Information Retrieval Systems

An Evaluation and Comparison

(Submitted in partial fulfillment of the requirements of Northeastern University's CS 6200 Fall 2016 course, taught by Prof. Nada Naji)

Shail **Shah**
Drashti **Bhuta**
Eswara Sai Surya Teja **Gummalla**

1. Introduction

Throughout the Fall of 2016, the authors were exposed to various core Information Retrieval concepts. In this project, an attempt has been made to put the concepts into practice by building, evaluating and comparing different search engines.

The goal of this project is to build IR systems, make variations to improve the baseline runs, and compare their performance levels in terms of retrieval effectiveness.

The project is mainly coded in Python and Java. Excel has been used for analysis and data visualization. Libraries that are used include BeutifulSoup, jSoup, and Lucene. PyCharms and IntelliJ by JetBrains are the IDEs that were used to aid in coding. Github has been used for version control.

This work on this project has been planned and executed by Shail Shah, Drashti Bhuta and Eswara Sai Surya Teja Gummalla. All of them actively studied various resources to implement IR systems.

Shail was responsible for cleaning the given files, making the Lucene IR system, evaluation of search engines and the documentation. Drashti was involved in making the BM25 algorithm, query expansion, removing stop words from the corpus, and snippet generation. Eswara coded the indexer, tf-idf and cosine similarity IR systems.

2. Literature and Resources

The techniques used in this project are:

1. **Indexing** using unigrams
   Indexing refers to the process of collecting, parsing and storing data for faster retrieval.

2. **Retrieval Modelling** with BM25, tf-idf, Cosine Similarity and Lucene
   A Retrieval model is a system that returns a list of relevant documents from the corpus, given a query.

3. **Stopping** using a stop list
   Stopping is used to remove stop words - words that are not of significance to the searching process. If stopping is performed on the corpus, it must also be performed on the query.

4. **Stemming** using a stem list
   Stemming reduces words to their root form. Like stopping, stemming must be performed on both the corpus and the query, if at all it is applied.

5. **Query Expansion** using pseudo-relevance feedback
   Query expansion reformulates the seed query by adding terms that would result in better results to being retrieved.

6. **Snippet Generation** using static summary
   Snippet generation is used to preview a page that is in the search results, so that the user can get an idea of the page.

The tools that were used in this project are:

1. PyCharms by JetBrains
   It is an integrated development environment for Python

2. IntelliJ by JetBrains
   It is an integrated development environment for Java

3. Github Desktop by GitHub, Inc.
   It is a software for source code management and collaboration

4. Lucene Core by The Apache Software Foundation
   It is a full-featured text search engine library written in Java

5. Excel by Microsoft
   It is used for visualizing the data to make evaluation simpler

The scholarly work and research articles that were referenced include:

1. Selecting Effective Expansion Terms for Better IR
   This paper discusses different ways of selecting and ranking co-occurring terms and to suggests use of information theoretic measures (KL Divergence) for ranking the co-occurring terms.

2. Query Expansion by Pseudo Relevance Feedback
   The paper describes the BM25 formulation for document-query similarity measure and describes approaches for query expansion (pseudo relevance feedback, Google retuned documents, synonyms

defined by WordNet, and random walk model), along with their evaluations.

3. **Query Expansion Using Term Distribution and Term Association**
   This paper illustrates expansion of a query by using a distribution based method to get candidate terms and subsequently refining based on the association terms' strength with original query terms.

4. **Query Expansion using Local and Global Document Analysis**
   This paper talks about automatic query expansion using techniques that analyze the corpus to discover word relationships and those that analyze the documents retrieved by the initial query.

5. **A Probabilistic Analysis of the Rocchio Algorithm with TF-IDF for text Categorization**
   This paper is about the Rocchio relevance feedback algorithm. The probabilistic analysis gives theoretical insight into the word-weighting scheme and similarity metric used in the algorithm.

6. **Generation of Document Snippets Based on Queries and Search Results**
   This patent discusses scoring paragraphs including the query terms based on the length of the paragraph and the distance of the paragraph from a location of the document. It also elaborates on selecting a snippet generating algorithm based on the type of query.

