INFORMATION RETRIEVAL FINAL PROJECT

Alexander Jo
Ravi Krishnan Iyer
Taashi Priya Khurana

CS 6200 Final Project

Spring 2018

Professor Nada Aladdin Jamel Naji

Introduction –

This project aims at developing a Search Engine. We have designed several retrieval systems like BM25, Lucene and tfidf. We run the codes for a given corpus to find out which of them gives a good result. We generate top 100 results for each query. We have used the indexer from our previous assignments.
The result of all retrieval systems is judged in the "Evaluation Stage" which helps to judge the performance and efficiency of each system.

Project contributions -

Phase 1

- Task 1:
    - BM25: Alex
    - tfidf: Alex
    - Smoothed Query Likelihood: Alex
    - Lucene: Ravi and Taashi
- Task 2 Query Expansion: Ravi and Taashi
- Task 3 Stemming and Stopping: Alex

Phase 2:

- Snippet Generation: Ravi and Taashi

Phase 3:

- Evaluation: Alex

Documentation: Alex, Ravi and Taashi

Alex took most of the work of the baseline implementations, including the parser, indexer, bm25, tfidf, and smoothed query likelihood implementations. These baselines were an adaptation of the previous homework assignments and were written in Rust. The three baseline implementations Alex wrote were also the three used for Task Three, stemming and stopping. Ravi and Taashi worked together on this project, pair programming, and took the work of modifying their Lucene java implementation to work on these queries as well as Task 2, the query expansion, choosing to use BM25 for the retrieval model. Ravi and Taashi also worked together on Phase 2, the snippet generation algorithm, while Alex implemented the evaluation metrics.

There are a few independent pieces to the project, each of which can be compiled and run individually of the others. This includes the indexer, retriever, and evaluation directories, each of which has their own README.md file explaining how to compile/run the code. The top level README.md file explains the rest of the files in the project.

Literature and resources –

BM25 Algorithm:
It is a popular retrieval model which extends the binary independence model to include query and term weights. The values of parameters k1 = 1.2, b= 0.75 and k2= 100 are set for maximum effectiveness.

TF-IDF Algorithm:
Also known as term frequency inverse document frequency id based on term and query frequency weights.
tf = no. of times a term occurs in a document/ document length
idf = 1 + log (Total no. of documents) / (number of documents in which term appears) + 1

Pseudo Relevance feedback: automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction. The method is to do normal retrieval to find an initial set of most relevant documents, to then *assume* that the top k ranked documents are relevant, and finally to do relevance feedback as before under this assumption.

Snippet Generation:
Helps in creating snippets or summaries of the documents. They show a part which is relevant to the document. Luhn's algorithm for calculating the significance factor – Significant factor = $\frac{\text{number of significant words}^2}{\text{Bracket size}}$

Lucene: We have used the version which was given in Assignment 4. It uses the following JAR files –

1. Lucene-core-VERSION.jar

2. Lucene-queryparser-VERSION.jar
3. Lucene-analyzers-common-VERSION.jar

JSOUP: We have used JSOUP in Lucene for scraping and parsing the HTML.

Implementation and Discussion -

Query Expansion: has been performed using Pseudo Relevance Feedback.

1) Normal retrieval is performed with initial query.
2) Top k documents (k – 10) from the results are included in the relevant set of documents.
3) Rocchio Algorithm is used to generate the new query.
   a) Initial query vector with tf scores which are aligned with inverted index terms are generated.
   b) Relevant set of document vector is generated.
   c) Non-relevant set of document vector is generated.
   d) Then modified query is generated using the formula:
   e) The algorithm proposes using the modified query qm:
      $qm = \alpha q0 + \beta / |Dr| \Sigma dj - \gamma / |Dnr| \Sigma dj$

      where q0 is original query vector, Dr and Dnr are the set of known relevant and non-relevant documents respectively and α, β and γ are weights attached to each term.
      A = 1.0, β=0.75 and γ=0.15
4) The top 20 terms with the highest weight and which are not present in the query are appended to the original query.
5) Normal retrieval is performed with the new query and the search results are generated.

Query Analysis

Here we will compare the results of a few different queries between the different retrieval models.

Query 1: Time Sharing System on OS for IBM computers

tfidf	tfidf*	tfidf^	Bm25	Bm25*	Bm25^	Ql	Ql*	Ql^	Lucene
1938	3127	1938	2319	3127	2629	2319	3127	2319	2319
1657	1461	1657	1657	2246	2319	**1410**	2246	2629	1657
2371	3068	2371	2629	1930	1657	1657	3196	**1410**	**1410**

* Indicates the model run on the stemmed corpus
^ Indicates the model run using the stop list
Bold Indicates a document which was on the relevant list provided

The highest ranked documents for this query included:
1938: Some Criteria for Time-Sharing System Performance
3127: Thoth, a Portable Real-Time Operating System
2319: Operating System Performance

And the second ranking documents included:
1657: Implementation of the SHARER2 Time-Sharing System
1461: Discussion Summary on Operating Systems
2246: Levels of Language for Portable Software
2319: Operating System Performance
1410: Interarrival Statistics for Time Sharing Systems
2629: The UNIX Time-Sharing system

For our first query only 30% of the retrieval models return a relevant document in the top three ranked documents, though they do all behave similarly with only three different options being returned for the top document. It seems that all our retrieval models fail to extract the true key terms in the query, “Time Sharing System”, instead focusing more on terms like “Operating Systems”, a term which is common across nearly all the documents returned.

