Metasearch Engine

Siobhán Grayson
12254530

MSc. Computer Science

School of Computer Science and Informatics
University College Dublin

July 2013

Contents

1 Introduction **1**

1.1 Overview . . . 1

1.2 Motivation . . . 1

2 Functional Requirements **2**

2.1 Complex Search Ability . . . 2

2.2 Web-Based User Interface . . . 2

2.3 Results Display . . . 3

2.3.1 Non-Aggregated . . . 3

2.3.2 Aggregated . . . 3

2.4 Evaluation . . . 3

2.5 Clustering . . . 4

2.6 Query Re-Write Functionality . . . 4

2.7 User Evaluation Survey . . . 5

3 Implementation **6**

3.1 User Interface . . . 7

3.2 Sending the Query and Collecting Results . . . 9

3.2.1 Complex Search Ability . . . 10

3.3 Aggregation . . . 10

3.4 Query Rewrite . . . 12

3.5 Clustering . . . 14

3.5.1 Tokenisation . . . 14

3.5.2 TF-IDF . . . 14

3.5.3 K-Means Clustering . . . 16

4 Technologies Used **18**

4.1 Languages Used . . . 18

4.2 Software Used . . . 19

5 Problems Encountered **20**

6 Evaluation Details **22**
6.1 MAP . . . 22
6.2 F-Measure . . . 23
6.3 User Evaluation Survey . . . 24

7 Conclusion and Future Work **27**

Bibliography **27**

Chapter 1

Introduction

1.1 Overview

The purpose of this project was to create a working meta-search engine, that took in a user's query, retrieved results from Google, Bing, and Blekko, and displayed a maximum of 100 results either aggregated, non-aggregated, or clustered.

1.2 Motivation

The purpose of designing a meta-search engine is to create a an engine that combines the results of three engines to create a better list of results. It allows users to enter a query once and search multiple engines simultaneously. It aims to improve on individual search engines by utilising the fact that the web is too large for any one search engine to fully crawl, so a combination of results should provide a more comprehensive results list. Another advantage is that they don't need to build up a physical database of documents.

Chapter 2

Functional Requirements

2.1 Complex Search Ability

Meta supports both boolean AND, and NOT searches completely, along with standard basic keyword search ability. No matter what option the user has selected they are able to use AND and NOT to help retrieve the required results. However, OR is not completely supported by this engine. Unfortunately, although Google and Bing both support the OR operator and return the desire results, Blekko does not. Blekko Help states that "There is no way to express an OR with blekko; use multiple searches instead." [1]. As such, OR boolean queries requiring aggregated results may return non OR results from Blekko. For non-aggregated results, The OR query can be seen to return the desired documents from Google and Bine, but not Blekko.

2.2 Web-Based User Interface

The web-based user interface has been designed with easy of use in mind. Keeping the engine simple and uncluttered was deemed vitally important in enhancing the user's experience. The search bar is placed centrally on the page. Each of the nine radio option buttons is labelled, and the options that a user can select kept to a minimum. This eliminates any sort of learning curve a user must overcome to use the engine at it's best. As providing a user with too many options may be overwhelming at times resulting in the user giving up altogether and selecting a different engine to use. As such, the amount of words returned for query re-write was kept to a maximum of 5. Results were displayed on the same page for non-aggregated results and clustered results. This allowed the user to avoid switching between multiple tabs or buttons every few seconds to compare each list.

The structure and layout of the interface is designed using HTML, while CSS provide the colour scheme and design. A lime green, light-grey, and white colour scheme being the predominate them. Links are coloured in a cyan blue, whilst visited links are coloured purple. In keeping with the simplicity theme of the engine's design, a minimal font style was selected from Google Fonts API. Through out the whole course of using the engine, the user should feel as though they have not changed page and nor should they feel the need to change page or have axillary pages involved.

2.3 Results Display

As apart of this engine's basic functionality, the user can select whether results are displayed aggregated or non-aggregated, for each search they commit. These options are displayed beneath the search text box. Only one of these options can be selected at time.

2.3.1 Non-Aggregated

Non-aggregated results are returned from each engine separately, and displayed in their original rankings. All three results lists are displayed at the same time, side-by-side, within the results page. Allowing the user to conveniently choose and compare the results from three different engines all at once.

2.3.2 Aggregated

Aggregated results, combine the results from all three underlying search engines, Google, Bing, and Blekko, and displays the most relevant results first. Result rankings in the list are determined using scores computed by Reciprocal Rank Fusion (RRF). An advantage of using this method is that it does not require relevance scores, unlike combSUM and combMNZ [2]. As well as this, RRF requires no special voting algorithm or global information [3]. Unlike Borda Fusion which uses positional algorithms, and Condorcet Fusion which uses majoritarian algorithms [4].

2.4 Evaluation

The performance of the meta-search engine was evaluated using a variety of Information Retrieval evaluation metrics. PHP scripts were especially constructed for the retrieval of search engine results to be used within the evaluation metrics. Using a static collection of 50 queries provided on the UCD CSI Moodle website, 100 results were returned for each of these queries under aggregated and non-aggregated conditions. These results were then stored in a text file where another specially designed PHP script extracted them and compared them to a second text file of results. This second text file of results being the

'Gold Standard' of results. In other words, if the engine returned the perfect set of results for a query, it would be them. The resulting MAP, Precision@10, Precision, Recall, and F-measures scores were then computed using further self-coded PHP scripts. Statistical analysis of these metrics were then conducted using paired T-tests in Microsoft Excel.

2.5 Clustering

The center radio option beneath the search text box is clustering. When the user selects for results to be clustered, aggregated results are displayed to the user divided into four groups, called clusters, instead of on combined list. Each cluster contains documents that are similar to each other. The advantage of this being that one needs only read through which ever list of results is most relevant to them. Without have to go through one long list returned results, many of which could be irrelevant.

2.6 Query Re-Write Functionality

A second additional feature that is incorporated into the meta-search engine is query re-write. Located to the left of the search text box, it is titled 'Suggest Words' so as to be a clear as possible as to the function it provides. Two options are available, on, and off. Whilst off, no query re-writing takes place upon the user's submitted query, apart from the boolean filter applied to queries going to Google and Blekko.

When 'On' is selected, the user's query is re-written using tokenisation and stop-word removal, the final product being condensed into a single word which is sent to a thesaurus engine [5] as outlined in Chapter 3, Section 3.4. A maximum of five 'suggested words' will then appear beneath the search text box along with the results of the the original query. The user can then decide to either select one of the suggested words and re-submit their query. Or they may dismiss the words entirely and enter a new query within the search box.

This, however, does not the affect the query that has been sent to each of the search engines to retrieve results, where only a boolean filter is applied. It was not deemed necessary to apply a query re-write operation upon the query being sent to the search engines, as each of the underlying engines themselves appear to incorporate a certain level of query re-write functionality already with regards to stop-word removal and relevant words etc.

