Project on
Information Retrieval System

By
Piyush Goel
Siddhant Benadikar
Kavya Vasudevaraju

Information Retrieval
Spring 2017
Northeastern University,
College of Computer and Information Science

Under the guidance of
Dr. Nada Naji

1 INTRODUCTION

Today, information on the internet is growing at a faster rate than ever before. Representation, storage, organization of and access to these information items form the basis of the field of Information Retrieval.

The objective of this project is to implement these core concepts of Information Retrieval by building search engines. And also, evaluate and compare their performance levels in terms of retrieval effectiveness.

The multiple phases of the project involves building an indexer for a given corpus, obtaining ranked list of documents for the given query list using different retrieval models, performing query expansion using pseudo relevance feedback technique. We also perform evaluation using effectiveness measures.

We also perform statistical tests to find the differences in performance and implement snippet generation technique and query term highlighting on documents for a query.

1.2 Contributions

Siddhant Benadikar :

a. Implemented tf-idf retrieval algorithm
b. Implemented MAP, MRR, P@K evaluation metrics
c. Modularized indexer to perform indexing on the stemmed corpus
d. Documentation

Piyush Goel :

a. Parsing documents to extract important text and building an inverted index
b. Implemented pseudo-relevance feedback algorithm for query expansion
c. Implemented statistical tests to compute difference in performances
d. Documentation

Kavya Vasudevaraju :

a. Added stopping and stemming to BM25 and tf-idf runs
b. Implemented snippet generation and query term highlighting
c. Modified BM25 to accept relevance information
d. Documentation

2 LITERATURE AND RESOURCES

2.1 Indexing

Indexing is a critical part of building a search engine. Text processing is an aspect of indexing where words are converted into index terms. We use a unigram indexer where the terms are stored in an inverted list as follows:

WORD (docID, tf), (docID, tf), …

where *docID* is the name of the document, *tf* is the count of the number of times the term occurs in that document, and *WORD* is a unigram.

2.2 BM25 Retrieval System

BM25[2] is a bag-of-words ranking algorithm which ranks documents based on the query terms appearing in each document. The algorithm extends the scoring function for the binary independence model to include document and query term weights. The common form of the scoring function is :

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1) f_i}{K + f_i} \cdot \frac{(k_2 + 1) q f_i}{k_2 + q f_i}$$

where the summation is over all the terms in the query, r_i is the number of relevant documents containing the term i, R is the number of relevant documents for the given query, n_i is the number of document containing the term i, N is the total number of documents in the corpus, f_i is the frequency of the term i in the document under consideration, qf_i is the frequency of the term i in the query, k_1 and k_2 determines how the *tf* component of the term weight changes as f_i increases (if 0, then tf component is ignored.), b regulates the impact of length normalization. k_1, k_2, and K values are set empirically[3].

2.3 tf-idf Retrieval System

Term Frequency - Inverse Document Frequency is a numerical statistic that indicates the importance of a word to a document in a corpus which is computed as -

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

where tf_{ik} is the term frequency weight of term k in document Di, f_{ik} is the number of occurrences of term k in the document, and the summation is the number of terms in the document.

$$idf_k = \log \frac{N}{n_k}$$

where idf_k is the inverse document frequency weight for term k, N is the number of documents in the collection, and n_k is the number of document in which the term k occurs.

2.4 Lucene Retrieval System

Apache Lucene is a free and open source information retrieval software library which provides Java based indexing and search capabilities. Lucene has been widely recognized for its utility in the implementation of search engines and local, single-site searching[7]

2.5 Query Expansion

Pseudo Relevance feedback is a automatic query expansion and refinement technique used to improve the performance of the system.
Expansion terms depend upon the whole query since they are extracted from highly ranked document for the given query but the quality of the expansion depends upon the relevance of the top-ranked documents.

2.6 Stopping

The task of removing the most common terms which are function words that help form a sentence structure but contribute little on their own to the context of the text. Creation of stop-word lists helps reduce the size of the index and also improve the effectiveness of the search engine.

2.7 Stemming

The task of reducing a word to its stem, i.e. the shortest term in the common stem. This technique produces small improvements in ranking effectiveness. Stemming can be performed aggressively or conservatively depending upon the application.

