PART 1 - Cranfield Collection Search Engine

Uzair Qureshi
15318872 MAI CD - 2020
CS7IS3 - INFORMATION RETRIEVAL AND WEB SEARCH

***Abstract*— There exists a paradox in not knowing what you are looking for in the sea of zillions of documents on the web, but you hope an inanimate machine does. Our daily life is heavily influenced by the ease in information access provided by big-tech search engines like Google. Which equates to lesser need for keeping simple facts in our memory and over time, because of the explosion of information and data, the need for better and more innovative information retrievals systems will continue. To study the forefront of information retrieval, one must know the basics. Within this assignment we were given a very practical problem to solve. Given a collection of documents with different fields, how do we index and search the collection effectively. We use Lucene [mccandless2010lucene-1], 'an open-source search software', which provides us with all the neccessary APIs to create indexes and build queries with great level of granularity. To evaluate the success of the search engine we create, trec_eval [buckley2004trec-1] is run, a standard tool established by the TREC community for assessing between results file and standard judged results.**

Main criteria being:

1) **Use of appropriate Lucene Analyzers for content processing (Stop Word and Stemming);**
2) **Implement and Test different scoring approaches in Lucene (BM25 and Vector Space Model)**

Project hosted on github. github.com/QUzair/LuceneSE Whole project can be run via a bash script provided in repo. Instructions laid out in README.md.

I. Main Project Structure

A. Files

- **cran.all.1400** - Contains 1400 documents from the Cranfield Collection.
- **cran.qry** - Queries that will be used to test our Implementation of the Search Engine with trec_eval.
- **QRelsCorrectedforTRECeval** - RelDocs used for evaluation of our own search results.

B. Classes

- **CranFileParser.java** - Parses Cran Docs File and Index it with specified Analyzer.
- **CranfieldQueries** - Parses Cran Queries File and creates DockRank for queries.
- **CranfieldModel** - Basic model for field in cranfield doc (id,title,author,source,content).

II. Indexing

The setup was relatively simple, since we were given a tutorial during a lecture, and the documentation for Lucene was well documented. We create an *IndexWriter* with a default StandardAnalyzer or a CustomAnalyzer. These analyzers perform the text operations on the fields prior to indexing. This is the same analyzer that operates on the query strings as well for consistency. Then, we look at parsing the Query File and breaking it into the different fields. The parser in place is then calls a custom *addDocToIndex* method which returns a Lucene document with fields *id,title,author,source,content*. Once all the documents are added by the IndexWriter, we have finished Indexing.

- Create Index Write with Analyzer
- Create and add Document
- Close IndexWriter and IndexDirectory

III. Searching and Results

A. Parsing Queries

Similar to the previous step, we parse the file of 255 queries. This time we only store the query string, since the id is sequential and is accounted for in the modified *QRelsCorrectedforTRECeval*

1) Search: For every query string create a Lucene query with a specified Analyzer (same as the IndexerWriters'), the IndexSearcher runs the

2) Scoring: Lucene Scoring uses Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine relevance of a document for a query. In essence, the more times a query term appears in a document relative to the number of times the term appears in all documents, the more relevant the document is. In VSM, documents and queries are represented as weighted vectors, each index term being a dimension and the weights are tf-idf values. Boolean Models narrow documents to be scored base on Boolean logic in Query Specification.

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d) \right)$$

Lucene Practical Scoring Function

Fig. 1. Practical Scoring Function

- **tf (t in d)** - Term Frequency, number of time term t appearing in currently scored document d.
- **idf (t)** - Inverse Documnent Frequency, Rarer terms give higher contribution to total score.
- **t.getBoost()** - Search time boost of term t in query q. Boost importance of field, doc

3) BM25: BM25 has the potential of giving negative scores for terms with high document frequency and belongs to probabilistic models. Both BM25 and TF-IDF define a weight for each term as a product of a idf and tf function

and summarise it as a score for the whole document for a given query. The most important aspect of BM25 being the saturation of provides by asymptotically approaching a limit for high term frequencies. For this reason it has proven more suited for collections with short documents.

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

Fig. 2. BM25 Scoring Function

IV. Results

Following Results were obtained by running the search engine with different Similarity(Scoring) Techniques and two Different Analyzers.

	StandardAnalyzer	CustomAnalyzer
TF-IDF	0.1557	0.2796
Boolean	0.1782	0.2781
BM25	0.2864	0.3375

This was the following P/R graph obtained by trecEval for BM25 with the specified CustomAnalyzer.

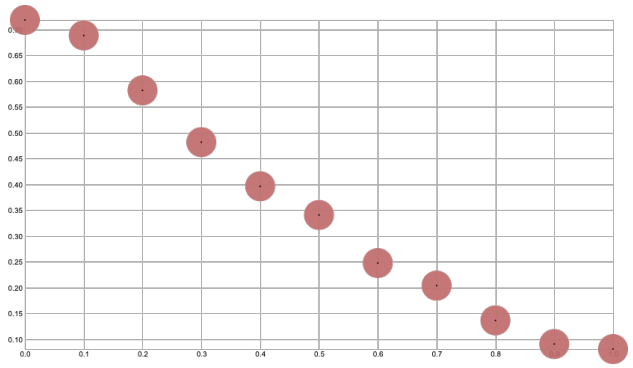

Fig. 3. P/R Precision(Y) vs. Recall(X)

A. CustomAnalyzer

The CustomAnalyzer extended the Analyzer class and applied the following Filters

- EnglishPossessiveFilter - May not be that effective since collection is not anecdotal.
- PorterStemFilter
- EnglishMinimalStemFilter
- KStemFilter

Finally, I created a text file full of the most common stop words from https://www.ranks.nl/stopwords which considerably increased the MAP.

V. Conclusions

The field of Information Retrieval is moving at a fast pace. It is important to note that this was a relatively monotonous collection and that the WWW has become multi-modal. Meaning, scoring algorithms need to refer to more than just terms but understanding of the collection at hand and how to match human intuition with these algorithms. Most Enterprise Scalable Big Data Analytics solutions are based on ElasticSearch, which is built on Lucene. So it was nice to get a more practical experience on how it all maps together and how we can customize the functionalities of the engine so simply.

References

buckley2004trecChris Buckley. trec eval ir evaluation package, 2004. mccandless2010luceneMichael McCandless, Erik Hatcher, Otis Gospodnetić, and O Gospodnetić. *Lucene in action*, volume 2. Manning Greenwich, 2010.