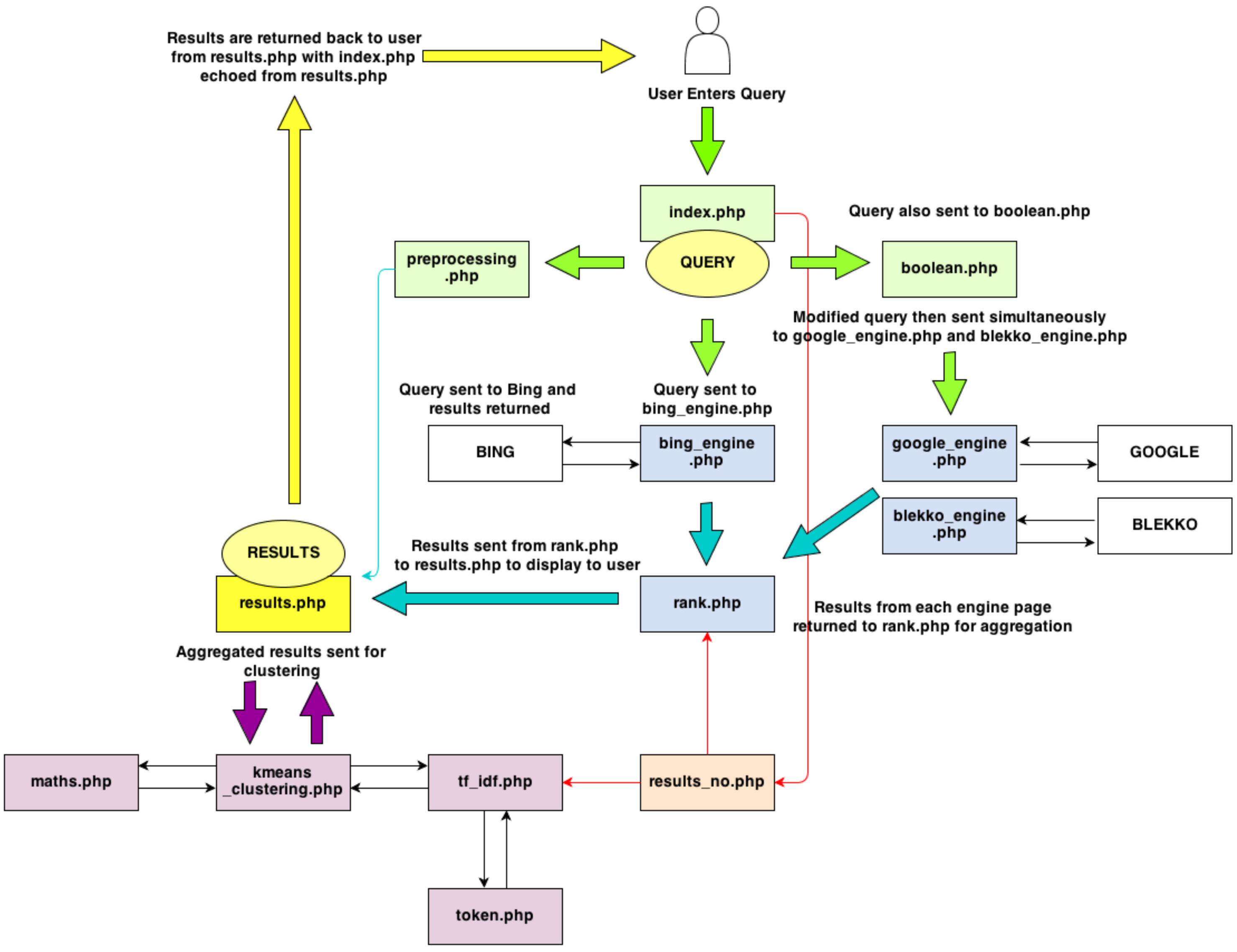
2.7 User Evaluation Survey

There are two parts to a search engine's user experience: the user interface, consisting of the design of the search form and results pages. The second part being it's functionality, how well it finds and sorts relevant results. To find out how satisfied an end user is with the overall *meta* experience, a survey was designed and linked to within the footer of the index and result pages. Peers and colleagues were then linked to the engine itself and encouraged to test and offer their feedback through submitting a complete survey. The survey itself was built using a third party website called 'Survey Monkey' which also collected and displayed all the results for easy analysis [6]. The questions used for the user evaluation survey, as well as the results, can be found in Chapter 6, Section 6.4, of this report.

Chapter 3

Implementation

An outline of the entire meta-search system implemented is depicted in the system architecture diagram of Figure 3.1 below.



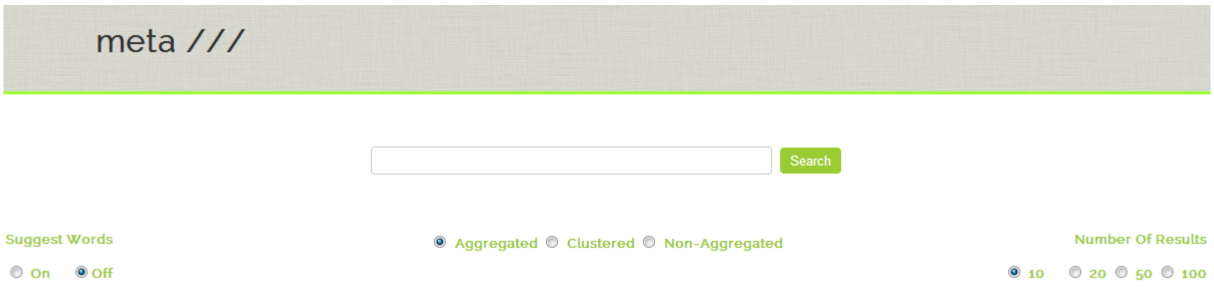
Results are returned back to user from results.php with index.php echoed from results.php
User Enters Query
index.php
QUERY
Query also sent to boolean.php
preprocessing .php
boolean.php
Modified query then sent simultaneously to google_engine.php and blekko_engine.php
Query sent to Bing and results returned
Query sent to bing_engine.php
BING
bing_engine .php
google_engine .php
GOOGLE
blekko_engine .php
BLEKKO
RESULTS
results.php
Results sent from rank.php to results.php to display to user
rank.php
Results from each engine page returned to rank.php for aggregation
Aggregated results sent for clustering
maths.php
kmeans _clustering.php
tf_idf.php
results_no.php
token.php

Figure 3.1: System Architecture Diagram.

