MetaSearch – Preliminary Research

Patrick Moorehouse

Student Number: Omitted

Email: Omitted

Overview

This report focuses on two topics. The first is summarising the findings of my research into the available methods of data fusion, or, results aggregation, from multiple search engines, into a single list of ranked results. I will list some of the various aggregation methods available today, giving a summary of their mechanics, and will state the method chosen for implementation, along with my reasons for choosing it over the other options available.

The second topic concerns query pre-processing. I will discuss the importance of query pre-processing, and explain some of the different types of pre-processing that can be carried out on user submitted queries. I will explain why pre-processing is advantageous in the context of Information Retrieval Systems, such as internet search engines, and caution on some of the drawbacks involved in its application.

Aggregation Techniques

Aggregation, or fusion, of data, is the merging of multiple data sets from different input systems, to produce a more accurate, relevant resulting set than would be achievable with only one input system. In the context of internet search engines, this means taking the results returned from submitting a query to multiple different search engines, and combining them together to get a list whose contents are more relevant to the query that was submitted than a single engine would produce. This works well because we can take data from multiple sources and the documents that are more consistently highly ranked across multiple engines gain an advantage in ranking, whereas a single search engine only may have erroneous results occasionally. Therefore, the concept of a meta-search is to improve query results by aggregating the results from multiple singular search engines into one, improved list.

There are many aggregation methods that have been developed. We will now take a look at a number of popular alternatives and explain briefly their functions, and how they compare to one another.

Borda-Fuse

Borda-Fuse uses a concept based on political elections, where the search engines are the voters, and the documents that are returned by the search engines are the candidates, and documents are ranked where the candidate with the most first choice votes is placed at the top of the list. This method was developed originally by Jean-Charles de Borda, a political scientist, for use in single winner elections, but was adapted to produce ranked lists (Aslam, & Montague, n.d., MforM).

Borda-Fuse works like this; each search engine returns its own ranked list of documents. If there are n documents returned, then the top ranked document is awarded n points. The second is awarded n-1 points, and so on, until the last document, which receives n-(n-1) points, or, 1 point. The points received from multiple engines are added together to produce an overall ranked list. However, since the larger a result set is, the higher the points awarded to the top ranked document will be, the scores must be normalised. This is to prevent bias, as an engine returning 6 documents will assign 6 points to its first choice, where an engine returning only 4 will assign 4 points, even though the second engine could be better and its first choice more relevant. Thus the ranking score for Borda-Fuse is given by

$S_{BF} = \sum_i S_{di} - S_{i\text{-}min} \ / \ S_{i\text{-}max} - S_{i\text{-}min}$

where S_{di} is the score given to a document, d, by the i^th^ search engine, and i-max and i-min refer to the maximum and minimum scores attainable by that engine for that set of results.

Condorcet-Fuse

Condorcet-fuse applies an alternative method, where two documents, d_1 and d_2 are compared against each other in all datasets, or search engine results, and the one which outranks the other the most often, is ranked higher in the final fused list (Aslam, & Montague, n.d., CFforIR). Ultimately the document that can beat all other documents in a pairwise comparison across all input systems would be ranked highest. This however is not always the end result. Indeed, there may be a candidate a, which beats b in the majority of their pairings, while b might beat candidate c, and c might beat candidate a.

CombSum

CombSum takes the relevance scores for documents, from all the datasets it is included in, and sums them together. This creates a score which rewards documents that appear in more datasets than others. The scores of course must be normalised as some engines may score on different scales to others. (He, & Wu, n.d.)

Examples of this method are shown below (Lillis, 2012).

Example:

- Document #1 is given a score of 0.45 by System A
- Document #1 is given a score of 0.3 by System B
- Document #1 is given a score of 0.35 by System C

Document #1's CombSUM score is 0.45 + 0.3 + 0.35 = 1.1

Example 2:

- Document #2 is given a score of 0.55 by System A
- Document #2 is not returned by System B
- Document #2 is given a score of 0.65 by System C

Document #2's CombSUM score is 0.55 + 0 + 0.65 = 1.2

CombMNZ

CombMNZ is similar to CombSum, with the extra detail that the summed, normalised scores are multiplied by the number of datasets a document appears in, hence the name MNZ, multiply non zero. This works similarly to CombSum but adds further emphasis to the Chorus Effect, which makes it very effective in practice. It requires relevance information, but in scenarios where relevance is not provided, the ranking can be normalised and used instead. (He, & Wu, n.d.)

