Jean Nei, Manpreet Kaur, & Mingming Li
Professor Nada Naji
Information Retrieval
Fall 2018

Introduction:

For this project, my teammates and I designed and developed 4 different retrieval models using both Java and Python. The retrieval models designed and developed by us were BM25, JM Smoothed Query Likelihood, TF-IDF and the final one using Lucene's Standard Analyzer. Each of our models used our own unigram indexer, while Lucene indexed the documents using their proprietary algorithms.

Jean implemented the BM25 retrieval model [1][2], the indexers, proximity and exact match retrieval systems, he also handled the implementation for Lucene. This includes phase 1, part 1 and part 3, including the write up. In addition to this Jean handled the extra credit. These parts were implanted using Java. Manpreet implemented the JM Smoothed Query Likelihood [1], query refinement, and snippet generation, using python. This includes phase 1 part 1, part 2, and phase 2. Mingming implemented the TF-IDF retrieval model [1] and generated all the tables and figures for the project, using python. This includes phase 1 part 1, part 3 for TF-IDF only, and phase 3.

Literature and Resources (Query Refinement and Snippet Generation):

The query refinement technique used for this project was **Pseudo Relevance Feedback** to expand the query on the JM Smoothed Query Likelihood retrieval model. The algorithm for refinement was inspired from the textbooks *Information Retrieval in Practice* by ***Croft Et al.*** [1] and *Introduction to Information Retrieval* by **Manning et al**.[2] We selected the top 10 documents from the ranked list. 10 was chosen because users in real life will typically see a limited subset of results, and since the document length in this corpus was pretty small with the average document length being ~74 words we selected the top 16% of terms present from these documents, which came out to be ~11.84, which resulted in us picking the top 12 terms from these documents to use for query expansion.

Query-dependent text summarization technique is used for snippet generation and was inspired from the textbook *Information Retrieval in Practice* by ***Croft Et al.*** [1]. By using Luhn's significant factor algorithm we were able to select the significant words based on the incoming query to make our snippet generation approach dynamic. The algorithm works by identifying text windows (sentences) and maximizing the amount of significant terms present from the query. Once we had the scores for each window we simply selected the window with the highest score and used that as preview sentence.

Implementation and Discussion:

For the BM25 retrieval model, there are several parameters that can be used to tune how effective the model retrieves documents. The values **k1** and **k2** correspond to term frequency saturation for both the document and query respectively, and the **b** value is used for document length normalization. The value selected for k1 was **1.2**, k2 was **500**, and b value of **0.75**. 1.2 and 0.75 were chosen based off literature review, specifically from ***Croft Et al.*** *Information Retrieval in Practice*. [1] Since the queries we were analyzing contained duplicate words k2 was selected to be 500, to factor this feature in, k2 is usually between 0 – 1000. Once the parameters were tuned,

retrieval was done by using document at a time processing with a slight modification, we first obtained all documents that contain the query words, this works by simplify selecting those terms from the index, then we created another data structure that was mapped by the document id (document name in this case), and the values were a list of term objects that contained the frequency of that given query term in the document. This implementation was chosen because it allowed for the most flexibility when designing the proximity and exact match algorithms for the extra credit, more on that later on. Also by focusing on the documents that contains at least one of the query terms, we significantly reduce the total number of documents we need to process. This model was also used to handle the stop words, instead of rebuilding our index to remove stop words, we simply applied query time stopping and ignored any stop word that was present in the query.

For the TF-IDF retrieval model, we used term at a time retrieval to build partial scores for each document based on each query term and then multiplied this value by query term frequency, to provide higher weights for terms that appear multiple times in the query. This was done to emphasize and score terms higher if they frequently appeared in the query. This model was used for the stemming part of the project and for the stopping part of the project once again we used retrieval time stopping.