3.1 User Interface

The first part of the meta-search engine that was constructed was the form used to enter and post queries. This form was created using a combination of PHP, HTML, and CSS. It is apart of the *index.php* page, which links to *meta2.css* where the overall design of the site is defined. It consists of a search text box, search button, and nine radio option buttons for auxiliary search features. The search text box being the area where the user enters their query and submits the entire form using the 'Search' button.

The first two radio buttons are to turn on/off the query rewrite functionality, located to the left of the screen. By default it is checked to be off. The next three radio buttons allow the user to decide whether they would like results returned as aggregated, clustered, or non-aggregated. Aggregated is selected by default and the user cannot select more than one of these options at a time. They are located under the search text box, in the center of the screen. The final four radio buttons are used to allow the user to specify the number of results they would like returned. The options being 10, 20, 50, or 100, with 10 selected by default.



Suggest Words
On [✓ Off]

[✓ Aggregated] Clustered Non-Aggregated

Number Of Results
[✓ 10] 20 50 100

Figure 3.2: The user interface of the meta-search form.

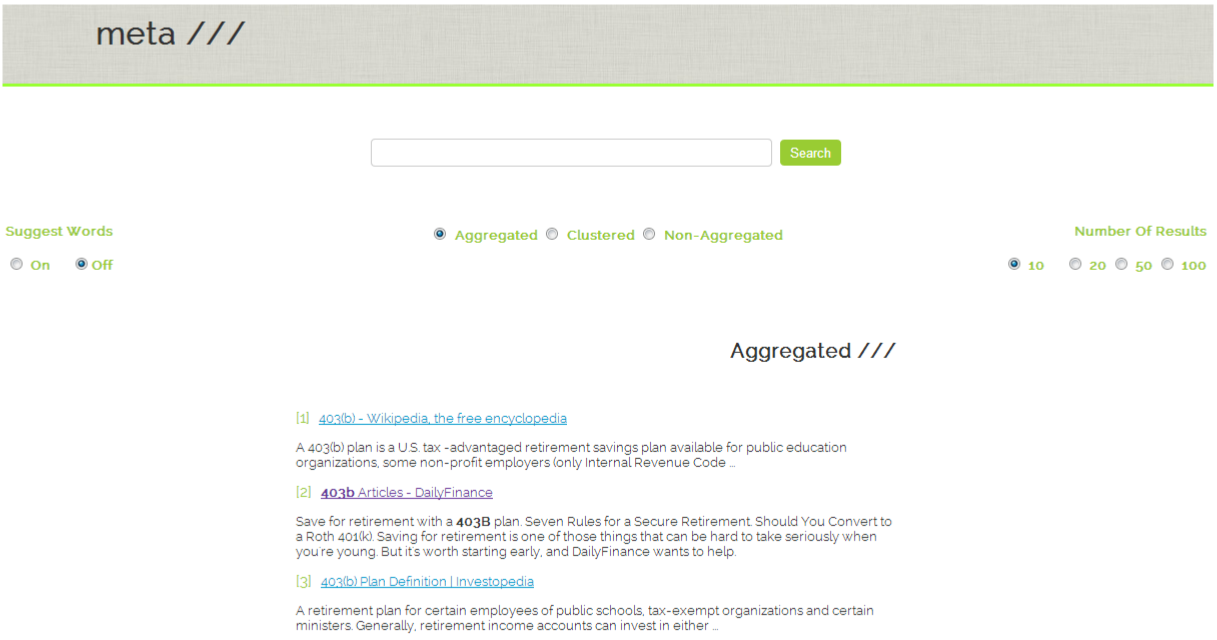
One then needed a way to display the returned results. In this instance, the *results.php* page is used. On this page the php function `file_get_contents()` was used as follows:

```
$contents = file_get_contents('index.php');
```

This function reads the contents of a file into a string [7]. Thus, since the entire contents of *index.php* is read into this string, the function `str_replace()` can be used to replaced certain parts of the *index.php* page. HTML comments are used, which are hidden from the interface itself, to identify and distinguish areas of the HTML code where results are to be incorporated. For example:

```
$contents = str_replace('<!--AGG RESULTS-->',$agg_string, $contents);
```

Once the `$contents` string has been adjusted accordingly, it can then be 'echoed' back to the user. This echo displays the entire *index.php* file via it's string counterpart, along with any modifications made to the string. The effect is that results appear to be displayed on the same page the initial query was entered. Another effect is that any queries entered after the initial query will now be submitted via the *results.php* page echoing the contents of **index.php**, as opposed to the actual *index.php* page. Figures 3.3 to 3.5 depict the display of how each result type is returned.



Suggest Words: ( ) On [✓ Off]

[✓ Aggregated] ( ) Clustered ( ) Non-Aggregated

Number Of Results: [✓ 10] ( ) 20 ( ) 50 ( ) 100

Aggregated ///

[1] 403(b) - Wikipedia, the free encyclopedia

A 403(b) plan is a U.S. tax -advantaged retirement savings plan available for public education organizations, some non-profit employers (only Internal Revenue Code ...