3. Implementation and Discussion

Cleaning was the first task at hand. Starting with the corpus, the HTML tags were removed. Then, punctuation was removed using regular expressions. Occurrences of "-" were not discarded. For numbers, ",", "." and ":" were also retained. Numbers at the end of the documents seemed non-relevant and thus they were removed.

The given 64 queries were extracted using an XML extractor and put in a new file. Each line had one query, so the new file was of 64 lines. The cleaning that was done for the corpus was done for the queries too.

A single file was provided for the stemmed corpus. The single file was converted into multiple files, one per document. Thus, we had a folder container 3,204 stemmed documents. The stemmed queries were not needed to be modified.

Stop words were removed from the corpus. For each document in the clean corpus, a new document was created that only contained words that were not in the stop words list provided.

Next, the tf-idf scores were calculated for each document in the corpus. The formula used was

$$tfidf(d) = \log \sum \left(\frac{N}{n_i}\right) \times \frac{tf}{dl}$$

where N is the number of documents in the corpus, n_i is the number of documents in which the term occurs in, and dl is the document length of the given document.

After this, the cosine similarity was calculated for each document. Next, Lucene was used to make an index, parse each query, and display the results. The raw queries and corpus were provided, as Lucene has an in-built method to handle raw documents. The code that was used in homework 4 was used here, with small modifications.

Later, a search engine with BM25 as a retrieval model was made. With relevance information provided in *cacm.rel,* values of r_i and R were obtained. The file did not have the relevance information for all the queries, so for queries with no relevance information, $r_i = R = 0$ was assumed.

The output of BM25 is fed to the Query expansion module which uses KL Divergence to compute additional terms to add to the initial query.

After getting six distinct runs with results of all 64 queries, a seventh run was performed that combined cosine similarity with stopping.

BM25 was again applied on the corpus with no stop words and again to the stemmed corpus and queries to check how well it performs with this variation.

Finally, MAP, MRR, P@K (with K = 5 and 20), precision, and recall were used to calculate the performance of the search engines.

KL Divergence was chosen as the algorithm for expanding the query because it considers the probability distance of terms in relevant set(defined as top 15 documents of each query) and the corpus and giving the divergence between them. This divergence proved to be better for giving a good result after expansion with the same base engine, moreover the choice was further backed by computing various results including Rocchio and normal expansion.

Results of implementation :

Measure	Base (No expansion)	Rocchio (cosine similarity)	Regular	KLD(BM)
MAP	0.591	0.159	0.398	0.593
MRR	0.864	0.6	0.58	0.698

Based on these values and the paper 'Comparing and Combining methods for Automatic Query Expansion' by Jose R. and Lourdes Araujo shows the result for term selected by distributing scores over the one selected by co occurrence (cosine and dice). We considered the value of k=15 after considering the following results.

K	5	10	15	20	25
MAP	0.49	0.57	0.59	0.48	0.477
MRR	0.625	0.6551	0.693	0.65	0.63

The number of words to be added was chosen after following experiment results:

No. of Words	5	10	15
MAP	0.38	0.593	0.58
MRR	0.572	0.693	0.65

Thus, we selected number of words to be 10.

The approach proposed by Carpineto et al. [as suggested in "Query Expansion Using Term Distribution and Term Association by Dipasree Pal, Mandar Mitra and Kalyankumar Datta"] is used for calculating the probability distribution.

In this method, we consider the top 15 documents for each query as (pseudo) relevant set R and all terms in R are considered as the candidate terms t. Let C represent the whole CACM corpus. p_r and p_c represent the unigram probability distribution of the candidate term t in the relevant set and the corpus respectively. $tf(t,d)$ represents the frequency of the term in that document. The calculation for p_r and p_c and the term divergence between them is given below.

$$p_r(t) = \frac{\sum_{d \epsilon R} tf(t,d)}{\sum_{d \epsilon R} \sum_{t' \epsilon D} tf(t',d)};\ p_c(t) = \frac{\sum_{d \epsilon C} tf(t,d)}{\sum_{d \epsilon C} \sum_{t' \epsilon D} tf(t',d)},$$

$$S(t) = p_r(t) \times \log \frac{p_r(t)}{p_c(t)}$$

Terms for which this score is largest are selected for expansion.

