UNDER GUIDENCE OF:

PROF. NADA NAJI

INFORMATION RETRIEVAL

SEMESTER I, FALL 2016

December 9, 2016

ROHIT RAJ, SAMANJATE SOOD, AND Sumit BHANWALA

1. INTRODUCTION

The goal of this project is to design and build an Information retrieval system. The performance of this system is then evaluated and compared in terms of retrieval effectiveness.

The project has two phases. The first phase is indexing and ranking the corpus. The second phase evaluates the results of the first phase.

* 1. Phase I

The first task of the project is to build four different retrieval modules namely Lucene, BM25, *tf.idf*, and cosine similarity. These programs are then run on the CACM test-collection. This gives use 4 different set of results.

We then use BM25 as the base search engine and perform query expansion using Pseudo relevance feedback approach to produce another set of result.

For the next task, we use BM25 as our search engine and perform Stopping based on the common words list given. These words are neither considered for document scoring nor considered as a query term. This gives us the sixth and final set of results for this phase. In the same task, we also index the stemmed corpus and run the stemmed queries on this index.

* 1. Phase II

Here we produce another set of results by performing stopping on the BM25 model with query expansion. We then evaluate the seven results. The performance assessment is done in terms of retrieval effectiveness. We take into consideration effectiveness matrices like MAP, MRR, P@K, and Precision & Recall tables to evaluate the performance of the seven distinct runs.

* 1. Snippet Generation

For the final task, we generate a snippet of the relevant documents based on the query which is loosely based on Lunn’s text summarization technique [1].

1. LITERATURE AND RESOURCES
   1. Resources

Most of the approaches used for the implementation of this project are based on the concepts and techniques explained by W. Bruce Croft et al. in the book Search Engines, Information Retrieval in Practice.

* 1. Phase I – Task 1
     1. Lucene [2]

This is a java program that uses three external Lucene.jar files to perform functionalities like indexing, parsing, and retrieval of documents based on the given queries. We have used the java Lucene libraries to perform search using the queries on the test collection provided, CACM [3].

* + 1. *tf.idf* [4]

For this model, we calculate the query term weight and the document term weight. The sum of the product of all such terms is take. To calculate the weights, we simply calculate the term frequency of the term in the document and divide it by the summation of the frequencies of all the terms in the document. This is then multiplied by the inverse document frequency which is the logarithm of the total number of document in the collection divided by the number of documents that term occurs in.

* + 1. Cosine Similarity

For this model, we take the dot product of the term weights for the matching query and document terms. This is then normalized by dividing the dot product with the product of the length of two vectors [7].

* + 1. BM25

This model extends the scoring function for the binary independence model to include document and query term weights [5]. Though this is not a formal model it is known to perform well in TREC retrieval experiments [6]

* 1. Phase I – Task 2

To perform query expansion, we have selected BM25 as the baseline run. Next, we have implemented a combination of Pseudo Relevance Feedback and *Rocchio algorithm* [8] to increase the size of the query. The relevance information required for the implementation of Rocchio algorithm, we use pseudo relevance feedback technique. And with use of this information we have applied the technique explained by Bruce Croft et al. [9] initial weights in the query vector Q is modified to produce a new query vector Q’.

As per W. Bruce Croft et al. in the book Search Engines, Information Retrieval in Practice [10], BM25 has performed very well in TREC retrieval experiments and has influenced commercial ranking algorithms.

Rocchio algorithm models a way to incorporate relevance information [11]. Rivas et al. Have found the use of BM25 to be superior [13].

* 1. Phase I – Task 3

In this phase two additional runs have been done by incorporating stopping and stemming using the provided stop list (common\_words.txt) and on the stemmed version of the corpus (cacm\_stem.txt).

For run 1 (BM25 with stopping), when the index is being read the terms of the index that also belong to list of common words, we will be ignoring such terms. Query terms will also be read with the similar restriction. Following these activities, we would successfully have removed the stop words and BM25 algorithm is run to fetch the results.

For run 2 (BM25 on stemmed corpus), given the corpus, we had to do a bit of additional setup before the actual run. Step 1 was to use the indexer, from HW3, to generate tokenised documents for the given stemmed documents. Step 2, from the tokenised documents an index is generated using the indexer created in HW3. Now that we have an inverted index for the stemmed corpus, and stemmed queries, we used this data to perform BM25 run and fetch the results.

* 1. Phase II – Evaluation

For all the all runs performed in previous sections mentioned above, we have evaluated those results in terms of effectiveness. As per the requirement specifications, the parameters used for effectiveness measure are Precision and Recall, Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and P@K (for k = 5 and 20).