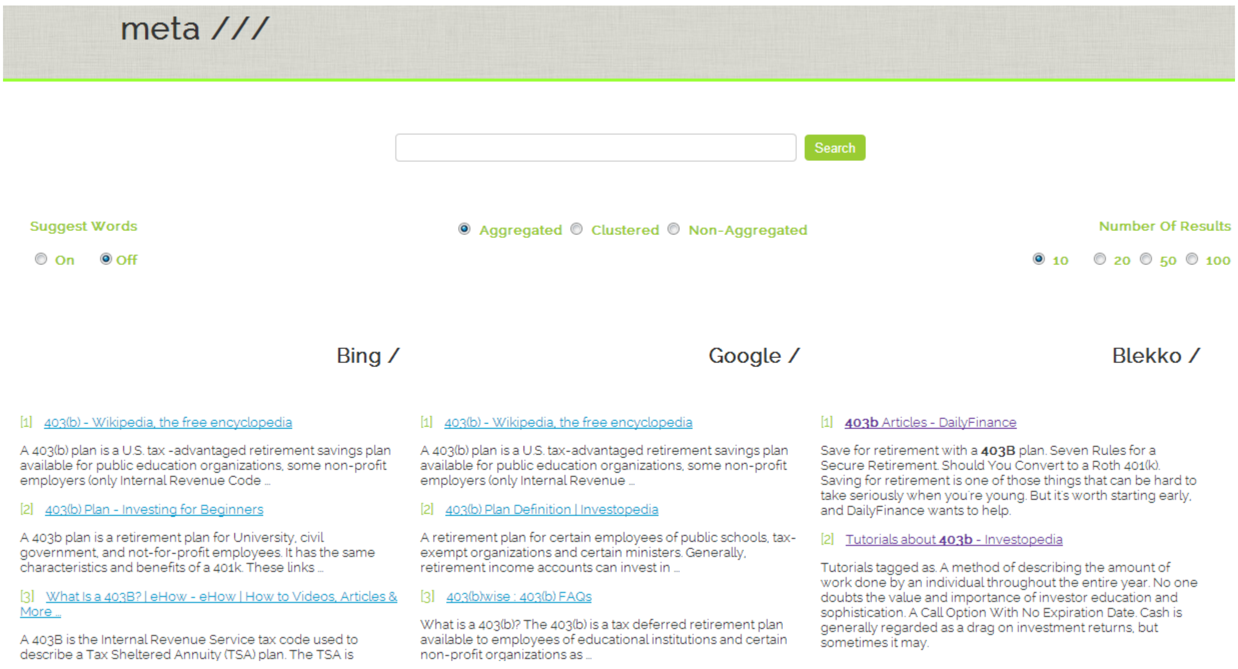
[2] **403b** Articles - DailyFinance

Save for retirement with a **403B** plan. Seven Rules for a Secure Retirement. Should You Convert to a Roth 401(k). Saving for retirement is one of those things that can be hard to take seriously when you're young. But it's worth starting early, and DailyFinance wants to help.

[3] 403(b) Plan Definition | Investopedia

A retirement plan for certain employees of public schools, tax-exempt organizations and certain ministers. Generally, retirement income accounts can invest in either ...

Figure 3.3: The meta-search engine interface for aggregated results.



Suggest Words: ( ) On [✓ Off]

[✓ Aggregated] ( ) Clustered ( ) Non-Aggregated

Number Of Results: [✓ 10] ( ) 20 ( ) 50 ( ) 100

Bing /

[1] 403(b) - Wikipedia, the free encyclopedia

A 403(b) plan is a U.S. tax -advantaged retirement savings plan available for public education organizations, some non-profit employers (only Internal Revenue Code ...

[2] 403(b) Plan - Investing for Beginners

A 403b plan is a retirement plan for University, civil government, and not-for-profit employees. It has the same characteristics and benefits of a 401k. These links ...

[3] What Is a 403B? | eHow - eHow | How to Videos, Articles & More ...

A 403B is the Internal Revenue Service tax code used to describe a Tax Sheltered Annuity (TSA) plan. The TSA is

Google /

[1] 403(b) - Wikipedia, the free encyclopedia

A 403(b) plan is a U.S. tax-advantaged retirement savings plan available for public education organizations, some non-profit employers (only Internal Revenue ...

[2] 403(b) Plan Definition | Investopedia

A retirement plan for certain employees of public schools, tax-exempt organizations and certain ministers. Generally, retirement income accounts can invest in ...

[3] 403(b)wise : 403(b) FAQs

What is a 403(b)? The 403(b) is a tax deferred retirement plan available to employees of educational institutions and certain non-profit organizations as ...

Blekko /

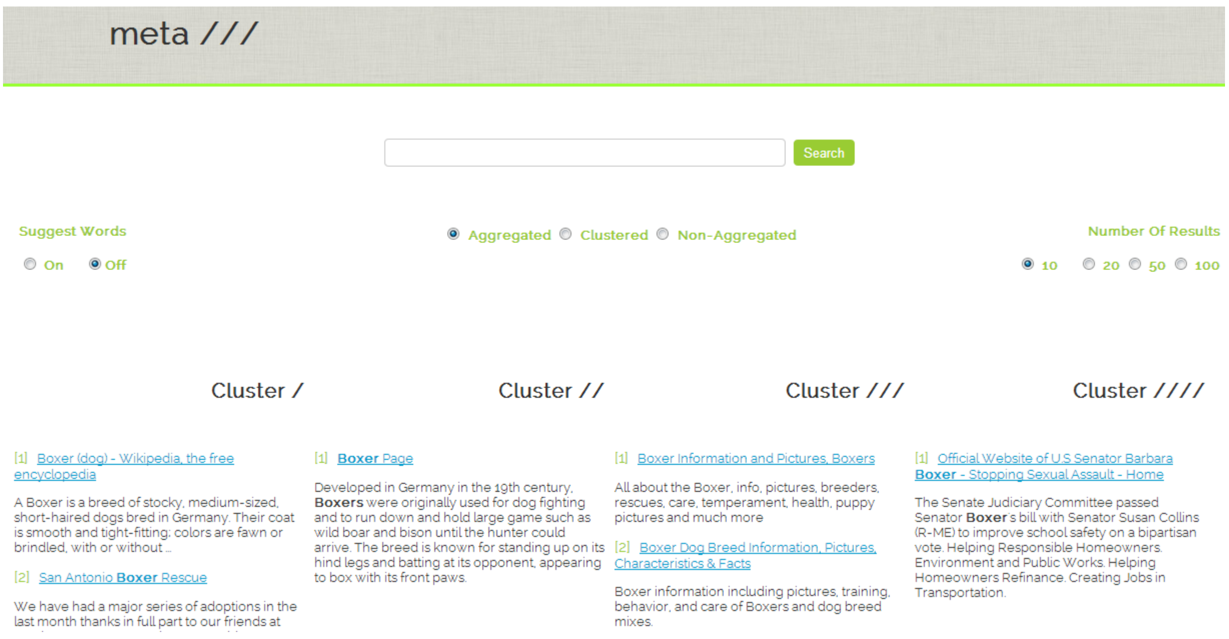
[1] **403b** Articles - DailyFinance

Save for retirement with a **403B** plan. Seven Rules for a Secure Retirement. Should You Convert to a Roth 401(k). Saving for retirement is one of those things that can be hard to take seriously when you're young. But it's worth starting early, and DailyFinance wants to help.

[2] Tutorials about **403b** - Investopedia

Tutorials tagged as. A method of describing the amount of work done by an individual throughout the entire year. No one doubts the value and importance of investor education and sophistication. A Call Option With No Expiration Date. Cash is generally regarded as a drag on investment returns, but sometimes it may.

Figure 3.4: The meta-search engine interface for non-aggregated results.



Suggest Words: On [✓ Off]

[✓ Aggregated] Clustered Non-Aggregated

Number Of Results: [✓ 10] 20 50 100

Cluster /

[1] Boxer (dog) - Wikipedia, the free encyclopedia

A Boxer is a breed of stocky, medium-sized, short-haired dogs bred in Germany. Their coat is smooth and tight-fitting; colors are fawn or brindled, with or without ...

[2] San Antonio **Boxer** Rescue

We have had a major series of adoptions in the last month thanks in full part to our friends at

Cluster //

[1] **Boxer** Page

Developed in Germany in the 19th century, **Boxers** were originally used for dog fighting and to run down and hold large game such as wild boar and bison until the hunter could arrive. The breed is known for standing up on its hind legs and batting at its opponent, appearing to box with its front paws.

