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.

* 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 Luhn’s text summarization technique [1].

* 1. **Contributions**

As a Team:

Extensive brainstorming sessions to discuss the concepts related to the techniques involved. Discussion on the results of phase 1 task 1, to select one model to apply query expansion on it. Also, decisions were made as a group over the outputs to be produced for each task and report design.

Rohit Raj

* Implemented the BM25 model – Part of Phase I task 1
* Incorporated Query expansion using Pseudo relevance feedback technique in BM25 – Phase I task 2

Documentation:

* Introduction - parts
* Literature and Resources
* Implementation and Discussion – 3.1.5 and 3.2
* Conclusion and Outlook

Sumit Bhanwala

* Implemented the evaluation of the different search engine while calculating the Precision, MAP, Recall and MMR, and P@K. – Phase 2
* Implemented the Bonus task for generating snippets from the query results – Bonus Task
* Implemented the tf.idf model - Part of Phase I task 1

Documentation:

* ReadMe.txt for Phase2 of the solution, Bonus part, Overall Project’s,

Phase 1’s task 2, and phase 1’s task 3

* Implementation and Discussion – 3.4, 3.5,
* Conclusion and Outlook
* Results

Samanjate Sood

* Implemented and optimized the Cosine similarity code and implemented Lucene codes – Part of Phase I task 1
* Used BM25 module to perform stopping and stemming – Phase 1 Task 3
* Combined Phase I task 2 with stopping – Phase II task 1 (for seventh run)

Documentation:

* Implementation and Discussion – 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.3
* Introduction – parts
* Conclusion and Outlook
* Bibliography

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 Vector Model**

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 Ranking Algorithm**

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).

1. 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.
2. 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. **Tokenizer and Indexer**

Given a new corpus we reused the indexer from previously created was reused. And new tokenizer was created before that to parse and clean the document and feed it as an input to the indexer.

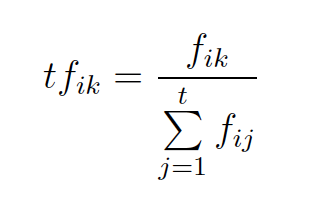
* + 1. **Lucene**

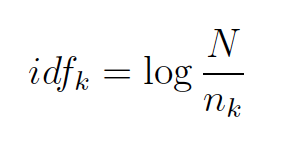
The program uses Lucene Version 4.7.2. We first import the three external jar files. We use the inbuilt SimpleAnaylzer, IndexReader, IndexSearcher, TopScoreDocCollector, and QueryParser classes from the references to the external libraries to tokenize, index, and rank the documents for the given queries.

* + 1. ***tf.idf***

The *tf.idf* program first loads the inverted index in a hash map. Similarly, the query terms are loaded in a different hash map. To calculate *the tf.idf* score of a document we take the product of the term weights of the document for the terms of the query that occur in the concerned document. For each query term the result is summed to generate the final document score.

calculate the *tf* for that term in the document and multiply it by the *idf*.

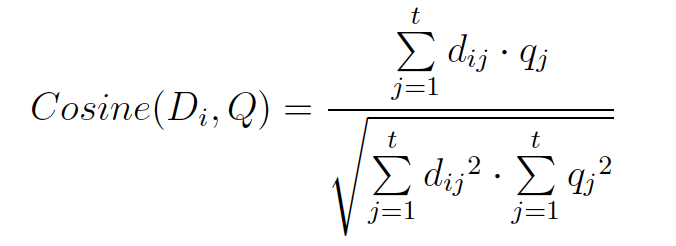


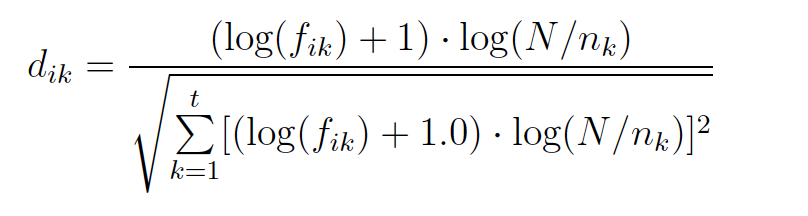


where *idfk* is the inverse document frequency weight for term *k*, *N* is the number of documents in the collection, and *nk* is the number of documents in which term *k* occurs.