We considered the Query term occurrence in each sentence of relevant documents. Relevant set compromises of the top 15 documents for each query as obtained by BM25 baseline run. Each sentence is assigned a score based a score based on the number of query term occurring in it. The two sentences with the maximum score are selected for representing the snippet for that document relevant to a query. Here we assume that while there may be some better words that can help differentiate the two sentences. But we consider that the user will be more interested for document whose snippet reflect more query terms. Thus we have

selected the query term occurrence to differentiate between sentences. This choice was inspired by the paper [SnipIt - A real time Dynamic Programing approach to Snippet Generation for HTML Search Engines by Ian Carr and Lucy Vasserman]. We represent a sentence only up to 100 characters for long sentences. These sentences have "…" at the end to denote that it has more content to it.

We read each document in the relevant set. store all the sentences in lower case in a dictionary. We eliminate the stop words from query to avoid its impact on the sentence score. We select the top ranked sentences.

Query highlighting is represented as * <query - term> * by traversing to the query term occurrence in the selected sentences and replacing it as the representation mentioned above.

Query-by-query analysis

The query "portable operating systems" becomes "potable oper system" when stemmed. Because "system" occurs in a lot of documents, a lot more documents match the stemmed query than the unstemmed query. So, many non-relevant documents are shown in the result.

The query "performance evaluation and modelling of computer systems" stems down to "perform evalu and model of comput system". The precision increases by 0.05 and the recall increases by 0.1. It may be because 'perform' and 'performance' basically mean the same thing, and all its verb forms would be stemmed down to 'perform' as well. Converting 'systems' to 'system' also helps as it broadens the search scope a bit.

3. Results

Here is a snapshot of the results of the performance of the seven search engines that were made. More granular details about the results can be found in the excel files.

1	Q0	CACM-1938	1	0.287399	COSINE-SIM2	NR	0	0
1	Q0	CACM-2319	2	0.262916	COSINE-SIM2	NR	0	0
1	Q0	CACM-1657	3	0.25995	COSINE-SIM2	NR	0	0
1	Q0	CACM-2379	4	0.240251	COSINE-SIM2	NR	0	0
1	Q0	CACM-2371	5	0.225384	COSINE-SIM2	NR	0	0

1	Q0	CACM-1938	1	0.037373	TF_IDF1	N	0	0	0.5
1	Q0	CACM-1657	2	0.03354	TF_IDF1	N	0	0	0.5
1	Q0	CACM-2371	3	0.029795	TF_IDF1	N	0	0	0.5
1	Q0	CACM-2319	4	0.025019	TF_IDF1	N	0	0	0.5
1	Q0	CACM-2629	5	0.024582	TF_IDF1	N	0	0	0.5
1	Q0	CACM-2796	6	0.024125	TF_IDF1	N	0	0	0.5

1	Q0	CACM-1657	1	32.14354	BM253	N	0	0
1	Q0	CACM-1938	2	30.28101	BM253	N	0	0
1	Q0	CACM-1410	3	30.12644	BM253	R	0.333333	0.2
1	Q0	CACM-2218	4	28.68626	BM253	N	0.25	0.2
1	Q0	CACM-1605	5	28.49329	BM253	R	0.4	0.4

