INFORMATION RETRIEVAL
Fall 2018

PROJECT REPORT

By:

Darshit Chanpura
Dharmish Shah
Soumyadeep Sinha

Instructor:
Prof. Nada Naji

Introduction

There are 3 phases of the project and each phase has its own tasks.

Phase-1 has 3 tasks. Task-1 consists of 4 retrieval systems namely BM_25, TF_IDF, JM_Query_Likelihood (lambda value is 0.35) and Lucene's default retrieval system. In Task-2 we did *query enrichment* using 'pseudo-relevance feedback' for query expansion while considering BM_25 system as the base. In Task-3, we generated new indexes: one with no stop words and other with stemming. These we generated using the 4 retrieval systems.

Phase-2 we created a Snippet generator which generates summary of the output page as ranked and the query term highlighted in the summary.

Phase-3 we assessed the performance of the system by evaluating results from 8 distinct runs (4 from task-1, 1 from task-2, 3 from task-3). We then calculated MAP, MRR, P@K with K=5 & 20, Precision & Recall. Lastly, we plot the Recall-Precision curve for all the 8 plots in one figure.

Dharmish Shah implemented Task-1 and Phase-2 of the project and contributed about the same in the report. Darshit Chanpura implemented Task-2 and Phase-3 of the project and contributed about the same in the report. Soumyadeep Sinha implemented as Task-3 and extra-credit of the project and contributed about the same in the report.

Literature and resources

The course textbook and the slides were the primary source of reference. For programming related questions, we used the Python documentation and BeautifulSoup documentation.

Implementation and Discussion

Phase 1: Indexing and Retrieval

Task 1:

1. To start with, we implemented the 4 Retrieval Systems, 3 of which (BM25, JMSmoothing, TF.IDF) were coded from scratch and 4th was Lucene's default model.
2. We created a cleaned corpus using the Generator.py and then we use this corpus for all the 4 systems.

3. BM_25, the first system: the assumptions – R = 0, r_i = 0, b = 0.75, k1 = 1.2, k2 = 100.
4. We start by refining the query (stopping and stemming) and split it into terms.
5. We calculate K as shown in the formula below.
6. We calculated the partial score using the basic ranking function.
7. While looping over each query term, we find its idf component; and calculate the tf component for each document, using the formula given below.

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$$

$$K = k_1((1-b) + b \cdot \frac{dl}{avdl})$$

where the summation is now over all terms in the query; and N is the set of non-relevant documents, R is the set of relevant documents, n_i is the number of non-relevant documents in which the term *i* is found, and r_i is the number of relevant documents in which the term *i* is found*. r and R are set to zero if there is no relevance information*; f_i is the frequency of query term *i* in the document; qf_i is the frequency of term *i* in the query; and k1, k2, and K. b is a parameter, *dl* is the length of the document, and *avdl* is the average length of a document in the collection. The constant b regulates the impact of the length normalization are parameters whose values are set empirically.

1. Jelinek Mercer Likelihood model, the second retrieval system. We used JM as our smoothing function. We were given λ = 0.35.
2. Since it is a language model it'd score all relevant documents even if query term doesn't exist int the document.
3. We start by refining the query and split it into terms.
4. We then get the list of docs in which each query term appears.

5. We calculate term count per document for all documents, and the total words in the collection.
6. We then calculate score for each document using the following formula:

$$\log P(Q|D) = \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

7. We sort the score in descending order, with highest relevant document ranked first.

8. TF_IDF, the third system:
9. We start by refining the query and split it into terms.
10. We calculate the *tf* and the *idf* component using the following formula:

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

where tf_{ik} is the term frequency weight of term *k* in document D_i, and f_{ik} is the number of occurrences of term *k* in the document.

$$idf_k = \log \frac{N}{n_k}$$

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

11. We then multiplied both the values to obtain the final score.
12. We used the same corpus as the other two.

13. For Lucene, the last retrieval system:
14. We used the same corpus as the last three systems.
15. We used Lucene's default ranking and indexing model.

For all these retrieval systems, we followed term-at-a-time strategy and used accumulator table to store the partial scores.

