Information Retrieval Assignment Two

Group One - No Brain in Index

Sean Martin, Neil Brockett, Sethuram Ramalinga Reddy, Arvind Ganesan, and Jianan Wei

ABSTRACT

The Text Retrieval Conference (TREC) has been a highly important Information Retrieval conference since 1992. In this paper, we will present our Lucene implementation to index and search a collection of news documents from the TREC Ad-Hoc track. We will demonstrate our experiments to fine tune our implementation, and compare our implementation to a standard Lucene implementation. After tuning, 32.4% Mean Average Precision (MAP) was achieved by our Retrieval system.

1 INTRODUCTION

The TREC Ad-Hoc track "investigated the performance of systems that search a fixed set of documents using new questions" [9]. Very strong research arose from the Ad-Hoc task, and many of the new ideas have become standard, such as the Okapi BM25 ranking function. Exploring the improvements in the Ad-Hoc track greatly benefited our own index and searching system.

In this paper, we will give background information on Information Retrieval and our system in particular. Then, we will describe our implementation and analyse the results of experiments with our system. Finally, we conclude on our system and give guidance for future work in the area.

2 BACKGROUND

"Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information." (Salton, 1968). In our case, we were indexing a collection of news documents and submitting topics, representing information needs, against the index. Binary relevance judgements are made against these topics, and the overall aim was to maximise the Mean Average Precision (MAP), which we define here.

Let Q be the number of queries, N be the number of documents retrieved for each query and $P@k$ be the precision (fraction of documents retrieved which are relevant to the information need) at cut-off k in the ranked retrieved list of size N. Then

$$\text{MAP} = \frac{\sum_{i=1}^{Q} \text{Average Precision for query } i}{Q} \quad (1)$$

$$\text{Average Precision} = \frac{\sum_{k=1}^{N} P@k \times \text{rel}(k)}{\text{Number of relevant documents}} \quad (2)$$

where $\text{rel}(k)$ is one if document k is relevant, and zero otherwise.

3 IMPLEMENTATION

For a retrieval system, the search results are the essential factor to satisfy the needs of the users. One of the most effective systems in TREC is the SMART system of Cornell University, led by Salton [7]. The algorithmic underpinning of SMART is to process documents and information requests into vectors by tokenisation, stop word removal, stemming, and optionally statistical phrase identification, then finally weighting each term in a vector. [9]. This was similar to processes we all previously employed on the Cranfield collection. Confident that SMART would be a strong start, this was selected as the basis for our system. Our primary contributions to build upon SMART are:

- Building a custom Lucene analyzer to combine the most effective tokenising, stop word removal and stemming.
- Automatic query expansion based on the local expansion methods in [10].
- Data fusion at search time using the CombSUM method of [3].
- Term boosting in queries to adequately weight more important parts of the query.

We will explain the parsing of the documents and queries, and then look at each of these improvements upon SMART in turn.

3.1 Document and Query Preprocessing

The data consists of 4 collections of documents, each from a different source:

(1) Financial Times Limited (1991, 1992, 1993, 1994)
(2) Federal Register (1994):
(3) Foreign Broadcast Information Service (1996)
(4) Los Angeles Times (1989, 1990)

This amounts to 1.85 GB of text data. One file typically contains several documents in a HTML-like format. In order to process the input data, we decided the best approach is to reformat the data into a common scheme. We used python to parse each document and structure the data into a JSON format. Each document contains the fields of text, headline, document number and date when possible and is provided in the JSON format. This processing step was performed once before importing the JSON files into java. To improve the performance of parsing, all files in python were handled using asynchronous, parallel processing functions. This allowed the full data to be parsed in 40 minutes on a Windows system, and had to be performed only once. Python multiprocessing is OS dependent and on MacOS the parsing was able to run in only 5 minutes.

The 50 employed queries were composed specifically for this data set and are composed of a title, narrative and description. The contents of the narrative were combined with the title and processed in java as sentence strings, which were filtered further to exclude any sentence containing the phrase "not relevant". We observed improved performance of our information retrieval system when adding this filter step.

3.2 Custom Lucene Analyzer

A Lucene analyzer is applied at query and index time to tokenise, remove stop words, and perform stemming. The choice of analyzer

can have a large impact on the efficiency and accuracy of the retrieval system. In our system, we designed the following analyzer with the steps applied by the analyzer given in order:

(1) *StandardTokenizer*: Lucene's grammar based tokeniser.
(2) *StandardFilter*: Normalises the tokens from the previous step.
(3) *LowerCaseFilter*: Turn all characters to lower case, necessary for porter stemming.
(4) *EnglishPossessiveFilter*: Remove all trailing possesive "'s" from words.
(5) *StopFilter*: Stop word removal using our custom list of stop words containing the standard English stop words provided by Lucene, and the other 80 most prominent words in the news collection.
(6) *PorterStemFilter*: Implements the stemming algorithm described by Porter in 1980 [5].