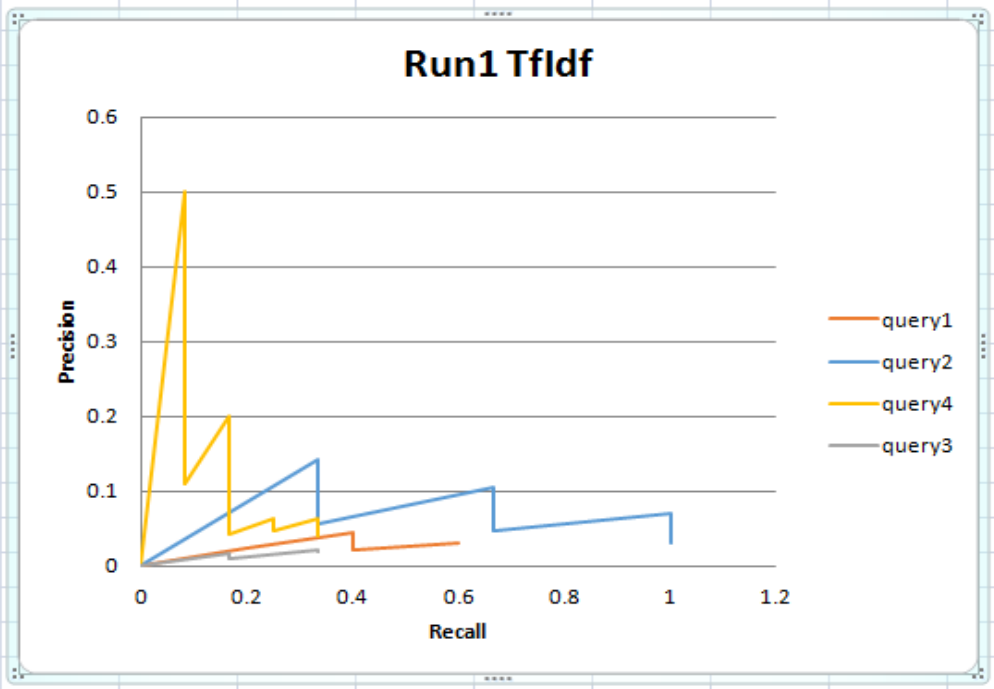
1	Q0	CACM-1657	1	0.337506	LUCENE4	N	0	0
1	Q0	CACM-1410	2	0.32017	LUCENE4	R	0.5	0.2
1	Q0	CACM-2319	3	0.310755	LUCENE4	N	0.333333	0.2
1	Q0	CACM-1938	4	0.290842	LUCENE4	N	0.25	0.2
1	Q0	CACM-1827	5	0.273018	LUCENE4	N	0.2	0.2

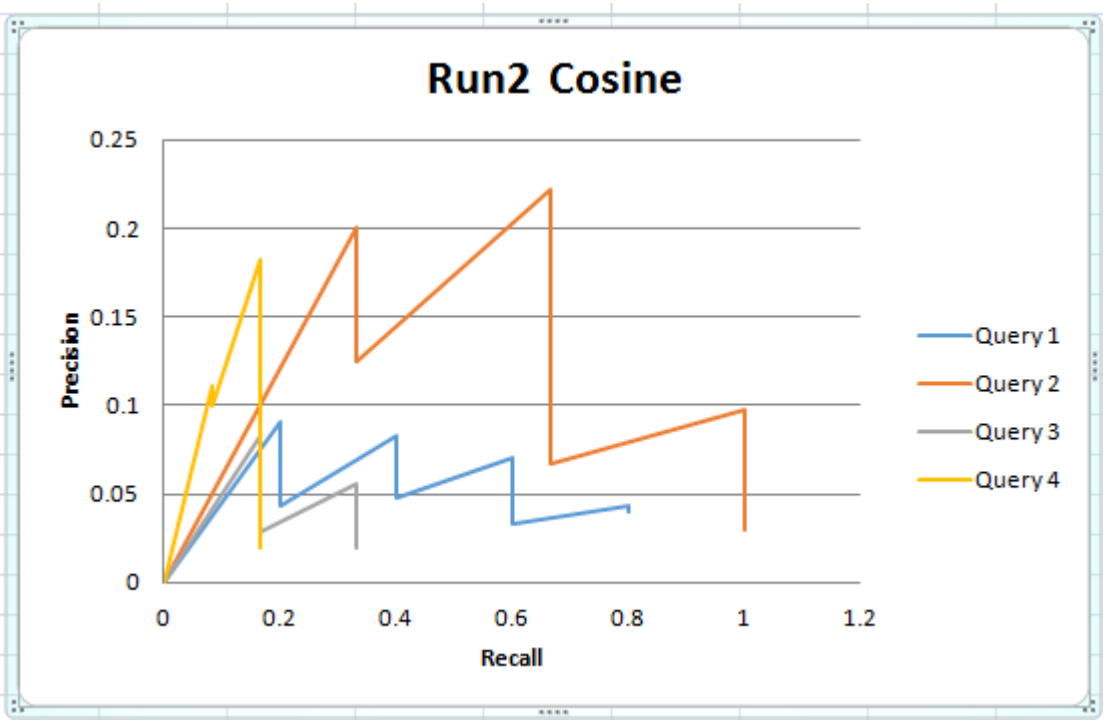
1	Q0	CACM-1410	1	25.65303	BM25_STOP5	R	1	0.2
1	Q0	CACM-1657	2	25.40408	BM25_STOP5	N	0.5	0.2
1	Q0	CACM-1938	3	24.34961	BM25_STOP5	N	0.333333	0.2
1	Q0	CACM-2218	4	23.16677	BM25_STOP5	N	0.25	0.2