Output: (no stemming and no stopping)

1. BM25: src/results/bm_25

2. JMSmoothing: src/results/jm_query_likelihood
3. TF.IDF: src/results/tf_idf
4. Lucene: src/results/lucene

Task 2:

1. For the query enrichment task, we used Pseudo-relevance feedback technique for query expansion and BM25 for retrieval. We retrieved top 1,2,5,10 documents for first run and passed it into second run which gave us the more accurately ranked documents.

Output:

1. Pseudo relevance feedback: src/results/prf
2. For better understanding we've shown results here:

No. of documents (top_k)	MAP	MRR
1	0.13517187499999997	0.607515762292209
2	0.103991875	0.5024509803921569
5	0.08956125000000001	0.38080303218775546
10	0.0803259375	0.35722495741248966

Hence, we finally set top_k to 1 which retrieves the most relevant result out of these. The reason behind choosing BM25 was it's *tf* component because we are using term frequency count for this project.

Task 3:

Part -A:

1. We created a list of Stop words from the “common_words.txt”, cleaned the corpus and then generated index of the same.
2. Then we performed and retrieved the results for the queries in “cacm.query.txt” for 3 baseline runs on this new index, namely: BM25, tf.idf and JMSmoothing.

Part-B:

1. We first created stemmed corpus of all the 3204 files using “cacm_stem.txt” and indexed them.
2. We performed retrieval on this index by using the stemmed queries given in cacm_stem.query.txt

Output:

The output for the baseline runs with 2 variants can be found in:

1. BM25 with stemming: src/results/bm_25_stemmed
2. BM25 with stopping: src/results/bm_25_stopped
3. JMSmoothing with stemming: src/results/jm_query_likelihood _stemmed
4. JMSmoothing with stopping: src/results/jm_query_likelihood _stopped
5. TF.IDF with stemming: src/results/tf_idf_stemmed
6. TF.IDF with stopping: src/results/tf_idf_stopped

Phase 2: Displaying Results

1. We started by reading the file from the clean corpus based on the ranked document names in query_id.txt.
2. We did this for each retrieval model and over each query.
3. We then searched for that query term in the read file in a window of 5 and then generated a snippet with that window. (e.g window of 5would give: ….got out on golden duck….)
4. We then calculated the number of query terms in that window.
5. More the number of query terms, higher the rank of the sentence.
6. We then extracted top 3 sentences.
7. Then we highlighted the term by making the term bold in the snippet text.

Output:

1. BM25: src/results/bm_25/
2. BM25 stemmed: src/results/bm_25_stemmed/
3. BM25 stopped: src/results/bm_25_stopped/
4. JMSmoothing: src/results/jm_query_likelihood
5. JMSmoothing stemmed: src/results/jm_query_likelihood_stemmed
6. JMSmoothing stopped: src/results/ jm_query_likelihood_stopped
7. TF.IDF: src/results/bm_25
8. TF.IDF stemmed: src/results/bm_25_stemmed
9. TF.IDF stopped: src/results/bm_25_stopped

Phase 3: Evaluation

We performed following 8 runs to assess the performance of our retrieval systems:

1. BM25, JM Smoothing, TF.IDF, Lucene (4 runs)
2. Query refinement run using Pseudo relevance feedback (1 run)
3. BM25, JM Smoothing, TF.IDF with stopped index

The evaluation criteria:

1. MAP

$$Mean\ \text{Average Precision (MAP)} = \frac{Sum\ of\ Average\ Precision\ for\ each\ query}{Total\ number\ of\ queries\ executed}$$

2. MRR

$$Mean\ Reciprocal\ Rank\ \text{(MRR)} = \frac{Sum\ of\ Reciprocal\ Rank\ for\ each\ query}{Total\ number\ of\ queries\ executed}$$

3. P@K

$$Precision\ @\ K = Precision\ for\ Kth\ rank\ document\ in\ the\ result\ of\ a\ query$$

4. Precision & Recall

$$Precision = \frac{Number\ of\ Relevant\ documnets\ Retrieved}{Total\ Documents\ Retrieved}$$

$$\text{Recall} = \frac{Number\ of\ Relevant\ documnets\ Retrieved}{Total\ Relevant\ documents}$$