We experimented with using different stemming algorithms, but Porter stemming was the most effective in our system. We also attempted stemming stopwords and applying stopword removal after stemming, however, Porter suggests this is not useful in [6].

3.3 Automatic Query Expansion

Automatic query expansion is the process of adding to query without any user input. We originally planned to use WordNet [4], a lexical English database to automatically expand queries. The process involves looking up a term in the database, and adding synonyms to the query. However, a lexical database is similar to a thesaurus, and Xu and Croft advise against using a general thesaurus for expansion in [10]. Instead, they propose to build a thesaurus of sorts from the indexed collection of documents. When a query is passed into the system, they pull the 50 most frequent terms from the top ranked documents and add them into the query. This requires storing the index, which adds a space overhead for the technique.

Using this idea as a basis, we implemented it slightly differently. In Lucene, the *MoreLikeThisQuery* class was applied, which returns a list of representative terms of a document using a set of heuristics, such as term frequency. The number of terms to extract from a document and the number of top documents to consider for a query had large performance impacts. Empirically, we found taking the top 10 ranked documents for a query, and adding the 7 most representative terms from these documents to be fruitful.

3.4 Search Time Data Fusion

Data fusion involves merging results from different schemes, such as different term-weightings and query expansions. It has been employed to great effect in the TREC Ad-Hoc task [9, p 91]. In our system, we fused the results of the three following Lucene similarity models:

- *BM25Similarity*: The Okapi BM25 model introduced at TREC-3, with the free parameters chosen as $k = 2$ and $b = 0.89$.
- *LMDirichletSimilarity*: A language model in which Dirichlet priors are applied to smooth probability estimates instead of linear interpolation. The free parameter μ is chosen to be 1500.
- *DFRSimilarity*: Divergence from randomness, a probabilistic model by Amati and Joost [1].

In [1], Amati and Joost explain that three models must be chosen for the diveregence from randomness process - a model of randomness and two normalisation models. For the model of randomness, we chose a term frequency, inverse document frequency model. As the first normalisation applied for aftereffect, we used the B model from the paper, which uses the ratio of two Bernoulli processes. Finally, for the second normalisation, which resizes term frequency by document length, we selected a uniform distribution for term frequency. See [1, p 374] for a full list of choices.

Following the CombSUM method in [3], at search time, the three scores from these models are added together to produce the final document similarity score for a query.

3.5 Term Boosting in Queries

Boosting is a process that improves the search results by altering the default formula used for scoring. Boost is represented as a floating point number which is used to modify the relevancy score of a document. The default value of boost is 1.0, having no effect. When it is larger than 1.0, the cumulative score will be increased, as the score is multiplied by the boost.

In our system, we applied query time boosting. This assigns more importance and relevance on one or more of the terms in the query by the boost value, giving them higher weight. Empirically, we found that boosting all terms in the topic title by 8.0, the topic description by 4.0, and the topic narrative by 1.0 was effective. A powerful improvement was to boost the original topic title and description to have higher weight than the terms picked up by query expansion, but the topic narrative to have lower weight.

4 RESULTS

In Fig. 1, we present the variation in performance of our retrieval system on the full set of relevance judgements. The primary evaluation metric of interest is the Mean Average Precision, and the recall is shown for comparison.

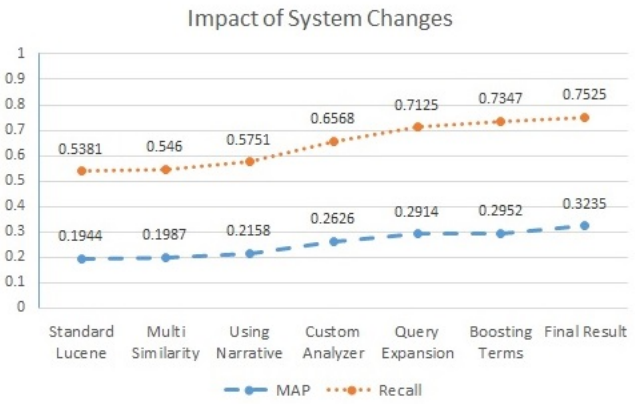

Figure 1: System performance increase over time

Beginning with a Standard install of Lucene with a standard analyzer and BM25 similarity, we improved the MAP by 0.13 at the final version of the system. The biggest improvement to the system was adding a custom analyzer, followed by query expansion.