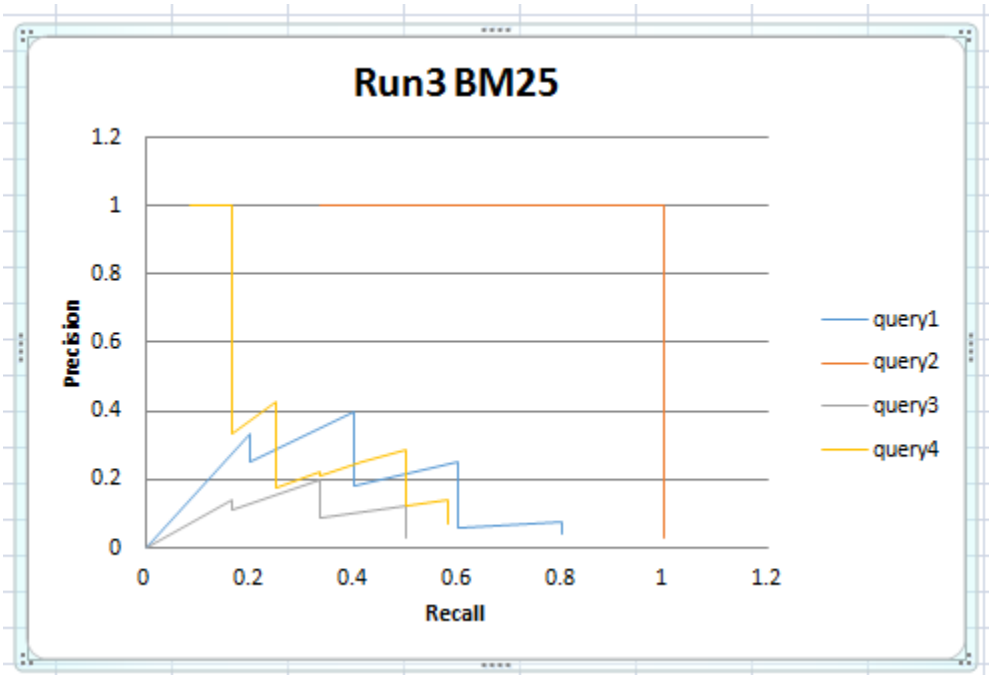
1	Q0	CACM-2371	5	22.47779	BM25_STOP5	N	0.2	0.2

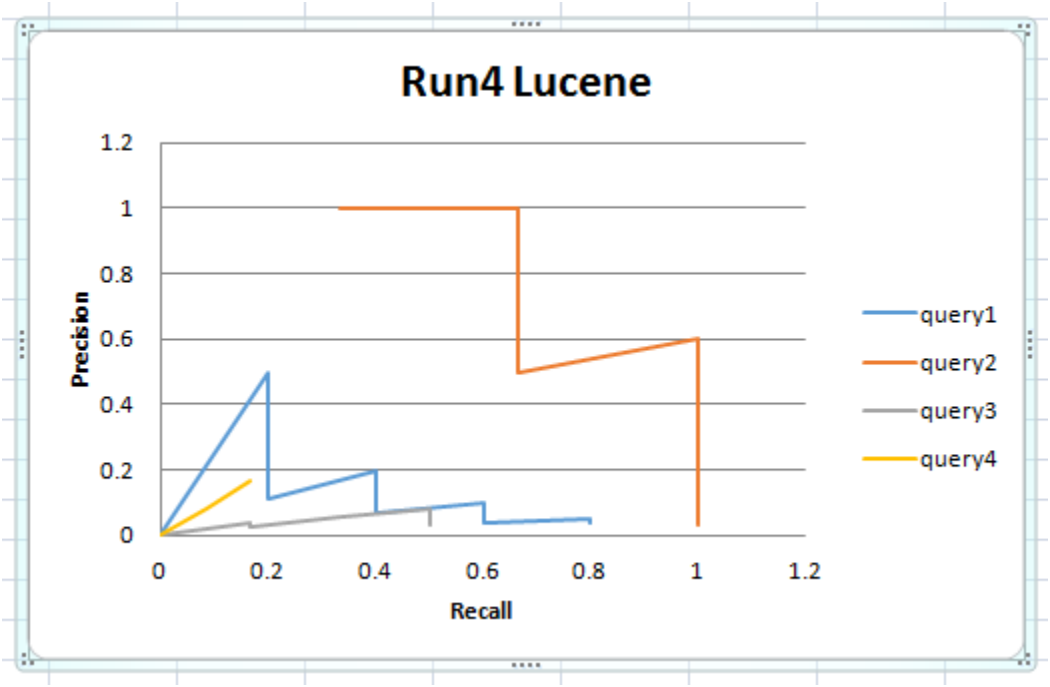
1	Q0	CACM-2947	1	47.17311	BM25EXP6	N	0	0
1	Q0	CACM-1657	2	35.96036	BM25EXP6	N	0	0
1	Q0	CACM-1938	3	30.28101	BM25EXP6	N	0	0
1	Q0	CACM-1410	4	30.12644	BM25EXP6	R	0.25	0.2
1	Q0	CACM-2218	5	28.68626	BM25EXP6	N	0.2	0.2