Cluster ///

[1] Boxer Information and Pictures, Boxers

All about the Boxer, info, pictures, breeders, rescues, care, temperament, health, puppy pictures and much more

[2] Boxer Dog Breed Information, Pictures, Characteristics & Facts

Boxer information including pictures, training, behavior, and care of Boxers and dog breed mixes.

Cluster ////

[1] Official Website of U.S Senator Barbara **Boxer** - Stopping Sexual Assault - Home

The Senate Judiciary Committee passed Senator **Boxer**'s bill with Senator Susan Collins (R-ME) to improve school safety on a bipartisan vote. Helping Responsible Homeowners. Environment and Public Works. Helping Homeowners Refinance. Creating Jobs in Transportation.

Figure 3.5: The meta-search engine interface for clustered results.

3.2 Sending the Query and Collecting Results

For sending the query request to each of the individual search engines, Google, Bing, and Blekko, the query is built into a url string. Each engine having it's own type of URL to be sent. The URL constructed for each, generally consisted of a search engine's root URL, an assigned API, the query itself, and result parameters stating how many results to be returned or skipped, and what format to return them in. For instance, the Blekko URL was initialised as follows:

```
$rootUri = 'http://blekko.com/ws/?';
$requestUri5 = "$rootUri"."q=$query+";
$requestUri = "$requestUri5"."/json+/ps=$return_no&auth=XXXXX";
$url = $requestUri;
```

Where `$query` enters the user submitted query into the url for parameter `q`, `$return_no` states the amount the results to collect (the amount chosen by the user through the form), and `auth` being the parameter for where the API key is included. The parameter `json+` is the format that results are to be returned in. Note that before the query string is included in the URL it is first URL encoded using the PHP function `urlencode()`.

Once the query url has been constructed the URL transfer library 'libcurl' is then employed to send the URL and return the JSON formatted data into a string. The PHP function **json_decode()** is used to take the collected JSON encoded string and convert it into a PHP variable, in this case an array. The array is then parsed for the relevant information, such as each result's Title, URL, and Description, which are stored in a returned results array.

The above processes are carried out for each individual engine on separate PHP pages entitled *google_engine.php, bing_engine.php*, and *blekko_engine.php* as can be seen in Figure 3.1. Using the user defined variable `$return_no`, the exact amount of documents required could be retrieved from each engine. However, since Google only returns ten results for each query, an increment by ten loop had to be incorporated into the code for collect results. The retrieval of results from Bing also had to be repeated if the user required 100 document. The results arrays are then incorporated into the *rank.php* page to either be passed straight to *results.php* for 'Non-Aggregated' results or processed for Aggregated results and Clustering.

3.2.1 Complex Search Ability

As the meta-search engine is required to support not only standard basic keyword search ability but also boolean AND, OR, and NOT searches, the user submitted query had to be modified before being sent to Google and Blekko. Bing already supports all three boolean operators [8]. This is why the query is sent directly to *bing_engine.php* as opposed to *boolean.php* initially, as it is for *google_engine.php* and *blekko_engine.php.*

Within the boolean page the query is URL encoded and then modified such that each occurrence of '+AND+' is replaced with '+', and each occurrence of '+NOT+' is replaced with '+-'. However, no modification is included for OR. The first reason is because Google already supports the OR operator [9]. The second reason being that Blekko does not support the OR operator at all, Blekko Help stating that "There is no way to express an OR with blekko; use multiple searches instead." [1]. Once the query has been modified it is sent on to be included in each engine's request URL.

3.3 Aggregation

For aggregation, the result arrays collected from each engine are sent to *rank.php*, where they are compared and combined into a single array and displayed as a single list of results. Each document is listed in the order it was retrieved. In other words, it's index indicates it's rank from the engine it was retrieved from. As well as a document's rank information, each document index corresponds to a sub-array. The first element of the sub-array holds the document's URL where 'www' has been trimmed from the start and the final '/' from the end. This is what will be used to compare each of the documents together and prevent replication of aggregated results. Hence, they are trimmed to create a uniformity such that documents from different engines may be compared accurately[1].

For example, a returned URL from Google written as:

http://website.com/home

is seen as equivalent to a result returned from Bing written as:

http://www.website.com/home/

With each becoming formatted and stored as:

website.com/home

The second element within each document array contains the URL link, Title, and Description, formatted to for display.

The method used to define the order in which these results are listed is Reciprocal Rank Fusion (RRF). RRF merges documents returned into a unified list using only the rank positions of the retrieved documents to assign a simple score, which are essentially determined by the underlying search engines themselves [10]. To implement Reciprocal Rank Fusion, one computed the rank score r of say document d_i, using the following formula:

$$r(d_i) = \frac{1}{\sum\limits_j 1/k(d_{ij})} \tag{3.1}$$

where k = position information of d_i in all the systems $j \in \mathbb{N}$.

As such, since each document is going to be compared to each other using their first array element as opposed to their index. Each document's index was switched with the first element of it's array. Thus, not only can documents now be compared to other engine results using their index, but their ranking information is maintained from which a RRF score may be computed using Formula 3.1.

Once switched, a score is then applied to each document, and documents are combined together using the their trimmed URL indexes for comparison. If two identical documents are retrieved from separate engines, they become one element in the aggregated array with their rank scores added together. When the array of aggregated results is compiled, each document's score is then used as the denominator to divide a numerator of 1, completing

[1]Note, the formatting above takes place within each result array's respective engine page, where they are collected.

the computation of each document's reciprocal rank score.

Though an explicit weighting is not applied to any of the engine's results. The order in which arrays are combined together using the `add_to_total_array()` function allows one to implement a slight bias towards one engine over another. In this case, Google results take preference, followed by Bing, and finally Blekko. Therefore, if two documents from different engines have the same score, one document from Google, and the second from Blekko. The Google document will precede the Blekko document within the list of returned results.

Finally, the aggregated array is then sorted using the PHP function `asort()`, which sorts the associative array in ascending order. I.e. sorts the array such that documents with the lowest reciprocal rank score are displayed first. A new array is then initialised to hold only the top `$return_no` of results of the aggregated array, using the PHP function `array_slice()`. The array is then sent to either *results.php* to be displayed to the user, or to *kmeans_clustering.php* to be clustered.

3.4 Query Rewrite

An advanced feature that is incorporated into the meta-search engine is query rewrite. The user decides whether to activate query rewrite functionality using the form contained within *index.php*. If it is 'switched on', the original query that the user enters, not only gets sent to the underlying engines to retrieve document results. But it also gets sent to a thesaurus engine via the *preprocessing.php* page. The engine used to retrieve synonyms from is *Big Huge Thesaurus* [5].

In much the same way as document results are retrieved, synonyms for the query are returned from *Big Huge Thesaurus* using an URL request containing an URL encoded query, an assigned API, JSON chosen as the format for results to be returned in, and cURL being used to complete the URL transaction.