Referring the relevance document provided with the data set, precision and recall is identified for the retrieved results in each query, and further calculations are performed and analysed get the search engines effectiveness.

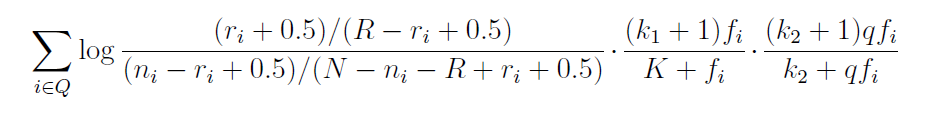
* 1. Snippet Generation

Since the size of the documents in the given corpus is small, a Luhn’s method [14] to generate document summary was not very helpful. Therefore, to identify a significant word we are performing stopping on the query terms, next we are picking up sentences form the top 10 ranked documents generated from search one of the search engine runs above. Based on number of occurrences of terms from the stopped list of query words each of the sentences are weighed and top two sentences are picked to be a part of the snippet. Due to small size of the documents in the corpus, and several extracted results showed that this method has produced meaningful snippets.

1. IMPLEMENTATION AND DISCUSSION
   1. Phase 1 – Task 1
      1. BM25 Ranking algorithm

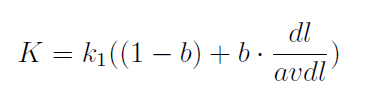
Methodology:

The form of this algorithm that has been implemented in this implementation is as below [14]:



R = Total number of relevant documents in the collection, ri = Number of relevant documents containing the term i, N = Number of documents in the collection, ni = Number of documents in the collection containing the term i, fi = frequency of term i in the document, qfi = frequency of term i in the query, k1, k2 = whose values are set empirically, as given by Croft et al. [14]

K = is used to normalise the *tf* component by document length given by the formula,



dl length of the document, avdl average length of a document in the collection, b is a parameter used for length normalisation.

Implementation:

We have used the above given equations to calculate the score of each document in the corpus in response to a query at a time.

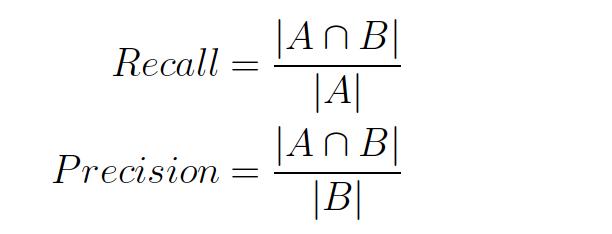
The inverted index is loaded from a text file and into a hash map (dictionary data structure in Python), with key as the index term and value as another hash map containing key as document ID and value as term frequency.

During the process of loading the

* 1. Phase 1 – Task 2
  2. Phase 1 – Task 3
  3. Phase 2 – Evaluation:-

**Implementation: -**

We have used recall, precision ,mean average precision , mean reciprocal rank as the parameters for evaluating our different search engines where recall and precision are defined as



where A is the set of relevant documents for a given query

B is the set of retrieved documents and |A intersection B| are the set of documents which are both relevant and retrieved.

**Average precision** can be defined as a technique to summarize the ranking by averaging the precision values from the rank position where a relevant document was retrieved.

**Mean average precision** is to average the precision from multiple queries.

**Reciprocal rank** can be defined as (1 / rank) where rank denotes the rank where the first relevant document was retrieved.

**Mean Reciprocal rank** is the average of reciprocal ranks over a set of queries.

In our implementation we have stored the data from the cacm

* 1. Snippet Generation

BIBLIOGRAPHY

[1] Search Engines, Information Retrieval in Practice – W. Bruce Croft et. al. page 216

[2] Lucene Version 4.7.2 Documentation - <http://lucene.apache.org/core/4_7_2/>

[3] CACM document collection - <http://www.search-engines-book.com/collections/>

[11] <http://nlp.stanford.edu/IR-book/pdf/09expand.pdf>

[12] <https://en.wikipedia.org/wiki/Rocchio_algorithm>

# [13] Study of Query Expansion Techniques and Their Application in the Biomedical Information Retrieval - [A. R. Rivas](https://www.ncbi.nlm.nih.gov/pubmed/?term=Rivas%20AR%5BAuthor%5D&cauthor=true&cauthor_uid=24723793), [E. L. Iglesias](https://www.ncbi.nlm.nih.gov/pubmed/?term=Iglesias%20EL%5BAuthor%5D&cauthor=true&cauthor_uid=24723793), and [L. Borrajo](https://www.ncbi.nlm.nih.gov/pubmed/?term=Borrajo%20L%5BAuthor%5D&cauthor=true&cauthor_uid=24723793)

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3958669/>