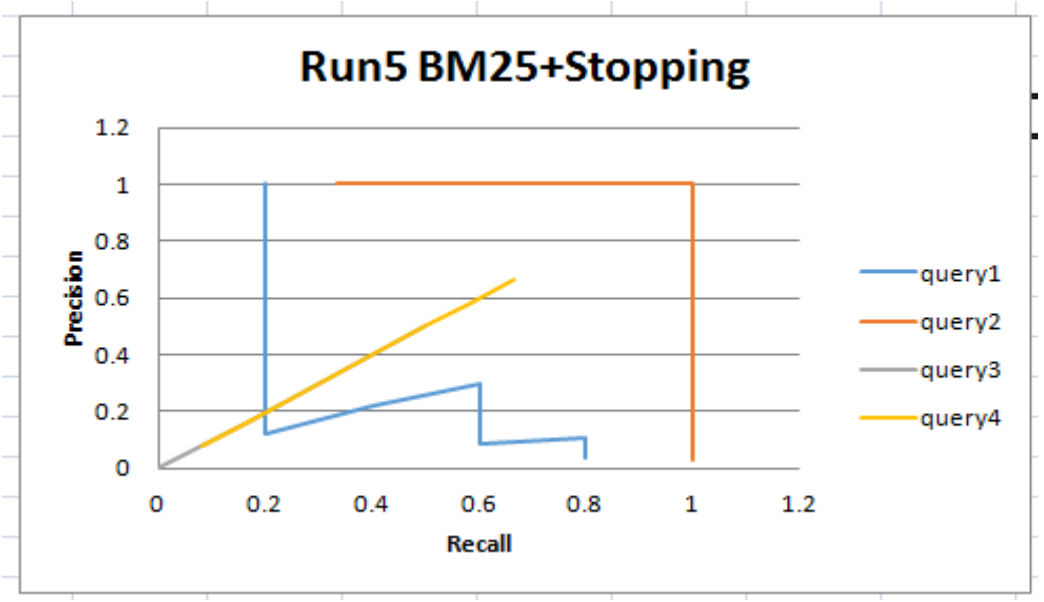
1	Q0	CACM-1938	1	0.323514	COSINE-SIM-STOP7	N	0	0
1	Q0	CACM-2319	2	0.299263	COSINE-SIM-STOP7	N	0	0
1	Q0	CACM-1657	3	0.292329	COSINE-SIM-STOP7	N	0	0
1	Q0	CACM-2629	4	0.268566	COSINE-SIM-STOP7	N	0	0
1	Q0	CACM-2371	5	0.262221	COSINE-SIM-STOP7	N	0	0