However, unlike the engine pages, *preprocessing.php* also 'tokenises' the query before it is sent to the thesaurus. This is because the thesaurus only allows for one word to be queried at a time. As such, to ensure that not only are results returned, but that the results are relevant, the PHP function, `explode()`, is used to turn the query string into an array. Each word within the array is then converted to lower-case, with non alpha-numeric characters and surplus white-space removed. Stop-words are also removed[2].

[2]The list of stop-words used was retrieved from *Bonomo's Blog* [11].

The resulting array is then filtered to remove any empty array elements that may have arisen due to stop-word removal, and the array's internal pointer is rewound to point at the first element within the array. This is the word that gets sent to the thesaurus. It was not deemed necessary to attain synonyms for all the remaining words within the array as generally a user is more likely to type higher relevant terms first. It is also not the engine's primary function to act as a thesaurus. Thus, in an effort to not overshadow the actual purpose of the meta-search engine by returning lengthy synonym lists for each query rewrite request. The amount of returned 'related words' was limited to a maximum of five words.

The returned list of synonyms is then displayed beneath the search text bar as can be seen in Figure 3.6. A short javascript script is included within *index.php* to allow the user to re-populate the search text box field of the meta-search form, with the desired query re-write word, by just clicking on the word, within in the list, below the search box.
Allowing the user to quickly assess, select, and search again, using the new suggest word.

javascript on index page.

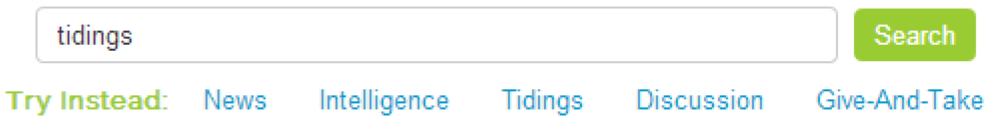


Figure 3.6: The suggested words that result from the query "word". The user has selected "tidings" which can be seen to have automatically filled the search text box.

If a query resulted in the no related results being returned, the following message "Sorry, there are no related words for this query." was displayed instead. As can be seen in Figure 3.7 below.

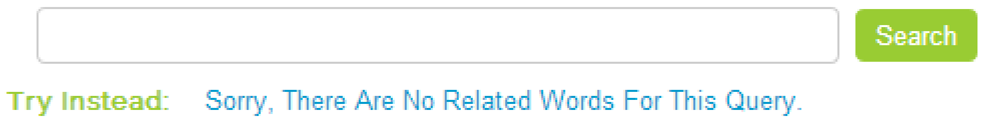


Figure 3.7: *Try instead:* displaying the message "Sorry, there are no related words for this query" when a query does not retrieve any synonyms.

3.5 Clustering

Clustering refers to the partitioning of data into a number of groups, or clusters. It's an unsupervised technique, unlike classification, as there aren't any examples to learn which groups different types of data should go into [14]. Clustering of documents was performed using a number of steps. The first step was to create a vector space model of the entire aggregated results set. Each document being represented by a vector of n-dimensions if it contains n amount of terms. Terms in this instance refers to the separate words that make up a document. Terms were identified for each document by tokenising the strings a document contained. Each term was then assigned a numeric value or weighting using a method known as Term Frequency - Inverse Document Frequency. After which, the K-means clustering method could then be applied.

3.5.1 Tokenisation

Documents are tokenised using functions within *token.php.* First, snippets and titles for each result are amalgamated into one string, with html tags removed, using a function called `snippet_array()`. This function utilises PHP functions `preg_replace()` and `strip_tags()`.

The function `tokenise_array()` then converts letters into lower-case, removes all non alpha-numeric characters, and replaces all stop-words with a space. Lastly, all surplus white-space is removed from the string, including extra white space between words.

A third function, `porter_stemming`, is also located within this page. It takes in the prepared words, after they have been 'exploded' into an array, and stems the words contained within using a Porter Stemming script obtained from tartarus.org [12]. This transforms each word into it's stem. Helping to reduce the amount of unique words contained within the document set.

Note, that although array strings are not technically tokenised on this page. They are prepared such that each string can be immediately 'exploded' into an array of tokens.

3.5.2 TF-IDF

The *tf_idf.php* page takes in the aggregated array and applies the above functions. A 'dictionary' array is then initialised, along with a document count array. Each modified document string is exploded into an array, `$terms`, where each element is an individual word. A Porter Stemming function is then applied, calling a Porter Stemming script. The variable `$docCount`, counts how many terms are contained within each document array.

The `$dictionary` array is used to store each term as it's index. Each term holding an array itself consisting of relating information. The overall structure of the `$dictionary` array is as follows:

```
$dictionary =>
    [df] = (increments for each document a term is found in.)
    [appearsInDocs] =>
        [document index] =>
            [tf] = (increments each time a term is found within a document).
```

Once the dictionary array is populated, a new array called `$index`, is constructed with the first element containing the `$docCount` array, and the second element containing the `$dictionary` array. From this array the TF-IDF value for each term is computed and stored in an array `$tf_idf`, where each document's ID is the index, holding the array of TF-IDF values for each term within that document. Consequently, each document snippet has now been transformed into a vector array whose dimension is equal to the amount of terms it contains. To calculate each term's TF-IDF value, formula 3.2 was used in conjunction with formula 3.3.

$$tf_{ij}idf_{ij} = tj_{ij}\log_2(N/df_i) \tag{3.2}$$

The 'tf' value contained for each term in the dictionary of `$index` is normalised across the entire corpus of document snippets, using formula 3.2 [13].

$$tf_{ij} = f_{ij}/\max\{f_{ij}\} \tag{3.3}$$

Where f_{ij} = 'tf' and $\max\{f_{ij}\}$ = the highest value for 'tf' across all document snippets.

The resulting code implementation of formulas 3.2 and 3.3 combined is as follows:

```
$tf_idf = ($appearsInDocs['tf']/$max_term_freq )*(log($docCount / $entry['df'],
2));
```

Where `$max_term_freq` holds the value found for $\max f_{ij}$.

A vector space model now exists in the form of an array of document vectors, held within the array, `$doc_vectors`. Each document vector is filtered to remove empty array elements in an effort to reduce each array's dimensionality, and the remaining term indexes are reset to start at zero and increment by one.

3.5.3 K-Means Clustering

K-means clustering is a process for grouping similar documents together. It works by identifying the space of possible solutions, creates each of the K clusters and distributes them randomly throughout an available solution space. For this project I decided to create 4 clusters each time, $k = 4$. Since the choice of initial clusters may be of random selection from the array of document vectors, the first four vectors were chosen. k was set to equal 4 as from testing different values of k over the different amount of documents that could be returned, it appeared to give the most adequate results when used for all four different amounts.

