).

38. Chengxian Zhai and John Lafferty: A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval, Carnegie Mellon University (2017).

39. Guido Zuccon, Bevan Koopman and João Palotti: ECIR 2015: Advances in Information Retrieval (2015). [https://link.springer.com/chapter/10.1007/978-3-319-16354-3_62#citeas]

40. Isabelle Stanton, Samuel Ieong and Nina Mishra: Circumlocution in diagnostic medical queries (2014). [https://dl.acm.org/citation.cfm?id=2609589]

41. Anupam Das: A Study of Transparency and Adaptability of Heterogenous Computer Networks with TCP/IP and IPV6 Protocols (2012).

42. Lucene Query Syntax [https://www.elastic.co/guide/en/kibana/current/lucene-query.html, last accessed 20th March 2019]

43. Leveling, Johannes and Goeuriot, Lorraine and Kelly, Liadh and Jones, Gareth J.F. (2012) DCU@TRECMed 2012: Using ad-hoc baselines for domain-specific retrieval. In: Medical Records Track, Text Retrieval Conference (TREC 2012), November 2012, Gaithersburg, USA.]

44. CLEF eHealth Challenges [https://sites.google.com/site/clefehealth2015/task-2, last accessed 20th March 2019]

45. Elasticsearch [https://www.elastic.co/products/elasticsearch, last accessed 20th March 2019]

Appendices

Please find project README, source code and unit tests attached, or visit the following Github repository: https://github.com/IRevaluation/IRmedical/