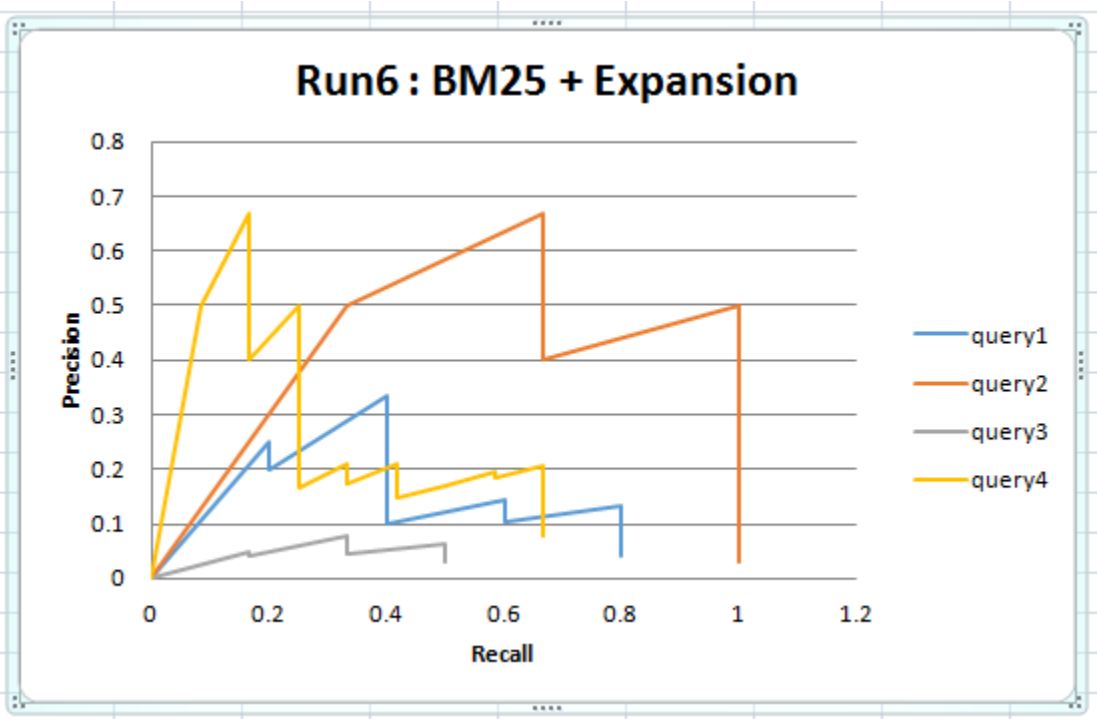


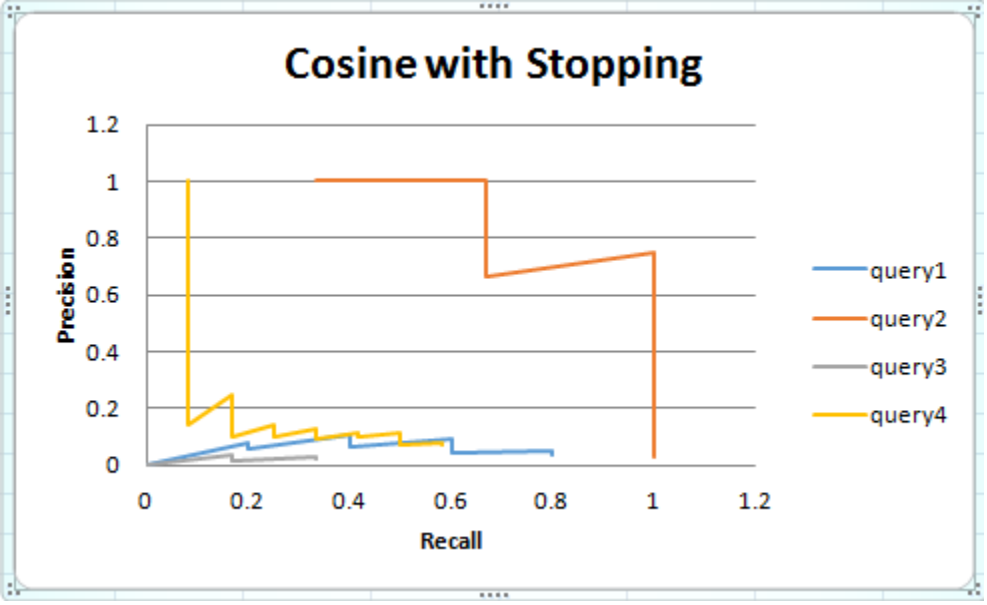

Run2 Cosine
Precision
Recall
0.25
0.2
0.15
0.1
0.05
0
0
0.2
0.4
0.6
0.8
1
1.2
Query 1
Query 2
Query 3
Query 4