5. Average Precision

$$\text{Average Precision} = \frac{Sum\ of\ Precision\ at\ Relevant\ documents}{Total\ Relevant\ documents}$$

6. Reciprocal Rank

$$\text{Reciprocal Rank } = \frac{1}{Rank\ of\ first\ Relevant\ document}$$

7. Average Recall

$$\text{Average Recall} = \frac{Sum\ of\ Recall\ at\ Relevant\ documents}{Total\ Relevant\ documents}$$

1. We started by retrieving all the relevant document names for each query id given in 'cacm-rel.txt'.
2. We then calculate precision and recall values for each query. Finally, we calculate MAP and MRR. We then repeat this step for each run.
3. For curve plot, we are storing the average precision and average recall for each query in a list; and we plot these lists as y and x points in the graph. We plot these for all the 8 runs, and finally we show the graph.

For each of these runs we have stored the outputs of MAP, MRR, P@K, Precision & Recall in the respective output files inside each run's folder. For Precision and Recall we've store it as a query_id.txt. For MAP and MRR we've stored it in 'map_mrr.txt'. P@K stored as 'p_at_k.txt'.

Output: "src/evaluation"

Query by query analysis

Query 1:
"Addressing schemes for resources in networks; resource addressing in network operating systems"

Top relevant documents for above query:
CACM-2625
CACM-2849
CACM-3032

Retrieval Models output:

BM 25	JM Smoothing	TF-IDF
#1 CACM-2625	#2 CACM-2625	#1 CACM-2625
#6 CACM-3032	#3 CACM-3032	#13 CACM-3032
#73 CACM-2849	#75 CACM-2849	

All relevant documents of this query are present in BM 25 and JM Smoothing. 1 document is not present in TF – IDF as it only considers term frequency and does not consider any other metric. Let us compare for 2 more queries.

Query 2:
"Graph theoretic algorithms applicable to sparse matrices"

Top relevant documents for above query:
CACM-1563
CACM-2695
CACM-2986

Retrieval Models output:

BM 25	JM Smoothing	TF-IDF
#1 CACM-2695	#1 CACM-2695	#3 CACM-2695
#3 CACM-2986	#3 CACM-2986	#5 CACM-2986
#89 CACM-1563	#85 CACM-1563	

Query 3:
"Any information on packet radio networks. Of particular interest are algorithms for packet routing, and for dealing with changes in network topography. I am not interested in the hardware used in the network."

Top relevant documents for above query:
CACM-2578
CACM-2849
CACM-2890
CACM-2949
CACM-3032

Retrieval Models output:

BM 25	JM Smoothing	TF-IDF
#1 CACM-3032	#1 CACM-2949	#1 CACM-2890
#2 CACM-2949	#2 CACM-3032	#2 CACM-3032
#4 CACM-2849	#4 CACM-2890	#3 CACM-2949
#5 CACM-2890	#13 CACM-2849	#4 CACM-2849

Stemming analysis for 3 Queries

Query 1: “portabl oper system”:

The Stemmed corpus and stemmed query were given to us and we determined the rankings based on BM25, JM Smoothing and TF-IDF. Once the ranking was generated we found the top 4 documents to be :

BM 25	JM Smoothing	TF-IDF
#1 CACM-3127	#1 CACM-3127	#1 CACM-3127
#2 CACM-3196	#2 CACM-3196	#2 CACM-1930
#3 CACM-1930	#3 CACM-1461	#3 CACM-2319
#4 CACM-2246	#4 CACM-1930	#4 CACM-1680

As we can see form the above data, the ranking given by the BM25 and JM Smoothing is almost same.The top ranking given by all three models are exactly the same.

Query 2: “parallel processor in inform retriev”:
The Stemmed corpus and stemmed query were given to us and we determined the rankings based on BM25, JM Smoothing and TF-IDF. Once the ranking was generated we found the top 4 documents to be

BM 25	JM Smoothing	TF-IDF
#1 CACM-0001	#1 CACM-0001	#1 CACM-0001
#2 CACM-0002	#2 CACM-0002	#2 CACM-0002
#3 CACM-0003	#3 CACM-0003	#3 CACM-0003
#4 CACM-0004	#4 CACM-0004	#4 CACM-0004

As we can see all three models give us the same ranking.