Before the arrays are clustered, each of the document vectors is normalised by dividing each component by the length of the entire vector, so that all vector documents have a unit length. This simplifies the latter calculation of cosine similarity. Cosine similarity is a formula which one to calculate the similarity between to documents. By taking the dot product of two vectors, one is computing the cosine of the angle between the two vectors. For vectors pointing in opposite directions this will be equal to -1, if they're at 90 degrees to each other it will be 0, and if they are identical, i.e. there is no angle between them, it will be 1. Hence, documents with the highest values relative to a cluster are the best matches, their vectors are pointing in the same direction as the cluster [15].

The k-means clustering takes place within the *kmeans_clustering.php* page. Before the data is clustered, a `$doc_set` array is initialised and filled with the index of each document, along with the value of their first TF-IDF weighting. This is so as to be able to identify and match back up the documents with their vectors after clustering has taken place. The function, `kmeans()`, is then applied to the normalised vector space. Firstly, it initialise four clusters from the vector space. In this case, the first four vectors are used. It then clusters the rest of the vectors based on these four clusters.

Taking one document vector at a time, the cosine similarity is computed between the document in question and each of the four clusters. Whichever cluster gives the largest result, is the cluster that the the document vector gets assigned to. Since the new document has been added to the cluster, the position of the cluster's centroid will have changed. To find the new position of the centroid, the vector average is taken between the original cluster and the newly added document cluster. The result being assigned as the new centroid for the cluster in question, to which the rest of the document vectors in the set are compared against.

Once the entire the document vector space is assigned to a cluster, vectors are then re-combined with their matching document index and sent to *results.php* to be displayed. Though originally, one had iterated the clustering process over the entire document vector space multiple times, it seemed to inhibit rather than enhance the clustering results. With each iteration placing more and more documents into the same cluster. As such, it was reverted back to assigning vectors to clusters just once.

Chapter 4

Technologies Used

A selection of different technologies were used throughout this project. In particular, API keys for each underlying engine of the meta-search engine were required to retrieve results. Four API keys used were:

1. Google API
2. Bine API
3. Blekko API
4. The Big Thesaurus API

4.1 Languages Used

Languages used to build the meta-search engine included:

1. HTML, used to define the structure of the pages.
2. CSS, used to embed design into the engine.
3. PHP allowed one to incorporate all the behind the scenes coding, required to allow the engine to work.
4. CURL was used to send request URL to retrieve results.
5. JSON was the format that results from engines were received in.
6. Finally, a modest amount of Javascript was used on the index page to allow for the suggest words buttons to re-populate the search-text box form.

4.2 Software Used

Software used included:

1. Dreamweaver, the application where the code was written.
2. Wamp server, for compiling and testing pages locally on one's laptop.
3. winScp for file transfers to the UCD server.
4. The UCD server which is used to host the site.
5. Google Chrome and Internet Explorer were used to view the site.
6. Survey Monkey was used to create the user evaluation survey and collect results.

Chapter 5

Problems Encountered

One of the first major challenges to overcome was in learning how to return results from each engine. Although at first, it was hard to find a place to start, eventually, after adequate research, results were returned using cURL. The next hurdle was in then experimenting with what exactly was important and not important within the JSON data returned. In the end, one felt that only the Title, URL, and Snippet/Description were needed for the purposes of the meta-search engine.

However, once results were returned, there were still problems. First, Bing returns a maximum of 50 results, and Google only 10 results for each query at a time. To overcome this the query was looped to be sent twice to Bing when over 50 results are required, and the query for Google had to be looped whenever over 10 results were required. Testing was further inhibited by Google's 100 queries a day limit. Thus, five different Google API accounts were activated to get 4 extra API keys. This allowed for 500 queries to be sent it day instead.

When aggregating results, it was initially problematic in finding a solution as to how best to combine each array in PHP whilst implementing Reciprocal Rank Fusion. Furthermore, during aggregation, it came to one's attention that Bing and Blekko were returning duplicate results when 100 results were requested. They also failed to consistently supply 100 results. Often returning fewer. To eliminate the duplication of results problem, the PHP function `array_unique()` was applied to the returned results arrays of Bing and Blekko. This course of action was not necessary for Google.

One of the last, but not the least, major challenges of this project was in implementing clustering of aggregated results. Initially, one had set the K-means clustering to iterate over the clustering process until no further changes occurred. However, for larger documents sets, such as when 100 results were required to be returned, all documents ended up in the same cluster. A number of different methods were used to try and correct this behaviour, such as changing the number of clusters, or changing how the first clusters were initialised. In the end, once one changed the conditions such that k-means only performed clustering once, these problems ceased to exist.

Chapter 6

Evaluation Details

When evaluating the meta-search engine, *meta*, a number of techniques were incorporated to analyse different aspects of the engine's functionality. Such aspects include how fast the engine returns results, the quality of the results returned, and whether or not the user interface is simple and intuitive to use. Not to mention end user's satisfaction with the overall user experience.

As mentioned in Chapter 2, Section 2.4, 100 results were collected for a list of 50 queries (provided on the UCD Moodle website) for both aggregated and non-aggregated displays. These results were then compared against a 'gold standard' list of results, to determine the quality of the results returned, and in particular, whether the aggregated results were superior in quality to the non-aggregated results or visa-versa.

6.1 MAP

MAP,or Mean Average Precision, for a set of queries is the mean of the average precision score for each query. Where Average Precision refers to only a single query. MAP refers to a set of queries. MAP is the most commonly used metric in IR literature to compare the performance of two systems. A PHP script was written to compare the similarity between two sets of results for all 50 queries. The script was run such that the 'gold standard' list of results was compared against all the results returned from the meta-search engine for Google, Bing, Blekko, and all three aggregated together. The script then computed and outputted the following MAP values for each results set.

Search Engine		**MAP (%)**
Google	:	28.7
Bing	:	28.4
Blekko	:	22.2
Aggregated	:	46.8

As one can see, Google appears to be the best performing engine out of all three, although, admittedly only marginal better than Bing. Blekko having the lowest MAP scoring. Aggregated is clear winner, it's MAP score almost double that of each of the individual underlying engines.

To demonstrate whether or not the difference in evaluation scores is significant, statistical analysis in the form of a Paird t-test in Microsoft Excel were performed between each individual engine compared against the aggregated engine. In this case, data consisted of the Average Precision score returned for each query by an engine. Thus, 50 data points for each engine were used altogether. A pair t-test was chosen, since all the queries are the same. The results of each engine compared against the aggregated engine are as follows:

Search Engine		**P{T <= t} two-tail**
Google	:	8.34207151838647E-18
Bing	:	1.39015851445012E-19
Blekko	:	1.19661427188502E-23

Each value is significantly less than 0.05. As such, the difference is considered to be statistically significant, at a 95% confidence level. Consequently, the aggregated is in fact a returning superior results than each of the underlying search engines. A complete list of each engine's results, including Precision@10, Precision, and Recall can be found in the Appendices of this report.

