Search Engine for the Cranfield Collection

Information Management (CS7IS3) - Continuous Assessment 1

Claire Gregg
cgregg@tcd.ie
ICS Year 5
Trinity College Dublin, Ireland

ABSTRACT

This assignment focuses on the basics of implementing a simple search engine using Lucene, with the Cranfield collection as a dataser. This includes creating an index from the document set, with any required pre-processing and a selection of an Analyzer. This same Analyzer should then be used when querying the index, along with some similarity scoring method. The best solution I found for this assignment (evaluating using trec_eval) is Lucene's built in *EnglishAnalyzer*, with some extra stopwords added, using BM25 similarity scoring, and an extra analyzer for queries - *QueryAutoStopWordAnalyzer*.

1 INTRODUCTION

The Cranfield Collection was collected in the 1950s. It consists of around 1400 "documents", containing the abstracts of aerodynamics journal articles, along with titles, authors, and more. It includes 225 example queries. The manually judged relevance of each document to each query is provided. [3, *Standard Test Collections*]

Lucene is a Java library which can be used to build a fully functioning search engine. It provides analysis, preprocessing, and tokenisation tools for indexing and querying documents. [1]

trec_eval is a tool used to evaluate search engines, given a set of queries whose relevance has been judged, and the results of the search engine on the queries. This provides a number of evaluations which can be used to compare different search engine implementations against each other.

2 METHODOLOGY

2.1 Creating the Index

First, the Analyzer. Analyzers contain tokenizers and filters, and are applied to text before indexing and applied to queries before querying an index. [2] I initially made use of the *StandardAnalyzer*. However, I switched to using the *EnglishAnalyzer*, as this does stop-word removal and stemming. Stop word removal removes commonly used words which do not impart much/any meaning. For example, the *EnglishAnalyzer* removes words like a, an, and, are, as and at. Stemming brings words to their base form - for example "drinks", "drinking", and "drinker" are all stemmed to "drink". *EnglishAnalyzer* uses the commonly used *Porter Stemming Algorithm*.

In addition to the default *EnglishAnalyzer* stop words set, I added which, us, been, were and from. I chose these by finding the most frequently used terms in the index and extracting words which did not have significant meaning.

2.1.1 Parsing input data. The Cranfield documents are provided in the following format: ".I <id> .T <title> .A <authors> .B <bibliography> .W <document text>". Each section is of varying length, with varying amounts of space/newlines. I seperate the overall input into documents based on .I, and split each document based on matches to this regex:

```
\.(\w)\s((.*\s)*?)(?=(\.\w)|(\Z))$
```

2.1.2 Text preprocessing. Any punctuation is removed from the text. This can then be written to the index using the analyzer.

2.2 Querying

Querying uses the same analyzer and preprocessing as discussed in the previous section. However, on top of this analyzer, it also makes use of a *QueryAutoStopWordAnalyzer*. This is intended to be used on top of another analyzer for queries, to stop common words from being passed in in queries.

2.2.1 Scoring Approaches. 2 scoring approaches are compared - the basic Vector Space Model (specifically using *ClassicSimilarity*) and the BM25 model. These models are applied at both index time and query time. BM25 deals better with high frequency terms (putting less weight on them), varying length documents (avoiding longer documents having more significance), and rare terms (putting more weight on rare terms).

2.2.2 Interactive Search. While the primary focus here was applying my search engine to the Cranfield queries, I also implemented an interactive search function.

2.3 Using trec_eval

trec_eval requires the results of the search engine on the predefined queries to be in the following format:

Table 1: Required format for results file for trec_eval. Columns 2 and 6 are only required for spacing.

QueryNum	Q0	DocID	Doc Ranking	Doc Score	RunID

The file containing the relevance judgements also needs to be modified. Initially, it starts off as *QueryNumber, DocumentNumber, RelevanceJudgement*, where Relevance judgement goes from 1-4, with the occasional -1. 1 is the most relevant, 4 is of minimum interest, and -1 is of no interest. For trec_eval, all -1s must be mapped to 5, and an extra column must be added to the relevance file in between *QueryNumber* and *DocumentNumber*, containing 0s.

Once this is done, trec_eval is run on the results.