* + 1. **Cosine Similarity Vector Model**

The implementation of this model is based on the tf.idf implementation. We represent the document and query as a vector for all the query terms. These are stored in two different hash maps. The following formula is used to compute the dot product and compute the final score of the document.

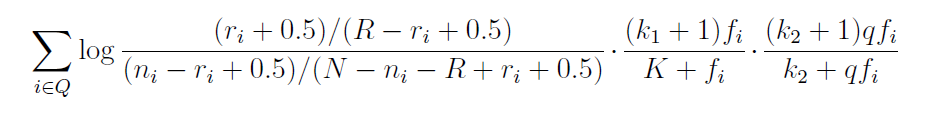
The multiplication of the two vectors stored in the hash map gives us the numerator. To normalize the score, we need to calculate the term weights of all the terms in the documents as well. This is pre-calculated and stored in a different hash map. To calculate the term weight for the document and the query, we use the following formula:

Thus, we take the square of all the document weight vectors and query weight vectors represented as a hash map in the program. These two values are then multiplied and the square root of this value gives us the final cosine score of the document for the given query stored in a hash map.

* + 1. **BM25 Ranking Algorithm**

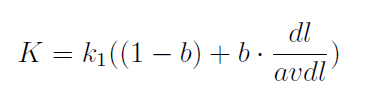
Methodology:

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.

From the given relevance information (cacm.rel.txt) the relevance information is recorded for each query result to calculate *R* and *ri*. at the time of processing.

Once all the documents are loaded in to the inverted index, *avdl* is calculated.

Now, for each document having any of the query terms, a score is calculated using the formula given above. Using this result the documents are ranked for each query and results are published.

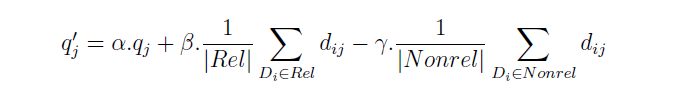
* 1. **Phase 1 – Task 2**
     1. **Query Expansion**

The baseline run selected from the previous task for query expansion is *BM25 ranking algorithm*, since this model focuses on topical relevance [15], we expect it to return topically similar results for a query.

Methodology:

This task builds upon the BM25 ranking algorithm implementation, and combines it with *Rocchio algorithm* (Rocchio, 1971) to expand the query. From this expanded query, another BM25 run is made to generate the final document list for each query. We have chosen *pseudo relevance feedback model*, to decide on the relevant documents after the first run of BM25 ranking algorithm. We are selecting some top-ranked documents from the first run to be in the relevant set.

*Rocchio algorithm:*



where *qj* is the initial weight of query term j, *Rel* is the set of identified relevant documents, *Nonrel* is the set of non-relevant documents, |.| gives the size of a set, *dij* is the weight of the jth term in document i, and α, β, and γ are parameters that control the effect of each component [16].

Implementation:

Assumption: α = 8; β = 16; γ = 4, based on W. Bruce Croft’s et al. conclusions [18].

The setup for BM25 ranking algorithm remains unchanged. Additions are made to incorporate query expansion using *pseudo relevance feedback*, and then using *Rocchio algorithm* to expand the query, which will be eligible for second sun of BM25 ranking module to generate new ranks.

In this implementation, we have chosen the to consider top 5 documents from the results of first run of BM25 to be relevant. This decision was made based on the observation that as we kept the threshold at a higher number, we ended up adding more and more terms from the query that may be from non-relevant documents, thereby reducing the topical relevance.