Examples of the CombMNZ method (Lillis, 2012);

Example:

- Document #1 is given a score of 0.45 by System A
- Document #1 is given a score of 0.3 by System B
- Document #1 is given a score of 0.35 by System C

Document #1's CombMNZ score is (0.45 + 0.3 + 0.35) x 3 = 3.3

Example 2:

- Document #2 is given a score of 0.55 by System A
- Document #2 is not returned by System B
- Document #2 is given a score of 0.65 by System C

Document #2's CombMNZ score is (0.55 + 0 + 0.65)x2 = 2.4

So we see that while document 2 scored better in the systems it was returned in, document 1 was ranked higher overall, because it appeared in 3 of the systems instead of just 2.

Choice of Aggregation Technique

Of the various techniques outlined, CombMNZ was chosen for implementation. This was chosen because of its simplicity in calculation, and also because it is an effective method. It tends to outperform both Borda-fuse and CombSum, while being no more complex than either. Condorcet-fuse is said to be superior to CombMNZ, but is more complicated, and it was my belief that the extra level of complexity was not worth the marginal benefits in performance over CombMNZ. CombMNZ is widely regarded as the standard against which other methods are judged against, because it is quite effective, yet more advanced techniques can often beat it.

Query Pre-processing

There are numerous pre-processing operations that can be applied to a search query before it is submitted. These operations improve and simplify the search process. When we search for documents in a search engine, we supply some words that we wish to search for, in the hope that finding documents containing those words will return documents that are relevant to our interests.

The words entered into a search engine are tokenised, turning it into a list of tokens rather than a list of words. These tokens can be checked against the tokens of indexed documents. This however is still not ideal, as there are many types of variation on how a word may be written.

This is why some additional operations are performed on the tokens, to normalise them. These operations include decapitalising, converting from plural to singular, equivalency classing, synonym lists and more. These procedures reduce the number of alternate versions of words, so that an index can be created where only one version of the word, the token, can be stored, along with a list of documents the token can be found in.

Some of these operations improve indexing efficiency, for example, only having to search for “roman” rather than “Romans”, “romans” and “roman”. It means that all documents containing any of these variations of these words will be returned to the user regardless of which they submit in their query.

Equivalency classing does the same, removing the need for multiple indexes of tokens which are equivalent, for example, “co-operation” and “cooperation” can be treated as being the same (Skorkin, 2010).

Some of these pre-processing operations do come with a cost however. Synonym lists, for example. They can be used to improve efficiency of the system, so that if a user enters “chair”, “stool”, “seat” or any other synonyms, documents containing any of these may be returned. It is labour intensive however, as it must be constructed by hand, and also, if the synonym list is applied at runtime to expand a query, it can impact the speed of the query (Skorkin, 2010).

Another form of pre-processing that can commonly be applied is stop word removal. Stop words are words like “a”, “the” and “and”, which form part of a query, but which generally add no benefit to the query, as these words are so common they appear in practically all documents. So removing them from the query altogether is often beneficial. This can have consequences as well however. Some names may be made up of what would typically be classified as stop words, so care is not taken, some things will never be returned. An example would be the bands called Yes, or The Who. If those words were removed, pages dedicated to those bands would never be displayed.

Stemming is an operation that effectively cuts off suffixes and prefixes to find the base form of a word. This helps in confining many variations of a word to one token.

Pre-processing of queries is important. By taking the time to apply some operations to queries at runtime, the need for a vastly larger index is prevented, while also making searches more efficient. A small penalty at runtime provides for much faster return of results and a much more efficient system in general.

References

Aslam, J. A., & Montague, M. (n.d.). Condorcet Fusion for Improved Retrieval.

Aslam, J. A., & Montague, M. (n.d.). Methods For Metasearch.

He, D., & Wu, D. (n.d.). Toward a Robust Data Fusion for Document Retrieval. Retrieved from http://www.sis.pitt.edu/~daqing/docs/datafusion-nlpke-117.pdf

Lillis, D. (2012). Software Engineering Project: Metasearch and Fusion.

Skorkin, A. (2010, March 1). *How Search Engines Process Documents Before Indexing*. Retrieved June 20, 2013, from http://www.skorks.com/2010/03/how-search-engines-process-documents-before-indexing/