2.8 Evaluation

The metrics used to measure effectiveness of a search engine are as follows:

2.8.1 Precision and Recall

Precision is the proportion of retrieved documents that are relevant.
Recall is the proportion of relevant documents that are retrieved.

Effectiveness is measured as follows -

$$Recall = \frac{|A \cap B|}{|A|}$$
$$Precision = \frac{|A \cap B|}{|B|}$$

A is the relevant set of documents for the query and *B* is the set of retrieved documents.

2.8.2 Mean Average Precision

To summarize the effectiveness of rankings from multiple queries, the average of the precision values from each ranking is obtained as an effectiveness measure which is the MAP[4], computed as -

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

where *Q* is the number of queries, *AveP(q)* is the average precision score for the *q*-th query.

2.8.3 Mean Reciprocal Rank

It is the average of reciprocal ranks[1] over a set of queries which is computed as -

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}.$$

where Q is the number of queries, $rank_i$ refers to the rank position of the first relevant document for the i-th query[5].

3 IMPLEMENTATION AND DISCUSSION

The two major functions of a search engine are Indexing and Query process. The indexing process includes text acquisition, text transformation and index creation.

3.1 Text Acquisition

A textual corpus cacm containing html documents are parsed using BeautifulSoup[6] library and the important content is obtained by extracting the text within the <pre> tag and noise which content occurring after 'AM' or 'PM' or 'PMB'.

3.2 Text Transformation

The documents are then processed to remove all punctuation from the text except , and . occurring between numbers using regular expression. The time taken to parse the given 3204 raw documents in less than a second.

3.3 Index creation

The index is created using a dictionary where the key is a term (WORD) and value is a list of tuples (document ID, frequency of the term within the corresponding document) as shown:

{'the': [[CACM-0001,3], [CACM-0002,5],...]}

[1] Reciprocal rank is the reciprocal of the rank at which the first relevant document is retrieved.

To extract uni-grams from the documents we used third party library nltk[2]. Index creation for the entire corpus takes approximately 15 seconds to complete.

3.4 Ranking

Ranking refers to sorting the documents obtained for a query in the order of the 'most' relevance. In this project we build our search engine based on three different retrieval systems: BM25 retrieval system, tf-idf retrieval system, and Lucene retrieval system.

3.5 BM25

As mentioned in *section 2.2*, the formula uses document and query specific values. The value of *K* which normalizes the *tf* component by document length is given as:

$$K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$$

where *b* is a parameter which regulates the impact of length normalization b corresponds to 0 if there is no normalization, *dl* is the document length, *avdl* is the average document length in the collection.

We have chosen typical TREC values for $k_1 = 1.2$, $k_2 = 100$, and b = 0.75. For each query, BM25 score is calculated for all documents which contains at least one query term. The values of r and R are obtained from the relevance judgement given[2] for a query and if there is no relevance information, then r and R are 0.

3.6 Lucene

In this system[7], a simple analyzer[8] is used. The analyzer filters LetterTokenizer which is a tokenizer that divides text at non-letters, with LowerCaseFilter which normalizes token to lower case. The results obtained from a simple analyzer is similar to the results obtained from other retrieval systems.

[2] *cacm.rel is the given file containing relevant documents for a particular query*

3.7 Query-by-query Analysis

The three queries that we have chosen are:
(query / stemmed_query)

1. Parallel Algorithms / parallel algorithms
2. Performance evaluation and modelling of computer systems / perform evalu and model of comput system
3. portable operating systems / portabl oper system

For the first query, we can see that the stemmed version and the non stemmed version are almost the same except for 'algorithms' getting stemmed to 'algorithm'.

However, for the second and third query we can see that seven terms ('Performance' -> 'perform', 'evaluation' -> 'evalu', 'modelling' -> 'model', 'computer' -> 'comput', 'systems' -> 'system', 'portable' -> 'protabl', 'operating' -> 'oper') have changed after stemming.

In case of a stemmed run, all terms that belong to the same stem class are being replaced by the stem. For example, the terms 'compute', 'computed', 'computing', 'computer', 'computation' etc. are being replaced by 'comput'. So, the documents having all these terms will be replaced by the same stem term by the stemming algorithm resulting in higher document scores, which will eventually result in a different ranked list.