Between the points of boosting terms and the final result, fine tuning was performed on:

- Hyper-parameters in our three similarity models.
- Number of documents to retrieve for query expansion and number of terms to add to the query.
- Exact floating point boost values.
- The list of stop words.

5 ANALYSIS OF RESULTS

To explore the successes and limitations of our system, it is most insightful to analyse the system performance on individual topics. This will highlight the areas the system performed adequately in, and those that require improvement.

5.1 Individual Topic Analysis

Some of the topics provided are hard to handle, such as topic number 417, described as "Find ways of measuring creativity". Reassuringly, the most relevant document our system returned was an interesting article from the LA times. It begins; "Why is it that Leonardo da Vinci could paint the Mona Lisa, while another person is barely able to draw a stick figure?" and is, unsurprisingly, one of the relevant documents from the relevance judgements. In fact, the system achieved a respectable 0.386 map for this query.

There was only one topic that we did not manage to retrieve any relevant documents for, number 433. The information need was for Greek philosophy specifically related to stoicism - the endurance of pain or hardship without the display of feelings and without complaint. The system failed because it found many articles on Greek philosophy, but not related to stoicism. This makes it clear that no application of phrase identification is a severe limitation to our system.

Another limitation in our system was the processing of the narrative part of a topic. In many cases, some of the narrative contained the phrase “not relevant" and when a sentenced contained this phrase, the sentence was removed. This worked well, but in some cases a more reserved approach was necessary. Instead of removing the sentence, the important part of the sentence would have to be identified, and a negative weight assigned to it - discouraging any document which contained it. Combining this with phrase identification may have allowed the system to perform well on topic 421, which asked for documents on industrial waste disposal, but they should not discuss nuclear or radioactive waste, or illegal dumping of waste. A MAP of 1.2% was the result for this topic because the system failed to respect the restriction in the narrative.

Looking at the best topics, over 80% MAP was achieved on four of the topics. For the most part, these topics have the opposite characteristics to the two above topics. The individual terms were very important in the topic, but there was not phrases. Furthermore, our application of query expansion and boosting performed well for these topics. Finally, the narrative further explained things that were relevant, rather than restricting the relevant result set.

6 CONCLUSION AND FUTURE WORK

Overall, we are pleased with our information retrieval system achieving a Mean Average Precision of 32.4% on the collection of news documents. Some of the topics are very difficult to satisfy, and certain documents are awkward to handle. As such, we are satisfied that our system can achieve a respectable performance on this news collection and topics.

In future work, some possible improvements to our system would be:

- Using N-grams, or statistical phrase identification. In our current implementation, we lose the meaning of links between words.
- Document expansion in addition to query expansion, similar to [2].
- Index time boosting, to give more weight to particular indexed fields, such as the headline of an article.
- Data fusion at index time, as in [8], where multiple streams are employed and combined.

A partial reason for not implementing some of these works in our system is that indexing is slower than searching. As such, it was better to experiment with researched ideas like expansion, boosting, and data fusion at query time as we could immediately see the results. We are confident that the same ideas applied at index time would increase the performance of the system.

REFERENCES

[1] Gianni Amati and Cornelis Joost Van Rijsbergen. 2002. Probabilistic Models of Information Retrieval Based on Measuring the Divergence from Randomness. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 357–389. https://doi.org/10.1145/582415.582416

[2] Bodo Billerbeck. 2005. Efficient query expansion. (2005).

[3] Edward A Fox and Joseph A Shaw. 1994. Combination of multiple searches. *NIST special publication SP* 243 (1994).

[4] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[5] Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

[6] Martin F Porter. 2001. Snowball: A language for stemming algorithms. (2001).

[7] Gerard Salton. 1991. The Smart Document Retrieval Project. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '91)*. ACM, New York, NY, USA, 356–358. https://doi.org/10.1145/122860.122897

[8] Tomek Strzalkowski, Fang Lin, Jose Perez-Carballo, and Jin Wang. 1997. Building Effective Queries in Natural Language Information Retrieval. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLC '97)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 299–306. https://doi.org/10.3115/974557.974601

[9] Ellen M Voorhees, Donna K Harman, et al. 2005. *TREC: Experiment and evaluation in information retrieval*. Vol. 1. MIT press Cambridge.

[10] Jinxi Xu and W Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 4–11.