Run3 BM25
Precision
Recall
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2
query1
query2
query3
query4


Run4 Lucene
Precision
Recall
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2
query1
query2
query3
query4


Run5 BM25+Stopping
Precision
Recall
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2
query1
query2
query3
query4


Run6 : BM25 + Expansion
Precision
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
1.2
Recall
query1
query2
query3
query4


Cosine with Stopping
Precision
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
1.2
Recall
query1
query2
query3
query4

5. Conclusions and Outlook

Summarizing the results of the runs,

RUNS	MAP	MRR	P@5	P@20
Run1 (tf-idf)	0.133854038462	0.486403846154	0.276923077	0.179807692
RUN2 (cosine)	0.149562115385	0.631557692308	0.338461538	0.203846154
RUN3(BM25)	0.590010270772	0.864926739927	0.511538462	0.302884615
RUN4(Lucene)	0.416184412755	0.689082354788	0.373	0.204
RUN5(Stopping + BM25)	0.585717928639	0.888621794872	0.5	0.311538462
RUN6(Query Expansion + BM25)	0.593802535887	0.69826007326	0.488461538	0.2875
RUN7 (Cosine + stopping)	0.165602008547	0.700711538462	0.392307692	0.216346154

From the results, BM25 triumphed as the best retrieval model for all the models we tried implementing. The precision and recall values were better than other models, and it was quick to give results too. Having a file which had data about relevant documents for different queries helped BM25 to perform better than others, as information about R and r can be deduced from it.

When we combine BM25 with query expansion the map value increased by 0.03 while the MRR value reduced significantly.

Even after experimenting with various k value the statistics didn't improve and thus expansion have not much of a positive effect on the result

The output form BM25 stemming was quiet good compared to the baseline run but it contains only 7 queries and its performance may degrade when dealing with the complete query set.

Cosine and tf-idf yields quiet poor result and from all these observation we conclude that bm25 baseline is better for the given corpus.

There is a lot of scope for improvement here. Automation of index updating, pre-processing frequent queries and keeping a log of user patterns are a few things which can be done in the future. Stemming can be replaced with Lemmatization. For bigger corpuses, parallel computing can be explored for maximizing resource utilization.

6. Bibliography

Iman, Hazra, and Aditi Sharan. "Selecting Effective Expansion Terms for Better Information Retrieval." International Journal of Computer Science and Applications 7.2 (2010): 52-64. Web. 4 Dec. 2016. <http://www.tmrfindia.org/ijcsa/v7i24.pdf>.

Feng, Zheyun. "Query Expansion by Pseudo Relevance Feedback." N.p., 27 Aug. 2012. Web. <http://fengzheyun.github.io/downloads/projects/before2015/document/DocRetri.pdf>.

Pal, Dipasree, Mandar Mitra, and Kalyankumar Datta. "Query Expansion Using Term Distribution and Term Association." N.p., 4 Mar. 2013. Web. 5 Dec. 2016. <https://arxiv.org/abs/1303.0667>.

Xu, Jinxi, and W. Bruce Croft. "4. Query Expansion Using Local and Global Document Analysis." N.p., 02 Oct. 2010. Web. 05 Dec. 2016. <http://www.eng.utah.edu/~cs7961/papers/XuCroft-SIGIR96.pdf>.

Joachims, Thorsten. "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization." N.p., 1997. Web. 4 Dec. 2016. <https://www.cs.cornell.edu/people/tj/publications/joachims_97a.pdf>.

Verstak, Alexandre A., and Anurag Acharya. Generation of Document Snippets Based on Queries and Search Results. Patent US 8145617 B1. 27 Mar. 2012. Print.

Carr, Ian, and Lucy Vasserman. "SnipIt - A Real Time Dynamic Programing Approach to Snippet Generation for HTML Search Engines." (n.d.): n. pag. Web.<http://www.cs.pomona.edu/~dkauchak/ir_project/whitepapers/Snippet-IL.pdf>.