Terms of the query are loaded into a list. To maintain time efficiency, we have considered all the terms from the relevant documents only to be part of the vocabulary, since it is very unlikely that a term from non-relevant documents will get a high score as part of *Rocchio algorithm*. Now for each term in the vocabulary, a score is calculated using the aforementioned – *Rocchio algorithm,* and top 10 words, that aren’t already part of the query are added to the list of the query terms. Completing the task of query expansion. Using this expanded query, BM25 we produce second run results.

* 1. **Phase 1 – Task 3**

1. **Stopping**

Here we take a list of common words which does not contribute towards the document scoring. We use the BM25 model and then load the common words given in the list (provided for this task). Then while loading the index, we ignore the words that are present in the common words list and not index them at all. Similarly, while loading the query terms list, we check whether that word exists in the common word list. Thus, we run the BM25 without taking into consideration these words and retrieve the documents.

1. **Stemming**

For this process, we are take the separate corpus and query list (provided for purpose of this task). We first tokenize the corpus and store the tokens of one document in a different file. The indexer then indexes these tokenized documents and gives a new index. Next, we use this index and feed it to the BM25 search engine and run the stemmed queries to retrieve the document ranked list.

Query-by-Query analysis.

For query: Applied stochastic processes (appli stochast process)

The top five retrieved documents are:

With stemming

CACM-1696

CACM-0268

CACM-1410

CACM-2535

CACM-1194

Without stemming

CACM-1696

CACM-0268

CACM-1410

CACM-2882

CACM-1540

For query: portable operating systems (portabl oper system)

The top five retrieved documents are:

With stemming –

CACM-3127

CACM-1591

CACM-1680

CACM-1033

CACM-3068

Without stemming –

CACM-3127

CACM-2246

CACM-3068

CACM-1930

CACM-1461

Here we can see, that for the first query there is not much difference between the ranking is not much different from each other. Hence we could say that the effect of stemming is rather limited.

However, stemming could have an adverse effect on the effectiveness of the search engine as seen in the second query as it retrieves non-relevant documents at higher ranks (considering the relevance information as given in the camc.rel .txt file). Thus, we should avoid stemming, at least for languages such as English where the effect is limited and stemming could adversely affect the search results.

* 1. **Phase 2 – Evaluation**

Implementation

We have used recall, precision, mean average precision (MAP), mean reciprocal rank (MRR) as the parameters for evaluating our different search engines.

In our implementation, we have stored the data from the cacm queries and stored them in a dictionary where key of the dictionary is query id and its values are the relevant documents. For each of the search engine run we store the data in a dictionary with a key as query id and values as the set of relevant and irrelevant documents.

After we have both the dictionaries, for each query id, we calculated precision,

Recall, at each rank reciprocal rank, and stored the respective results in dictionary data type.

In the same way, we stored reciprocal rank in a dictionary where key is the query id and value denotes the reciprocal rank.

After we have the recall dictionary and precision dictionary for each query id we calculated average precision for each query and then averaged the results over all the queries and got MAP and in the same way we calculated MRR.

* 1. **Snippet Generation**

For generating snippets, we have used *query-dependent* [17] techniques. And while generating the snippet we make sure that the *title* is there in the generated snippet and the query terms are highlighted, we are *uppercasing* the query terms which appear in the snippets. For each query, we generated snippets for the top 10 retrieved documents and as content was very less in the document body so we had to modify the significant word definition per the need of the corpus.

From the document body, we created a list of sentences and then each sentence was weighted per the query term and the frequency of the query terms in a sentence. Stopped Query terms were used for weighing the sentence and then in the end the top three sentences which had the highest weight were used in the snippet. Rather than using phrases in the snippet we used entire sentences as it makes the snippet more readable.