6.2 F-Measure

F-measure assesses the trade-off between precision and recall and is sometimes referred to as the harmonic mean. If precision and recall are both high, the F-measure will be high. The following table displays each engines Average F-measure score.

Search Engine		**Avg. F-Measure (%)**
Google	:	47.7
Bing	:	49.9
Blekko	:	40.5
Aggregated	:	78.4

Interestingly, this time Bing has over-taken Google, meaning that although Bing scored slightly lower in the MAP evaluation, it's recall score must be a lot better than Google's. Hence, why it's F-measure is higher. However, Aggregated is yet again the highest scoring engine, scoring well in both precision and recall.

6.3 User Evaluation Survey

Twenty-one users altogether submitted the survey before the submission of this report. One insightful response I got from a user is that they found the interface, not as intuitive as it could be. Stating that they had never used a meta-search engine before and did not know what ‘Clustered’ or ‘Aggregated’ referred to. As such, if I had further time, I would have incorporated hover descriptions to appear over the nine option buttons. To explain to the user what they are, and what they are used for.

The entire list of survey results is as follows:

[1] Have you ever used a meta search engine before?

Yes	:	7
No	:	14

[2] Which search engine do you normally use?

Google	:	15
Bing	:	2
Yahoo	:	3
DuckDuckGo	:	0
Other	:	1 person entered ‘Tor’

[3] After submitting my query, the meta search engine relayed results in a time comparable to that of my typical search engine of choice.

Strongly Disagree	:	0
Disagree	:	0
Neither Agree Or Disagree	:	4
Agree	:	11
Strongly Agree	:	6

[4] I found that the quality of the returned results were superior to my typical search engine of choice.

Strongly Disagree	:	0
Disagree	:	2
Neither Agree Or Disagree	:	9
Agree	:	9
Strongly Agree	:	1

[5] In general, I found the quality of the aggregated results returned to be superior to the non-aggregated results returned.

Strongly Disagree	:	0
Disagree	:	0
Neither Agree Or Disagree	:	5
Agree	:	12
Strongly Agree	:	4

[6] I found the query suggestions feature to be very useful.

Strongly Disagree	:	0
Disagree	:	0
Neither Agree Or Disagree	:	3
Agree	:	15
Strongly Agree	:	3

[7] I found the interface very intuitive to use.

Strongly Disagree	:	0
Disagree	:	1
Neither Agree Or Disagree	:	2
Agree	:	12
Strongly Agree	:	6

[8] I liked how the results were displayed.

Strongly Disagree	:	0
Disagree	:	0
Neither Agree Or Disagree	:	3
Agree	:	13
Strongly Agree	:	5

[9] I found it useful when the final results were clustered.

Strongly Disagree	:	0
Disagree	:	2
Neither Agree Or Disagree	:	3
Agree	:	11
Strongly Agree	:	5

[10] If give the option, I would consider making this my default search engine.

Strongly Disagree	:	0
Disagree	:	2
Neither Agree Or Disagree	:	6
Agree	:	9
Strongly Agree	:	4

Chapter 7

Conclusion and Future Work

I have learnt numerous new skills throughout this project. In particular, I learnt a lot about PHP, cURL, and JSON. The foundation of this project is based upon PHP and it was an interesting language to have engaged with. It was also interesting to have gained knowledge into the workings of how search engines themselves function. How a query is processed or not processed. Knowing when a query should be processed and in what way. How to send a query and receive results.

Probably the most fascinating part of this project was in implementing the clustering of documents using a vector space model and K-means clustering. An extremely practical use of linear algebra. Learning how a document could be easily translated into a n-dimensional vector and then compared to over document in the same vector space to quantify similarity between the two was extremely interesting and relevant to the needs of today. Where information retrieval systems are playing a larger and larger role in our society.

In future work, one would like to expand on the TF-IDF and K-Means implementation within this project. To specialise the clusters more and assign the clusters more meaningful names. I would also change the layout of the overall engine to be more compact. Especially on the repetition of cURL throughout the pages. To increase the success rate of the 'Suggest Words' feature, I would also like to incorporate spelling auto-correction. This would also enhance the results returned. Perhaps even extend the suggest words feature and combine it with TF-IDF to pull suggested words from the top K documents returned from the original query.

Bibliography

[1] Greg. (2012, September 21). "Advanced Search Features". *Blekko Help.* Retrieved from http://help.blekko.com/index.php/advanced-search-features/.

[2] Aslam, J. A., & Montague, M. (2001, September). Models for metasearch. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 276-284). ACM.

[3] Cormack, G. V., Clarke, C. L., & Buettcher, S. (2009, July). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (pp. 758-759). ACM.

[4] Montague, M., & Aslam, J. A. (2002, November). Condorcet fusion for improved retrieval. In *Proceedings of the eleventh international conference on Information and knowledge management* (pp. 538-548). ACM.

[5] Watson, J. (2013). "Thesaurus". *Big Huge Thesaurus Tech Republic.* Retrieved from http://words.bighugelabs.com/

[6] Survey Monkey. (2013). Retrieved from http://www.surveymonkey.com/.

[7] The PHP Group. (2013). "File_get_contents." *PHP.* Retrieved from http://php.net/manual/en/function.file-get-contents.php

[8] Microsoft. (2013). "About Bing Query Language". *MSDN Library.* Retrieved from http://msdn.microsoft.com/en-us/library/ff795657.aspx.

[9] Google. (2013). "Search Operators". *Web Search - Help - How to search.* Retrieved from https://support.google.com/websearch/answer/136861?hl=en.

[10] Nuray, R., & Can, F. (2006). *Automatic ranking of information retrieval systems using data fusion. Information processing & management, 42(3)*, 595-614.

[11] Bonomo. "List of English Stop words hard coded in the php array". (2010, February 23). *Bonomo's Blog.* Retrieved from http://bonomo.wordpress.com/2010/02/23/list-of-english-stop-words-hard-coded-in-the-php-array/

[12] Hayes, R. “PorterStemmer Class”. Retrieved from http://tartarus.org/martin/PorterStemmer/php.txt

[13] Manning, D, C., Raghavan, P., Schutze, H. (2008). “Document Representation using TF-IDF and Vectors”. *Introduction to Information Retrieval.* Cambridge University Press.

[14] Barber, I. (2009, August 14). “K-Means Clustering”. *PHP/ir - Clustering.* Retrieved from http://phpir.com/clustering.

[15] Barber, I. (2009, September 19). “Simple Search: The Vector Space Model”. *PHP/ir - search and boolean retrieval.* Retrieved from http://phpir.com/simple-search-the-vector-space-model