For the JM Smoothed Query Likelihood model, we used term at a time retrieval to build partial scores for each document. By doing this it allowed us to utilize the smoothing ability of JM by providing non-zero weights to terms that did not appear in the document. This model was also used for the query enrichment process, which was detailed earlier, but as a summary we selected the top 10 documents from the retrieval results and selected the top 16% of terms from those documents to expand the query with.

For the extra credit part of this project, we selected the BM25 retrieval model. This was done because it allowed us to get **Best Match** for free, which will return a result if at least one the query terms appear in the document. In order to support both **Best Match with Proximity** and **Exact Matching**, we extended this model to utilize the index with positions. The exact match algorithm works by first selecting all documents that contains each of these query terms, while eliminating the documents that do not contain each query term. Then we utilize the position information to check if the terms appear in the appropriate windows (proximity of 0), if it does we include it, else we ignore it. To support best match with proximity we made a minor adjustment to the algorithm for exact match, instead of checking if each query term appears in the document, we include the document if a query term appears at least once.

The first query we are going to analyze is query number 33, which was one of the longest queries available from the sample query set with a length of 73. The model that retrieved a relevant document the earliest across all models was TF-IDF w/ stop words, it retrieved a relevant document at rank 2. The next best retrieval model was Lucene w/ stop words, it retrieved a relevant document at rank 11. The next best retrieval model was the standard Lucene implementation which retrieved the first relevant document at rank 21. The next best retrieval model for this query was BM25 w/ stop words which retrieved its first relevant document at rank 28. The JM smoothing model performed the worst overall with its first ranking coming at 46, for

the baseline, and rank 32, using query refinement. As you see the retrieval systems perform better when they exclude common words from the query.

The next query we are going to analyze is query number 19, which is the shortest query with a length of 2 words. Both BM25 models (baseline and with stop words), both Lucene models (baseline and with stop words), and TF-IDF with stop words, all retrieved a relevant document at rank 1. JM smoothed with query refinement and TF-IDF baseline retrieved its first relevant document at rank 3, and JM smoothed baseline retrieved a relevant document at rank 2.

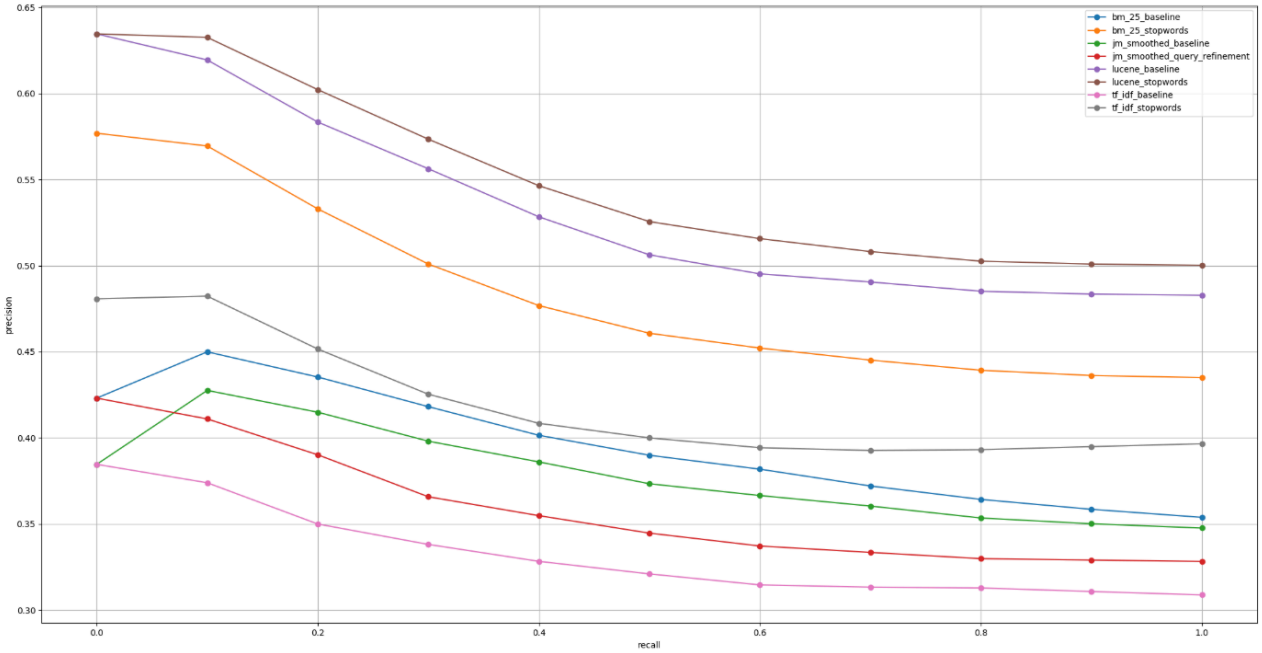
Results:

Table 1:

Retrieval System	MAP	MRR	p@5(AVG)	p@20(AVG)
bm_25_baseline	0.333285621	0.587662888	0.311538462	0.171153846
bm_25_stopwords	0.395302918	0.688516308	0.357692308	0.188461538
jm_smoothed_baseline	0.340611555	0.58277808	0.342307692	0.176923077
jm_smoothed_query_refinement	0.27852467	0.536055712	0.226923077	0.132692308
lucene_baseline	0.447455523	0.75746337	0.4	0.225
lucene_stopwords	0.462302459	0.747952705	0.4	0.235576923
tf_idf_baseline	0.269376198	0.508294137	0.219230769	0.134615385
tf_idf_stopwords	0.350926557	0.615758547	0.284615385	0.175961538

This table represents the average precision, mean reciprocal rank, and precision at ranks 5 / 20 for the 8 runs. A trend you can see by looking at this table is that with both BM25 and TF-IDF retrieval performed better when we ignored common words. Lucene had a minor improvement when common words were excluded. JM smoothed baseline performed better than the baselines for BM25 and TF-IDF, but performed worst when those models excluded common words. The Lucene analyzer outperformed all of the retrieval models. This observation is not surprising to us because Lucene by default has a list of common words that it uses when indexing / querying, and from the table above you see an increase in effectiveness when common words are removed. In order to see a more detailed breakdown of the precision and recall values please look in the precision recall directories for all the tables.

Figure 1:



With this figure, it's obvious that Lucene was the best retrieval model, purple and brown lines, which does not surprise us. By default, Lucene uses a list of common words to exclude which plays a factor in effectiveness. Not only that but the models that did incorporate stop words had their effectiveness improved overall.

Conclusions and outlook:

In conclusion, the Lucene retrieval model was better than our custom implementations. By default, Lucene uses a default list of common words for stopping which we believe contributed to its overall better effectiveness. As you can see from the results figure 1, when common words were used for stopping the retrieval models performed better. The next best retrieval model was the BM25 model with stopping. We did expect the JM Smoothed Query Likelihood model to perform better than it did. We expected this because it also allows for parameter tuning with the lambda score. A contributing factor to why it did not perform as well could be due to the length of the queries and the fact that we did not remove common words when performing the retrieval. In general, the queries used for the assignment were long, and by using common words, the query became shorter which backs up the known fact the shorter queries perform better than longer queries.

If we had the opportunity to modify this project, we would want to use JM Smoothed Query Likelihood with stop words. This model's baseline outperformed TF-IDF baseline but once stop words was introduced for TF-IDF, TF-IDF outperformed the JM baseline. What we would also like to do is experiment with the parameters for JM and BM25 to see if the effectiveness improves or decreases. This project gave us a good baseline to test our future modifications with to further improve the effectiveness.

Bibliography:

1. Croft, W. Bruce., et al. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010.
2. Manning, Christopher D., et al. *Introduction to Information Retrieval*. Cambridge University Press, 2009.