Also, generally the last line of the snippet contains the document id for which snippet is generated as we don’t have the hyperlink available for the same document.The general syntax for snippets are:

Query ID:

Query Text:

Snippets:

Snippet1

Snippet2

Up to 10 snippets for each query

1. RESULTS

Taking the generated results from each task through the given evaluation matrix. We have arrived at the below given results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Run/  Measure | BM25 | BM25 with Query Expansion | BM25 with stopping | BM25 with query expansion & stopping | CSVM | Lucene | *tf.idf* |
| MAP | 0.366 | 0.282 | 0.342 | 0.327 | 0.319 | 0.283 | 0.215 |
| MRR | 0.822 | 0.686 | 0.752 | 0.729 | 0.758 | 0.689 | 0.546 |

Table 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Metric/  Filename | MRR, MAP & P@K k = 5, 20 | Precision | Recall |
| BM25 | MRR\_BM25.txt | precision\_dict\_cacm\_queries\_bm25.csv | recall\_dict\_cacm\_queries\_bm25.csv |
| BM25 with query expansion | MRR\_BM25\_qe.txt | precision\_dict\_cacm\_queries\_bm25\_qe.csv | recall\_dict\_cacm\_queries\_bm25\_qe.csv |
| BM25 with stopping | MRR\_BM25\_stop.txt | precision\_dict\_cacm\_queries\_bm25\_stop.csv | recall\_dict\_cacm\_queries\_bm25\_stop.csv |
| BM25 with query expansion and stopping | MRR\_BM25\_qe\_stop.txt | precision\_dict\_cacm\_queries\_bm25\_stop\_qe.csv | recall\_dict\_cacm\_queries\_bm25\_stop\_qe.csv |
| CSVM | MRR\_csvsm.txt | precision\_dict\_cacm\_queries\_csvsm.csv | recall\_dict\_cacm\_queries\_csvsm.csv |
| Lucene | MRR\_lucene.txt | precision\_dict\_cacm\_queries\_lucene.csv | recall\_dict\_cacm\_queries\_lucene.csv |
| *tf.idf* | MRR\_tfidf.txt | precision\_dict\_cacm\_queries\_tfidf.csv | recall\_dict\_cacm\_queries\_tfidf.csv |

Table 4.2

Due to the large amount of data we deemed it better not to add all the results here. MAP and MRR are among the prominent measures, which we have put in Table 4.1 above. Spreadsheets related to all the other evaluation metrics is attached along with the report. And detailed a Table 4.2

For Individual task results of each phase in output files are provided in their respective folders, and readme.txt gives the details on the output file. Same is applicable for the bonus task as well.

1. CONCLUSIONS AND OUTLOOK

Conclusions

As per our observation, BM25 works the best out of all the retrieval methods. It is evident by the fact that MAP for BM25 is much higher than the other retrieval module which proves that BM25 retrieve more relevant documents at higher ranks also the same is supported by the MRR which is approximately 0.82 supporting the observation that it retrieves a relevant document at very higher ranks.

Also, we can see that the addition of *stopping on BM25 with query expansion*, improves the MAP and MRR values of the algorithm.

A simple implementation of *tf.idf* algorithm, gives the poorest result among all other techniques, this can be attributed to the fact that it doesn’t take into consideration, any topical relevance for retrieving documents.

BM25 is performing better than BM25 with query expansion. Although, with addition on pseudo relevance feedback and *Rocchio algorithm* to include topical relevance. This can be attributed to the fact that pseudo relevance takes some of the top documents to be relevant and bases the further retrieval processing on that assumption. This assumption may not be of benefit if the top documents on the first run itself are not of the relevant documents.

Outlook

Incorporating Machine learning approach to BM25 retrieval model can improve retrieval effectiveness [18]. We can also provide a user interface, from which a user can easily provide relevant documents, and this information can further be used to improve upon the relevance feedback and generate the result.

We can also incorporate a spell check to improve search results if there are any misspelled words in the query that can already be taken care of.

We can also classify the retrieved documents into a set of clusters, this will help us to provide similar documents easily for such a request.

1. 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/>

[] <https://www.microsoft.com/en-us/research/publication/a-machine-learning-approach-for-improved-bm25-retrieval/>