3 EVALUATION

The results from trec_eval are given in Table 2. These show:

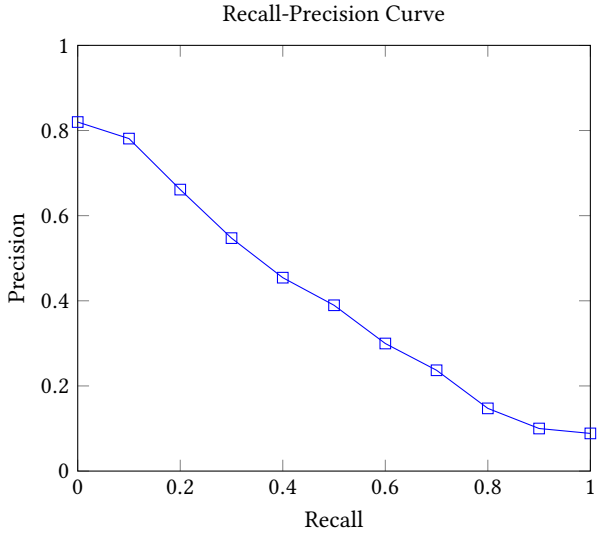
Table 2: trec_eval results with different configurations. Configurable elements include the analyzer used for indexing and querying, whether the additional QueryAutoStopwordAnalyzer is used for queries, and the similarity scoring model used. MAP stands for mean accurate precision. P_5 represents the precision of the first 5 documents returned, etc.

Configuration	MAP	P_5	P_10	P_15	P_20	P_30	P_100	P_500
Standard Analyzer, VSM scoring	0.3420	0.3902	0.2643	0.2060	0.1701	0.1289	0.0443	0.0089
Standard Analyzer, VSM scoring, Query Auto Stopword Analyzer	0.3480	0.4018	0.2674	0.2089	0.1748	0.1329	0.0458	0.0092
Standard Analyzer, BM25 scoring	0.3490	0.4000	0.2777	0.2140	0.1792	0.1333	0.0451	0.0090
Standard Analyzer, BM25 scoring, Query AutoStopword Analyzer	0.3501	0.4009	0.2777	0.2158	0.1788	0.1327	0.0454	0.0091
English Analyzer, VSM scoring	0.3752	0.4188	0.2857	0.2217	0.1850	0.1394	0.0484	0.0097
English Analyzer, VSM scoring, Extra Stopwords	0.3790	0.4196	0.2875	0.2253	0.1862	0.1409	0.0490	0.0098
English Analyzer, VSM scoring, Query Auto Stopword Analyzer	0.3769	0.4241	0.2844	0.2226	0.1842	0.1393	0.0484	0.0097
English Analyzer, VSM scoring, Extra Stopwords, Query Auto Stopword Analyzer	0.3808	0.4277	0.2853	0.2244	0.1857	0.1411	0.0490	0.0098
English Analyzer, BM25 scoring	0.3840	0.4268	0.2875	0.2256	0.1862	0.1396	0.0483	0.0097
English Analyzer, BM25 scoring, Extra Stopwords	0.3874	0.4268	0.2888	0.2259	0.1875	0.1403	0.0487	0.0097
English Analyzer, BM25 scoring, Query Auto Stopword Analyzer	0.3853	0.4268	0.2875	0.2247	0.1859	0.1385	0.0481	0.0096
English Analyzer, BM25 scoring, Extra Stopwords, Query Auto Stopword Analyzer	0.3870	0.4277	0.2875	0.2253	0.1868	0.1396	0.0484	0.0097

(1) *EnglishAnalyzer* is generally better than *StandardAnalyzer*. This is likely due to its handling of stopwords and stemming.
(2) BM25 similarity scoring is generally better than Vector Space Model scoring.
(3) The addition of extra stopwords (found to be high frequency in the document set) increases the accuracy of the search engine.
(4) The use of *QueryAutoStopwordAnalyzer* on top of any analyzer for querying improves precision, but is can still be improved by the removal of extra stopwords.

The recall/precision graph of the best configuration is shown in Figure 1. Only this configuration is shown as other configuration results are difficult to distinguish on the graph. As expected, as the number of documents received increases (recall), the proportion of retrieved documents which are relevant decreases. The ideal recall-precision graph would have the line close to the right-upper side of the graph.

Figure 1: This graph shows the recall precision graph. Recall represents the proportion of documents retrieved, while precision represents the proportion of *relevant* documents retrieved. As recall goes up, precision goes down.



4 CONCLUSION

This was an interesting assignment to become acquainted with information retrieval in practice - and Lucene in particular.

If repeating this assignment, I would upgrade to a more recent version of Lucene to gain access to more modern features, and would try to use a larger variety of Similarity Scoring models. However, I'm happy with the structure of the code I created, and my work with the *EnglishAnalyzer*.

Overall, I found that the best configuration for a Lucene search engine on the Cranfield collection is one using the inbuilt *EnglishAnalyzer* (with a few extra stopwords included), BM25 scoring, and *QueryAutoStopwordAnalyzer* to further filter queries.

REFERENCES

[1] Apache [n. d.]. *Welcome to Apache Lucene.* Apache. https://lucene.apache.org/
[2] baeldung. [n. d.]. *Guide to Lucene Analyzers.* baeldung. https://www.baeldung.com/lucene-analyzers
[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. *Introduction to information retrieval* (reprinted ed.). Cambridge Univ. Press, Cambridge [u.a.]. https://nlp.stanford.edu/IR-book/html/htmledition/contents-1.html