Query 19: Parallel Algorithms

tfidf	tfidf^	Bm25	Bm25^	Ql	Ql^	Lucene
141	**141**	**2973**	**2973**	**2973**	**2973**	**2973**
2973	**2973**	**3075**	**3075**	**3075**	**3075**	**3075**
371	371	**2266**	**2266**	**2266**	**2266**	2557

* Indicates the model run on the stemmed corpus
^ Indicates the model run using the stop list
Bold Indicates a document which was on the relevant list provided

The highest ranked documents for this query included:
141: Some Thoughts on Parallel Processing
2973: Properties of the Working-Set Model

And the second ranking documents included:
2973: Sorting on a Mesh-Connected Parallel Computer
3075: Fast Parallel Sorting Algorithms

I picked this query for analysis because I thought the consistency of it was interesting. Almost all of our models returned the same top document for this query, 2973. Tfidf was the only model to have a different document in rank one, but 2973 was just below at rank 2. Combining the top three for all models 18/21 were relevant with tfidf and Lucene throwing in non-relevant documents in rank three.

Results-

MAP:

tfidf	tfidf^	bm25	bm25^	ql	ql^	lucene	bm25 w/ expansion
0.1381	0.1386	0.1604	0.1707	0.1530	0.1694	0.1594	0.2133

MRR:

tfidf	tfidf^	bm25	bm25^	ql	ql^	lucene	bm25 w/ expansion
0.5343	0.5614	0.7091	0.7214	0.6820	0.7344	0.7148	0.8250

^ Indicates the model run using the stop list

These two metrics give us a few good insights into the models as a whole. First, we can see that stopping has a globally positive effect on results. While the amount of increase does vary, stopping increases both MAP and MRR scores for all three retrieval models where stemming was run. Query Likelihood seems to have been given the highest boost from stemming, increasing the MAP score by 0.0164 and MRR score by 0.0524 compared to tfidf which only had a bump of 0.0005 and 0.027 in MAP and MRR respectively. Query expansion as a whole had a positive impact on results. The BM25 implementation with pseudo relevance feedback performed best out of our 8 runs overall when comparing MAP and MRR scores.

Lucene performed best when looking at MRR scores, indicating that it is best out of the four of giving a quick relevant document near the top of the list. BM25, however has the best score when comparing the MAP scores for the baseline runs indicating that it returns more relevant documents overall than Lucene or any of the other baseline models.

Unfortunately, we only have a query expansion run with BM25 so it is difficult to judge how much the query expansion aided retrieval. Future work may include running expansion on the other three baseline implementations to judge if query expansion is more valuable when using any one retrieval model in particular.

P@K, Precision, Recall:
These results can be seen in the evaluation/results directory.

Query 5:

P@K	tfidf	tfidf^	bm25	bm25^	ql	ql^	lucene	bm25 w/ expansion
K=5	0.2	0	0.2	0.4	0.2	0.4	0.2	0.2
K=20	0.15	0.2	0.1	0.1	0.1	0.1	0.15	0.2

Comparing P@K with K=5 vs 20 gives a good indication that including more documents does not seem to help any of our retrieval models achieve a higher precision. The models with the best performance were able to retrieve 2 of the 8 relevant documents in the top 5 documents retrieved, however for both the models able to do so P@K = 20 was reduced to 0.1, indicating that no relevant documents were returned in rank 6 through 20. Only one model performed better with the higher K, tfidf with stemming, which did not return any relevant documents in the top 5, however when looking at the top 20 performed better or equal to all other models.

Query 22:

P@K	tfidf	tfidf^	bm25	bm25^	ql	ql^	lucene	bm25 w/ expansion
K=5	1	1	0.8	1	0.6	0.8	0.6	0
K=20	0.55	0.6	0.65	0.65	0.5	0.65	0.45	0.15

Interestingly enough it is the simplest retrieval model in our analysis which performs the best in this query. All of the top 5 documents retrieved with either tfidf implementation were relevant, while none of the top results from the bm25 with query expansion were. The results for this query are completely opposite from the general figures we see with MAP and MRR where query expansion clearly outperforms tfidf in the general case, however with this query in particular tfidf wins out. My guess is that query expansion includes terms which are not relevant to the initial query, throwing off the results completely. It is an interesting case which shows that the best model in general may perform quite poorly when given some specific tasks.

Conclusion:

As expected, the more complex models evaluated in this project performed better than the simple ones in the general case. MAP and MRR scores give us a good indication that results using techniques such as query expansion and stopping are, in general, better than the results without them. However, comparing P@K scores for specific queries across all our models does show that this is not always the case. Query expansion seems to perform poorly when given long queries, whereas tfidf performance was not greatly impacted. As is expected, different models perform better than others when given certain kinds of tasks and if we were building out a real search engine with these tools this would need to be something we take into account. In general, our implementation of BM25 with query expansion seems to work best, achieving relatively good scores for MAP, MRR, and P@K=5 across most of the queries.

Outlook:

We can improve our model by incorporating the following schemes –

1) Use Pagerank to calculate the popularity of a page.
2) Use compression techniques like Elias encoding.
3) Using a more complex index model such as a bigram term or by storing position information rather than raw term counts.
4) Perform and evaluate query expansion on all baseline retrieval models.

Bibliography:

Croft, W.Bruce; Metzler, Donald; Strohman, Trevor *Search Engines: Information Retrieval in Practice.* Pearson Education 2015

Lucene Version 4.7.2 documentation http://lucene.apache.org/core/4_7_2/

https://nlp.stanford.edu/IR-book/html/htmledition/pseudo-relevance-feedback-1.html