On the other hand, in-case of non-stemmed run, each term is distinctly considered. For example, 'computer', 'compute', 'computed' etc. will be considered as different terms. As these documents and queries are not stemmed, each term will have a different weightage.

In case of the first query, if we consider the top 20 documents retrieved by the BM25 model during stemmed and unstemmed runs, we find that there are fifteen common documents.

In case of the second query, for stemmed and unstemmed runs, we find that only nine common documents. This happens because stemming has considerable impact on query terms.

3.8 Statistical Tests

To compare the performances between different retrieval models, we performed t-test with one baseline model and an improved model. Null hypothesis for each pair is “there is no difference in the effectiveness of the two given models”, while an alternate hypothesis is that “improved model is more effective than baseline model”. To check these hypothesis we took the average precision value for each query of both the models and applied the following formula to calculate its t-value:

$$t = \frac{\overline{B - A}}{\sigma_{B-A}} . \sqrt{N}$$

where, $\overline{B - A}$ is the mean of difference between average precision of two models, σ_{B-A} is the standard deviation of the differences and N is the number of queries.
After calculating t-value for each pair of models, we calculated their *p-value*[3] using third party libraries numpy and scipy. If the p-value for a pair is <= 0.05 than the null hypothesis is rejected in support of alternate hypothesis.

3.9 Snippet Generation

Snippet generation[1] for a document is performed by ranking each sentence in a document using a significance factor (using Luhn’s Law) and the top ranked sentences are used for the summary, which is computed as :

(No. of Significant terms) ^ 2 / Total no. of terms

where the number of significant terms and the total number of terms are obtained for a text span which is the lower and upper bound for significant terms occurrence in a sentence.

[3] *Is the probability for a given statistical model when the null hypothesis is true.*

4 CONCLUSION

Mean Average Precision values for TF-IDF, BM25, and Lucene :

TFIDF:	0.231883632146
BM25:	0.470553281189
Lucene:	0.341920002675
TFIDF_PseudoRelevance:	0.227946274959
BM25_PseudoRelevance:	0.468365693091
TFIDF_Stopping:	0.230674416304
BM25_Stopping:	0.459029447728

Mean Reciprocal Rank values for TF-IDF, BM25, and Lucene :

TFIDF:	0.413422969453
BM25:	0.694029017857
Lucene:	0.57709036045
TFIDF_PseudoRelevance:	0.390215
BM25_PseudoRelevance:	0.681293402778
TFIDF_Stopping:	0.411532213446
BM25_Stopping:	0.697544642857

Based on the above results, it can be observed that *BM25 with stopping* performs better than other approaches. Hence, BM25 with stopping is used for snippet generation.
All other results are placed in the Project folder.

5 OUTLOOK

Our project incorporates the basic features of an information retrieval system. However, certain features and functionalities as mentioned below can be included in future to hone the performance of the search engines:

1. Query Logs: Use of query logs can improve the performance of the retrieval system.
2. Multithreading: Using multiple threads can drastically improve the response time of the system.
3. Compression Techniques: As the corpus grows, it will be essential to use different compression techniques like ‘Elias-γ encoding’, ‘v-byte encoding’ to store the inverted index.

6 BIBLIOGRAPHY

[1] W. Bruce Croft, Donald Metzler, and Trevor Strohman: Search Engines: Information Retrieval in Practice, Pearson Education Inc., 2015

[2] BM25: In Wikipedia. Retrieved on April 16, 2017, from https://en.wikipedia.org/wiki/Okapi_BM25

[3] Cornell: BM25 http://www.cs.cornell.edu/courses/cs4300/2013fa/lectures/retrieval-models-2-4pp.pdf

[4] MAP: In Wikipedia. Retrieved on April 17, 2017, from https://en.wikipedia.org/wiki/Information_retrieval#Mean_average_precision

[5] MRR: In Wikipedia. Retrieved on April 17, 2017, from https://en.wikipedia.org/wiki/Mean_reciprocal_rank

[6] BeautifulSoup: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[7] Apache Lucene: https://en.wikipedia.org/wiki/Apache_Lucene

[8] SimpleAnalyzer: https://lucene.apache.org/core/4_0_0/analyzers-common/org/apache/lucene/analysis/core/SimpleAnalyzer.html