Query 3: “parallel algorithm”:
The Stemmed corpus and stemmed query were given to us and we determined the rankings based on BM25, JM Smoothing and TF-IDF. Once the ranking was generated we found the top 4 documents to be

BM 25	JM Smoothing	TF-IDF
#1 CACM-2714	#1 CACM-2266	#1 CACM-2714

#2 CACM-0950	#2 CACM-2714	#2 CACM-1811
#3 CACM-2785	#3 CACM-2973	#3 CACM-2342
#4 CACM-2973	#4 CACM-3075	#4 CACM-0950

As we can see from the above data, the top document of the BM25 and TFIDF are exactly same. However, for JM smoothing the same document is ranked 2 which is not that drastic of a change. We can also see that the few of the documents that showed up in BM25 is also there in JMSmoothing as well as TFIDF.

Advance Searching (Extra Credit)

The Advance searching allows user to search using various criteria such as exact phrase match, best match or ordered best match with a certain proximity.

Exact match: In the exact match the exact query needs to match with a document for it to be retrieved. If the query doesn't match with any of the documents then none of the documents will be retrieved. The program/ code has been written in such a way that if there are multiple documents having the exact same query then all the documents having the query will be retrieved and then using BM25 we can rank the documents retrieved to get the most relevent document. The User can give their own query in the console or if the user wants to run the predefined query from the query list given, he/she can do that as well. An Option has been made available to the user to choose if they want to remove the stop words from the query or not.
Once, the query is preprocessed, the program takes the query and find the all the documents that contains the query terms which is derived by splitting the query. Once all the documents for all the terms are retrieved. We find the intersection of the document which would only contain documents which contains all the query words and in the order specified by the user while constructing the query. Once the documents which contains the query exactly as it is, the list of document is sent to BM25 for the documents to be ranked so that the relevent documents are at the top.

Now, because of the nature of the exact match, which tries to match the exact phrase it is very difficult to get a hit.
For example:
for the query "Parallel Algorithms": there was a single hit in the document "CACM-2433" as evident by the line in the document:

"Language structures to utilize this storage method and express **parallel algorithms** are described."

Thus , the exact match works well.

Best match: In the best match, its not required for all the query words to be present. If any on e of the query terms is present in the document, we consider it as a relevent document. Now, once all the documents which contains any of the query terms are retrieved, we have no way of finding out which documents are relatively more relevent so we use BM25 to rank the documents that were retrieved and score them to find the most relevent document. Here, in this case BM25 works well with the Best Match algorithm.
For example: for the query "code optimization for space efficiency"
The top ranked documents retrieved are :

BM 25
#1 CACM-1947
#2 CACM-2491
#3 CACM-3054
#4 CACM-1886
#5 CACM-3080

If we check the document "CACM-1947" we find that the document is based on "Object code Optimization". "Code" and "optimization" are both in the document. Apart from that even "efficiency" is present in the document. Hence, it is ranked at the top.
Similarly, in document "CACM-2491" , we find that "code" and "efficiency" are present in the document hence, this document also shows up in the list.

Ordered Best Match Proximity: In ordered best match proximity, first we define a proximity window. In our case, we ran it for the proximity window of 5. In Ordered best match, the documents which contains the query terms are retrieved. However, only those documents are considered where any two query terms are separated by no more than N, here 5, other tokens. Once, the relevent documents are retrieved, the list of documents are scored by BM25. This is to get the most relevent documents of the list of documents that were retrieved.

For example for the query: "Parallel Algorithms"
These were the results:

BM 25
#1 CACM-2714
#2 CACM-1262
#3 CACM-2700
#4 CACM-2685
#5 CACM-1158

If we check the document "CACM-1158" we will find the it contains:

“These performances compare very favorably withthe previous best **parallel** merging **algorithm**,”.

As we can see that both the query terms are present and in order and within th ewindow of 5 words. So this document is highly relevent. Hence, the ranking of this document as shown by the above data is justified.

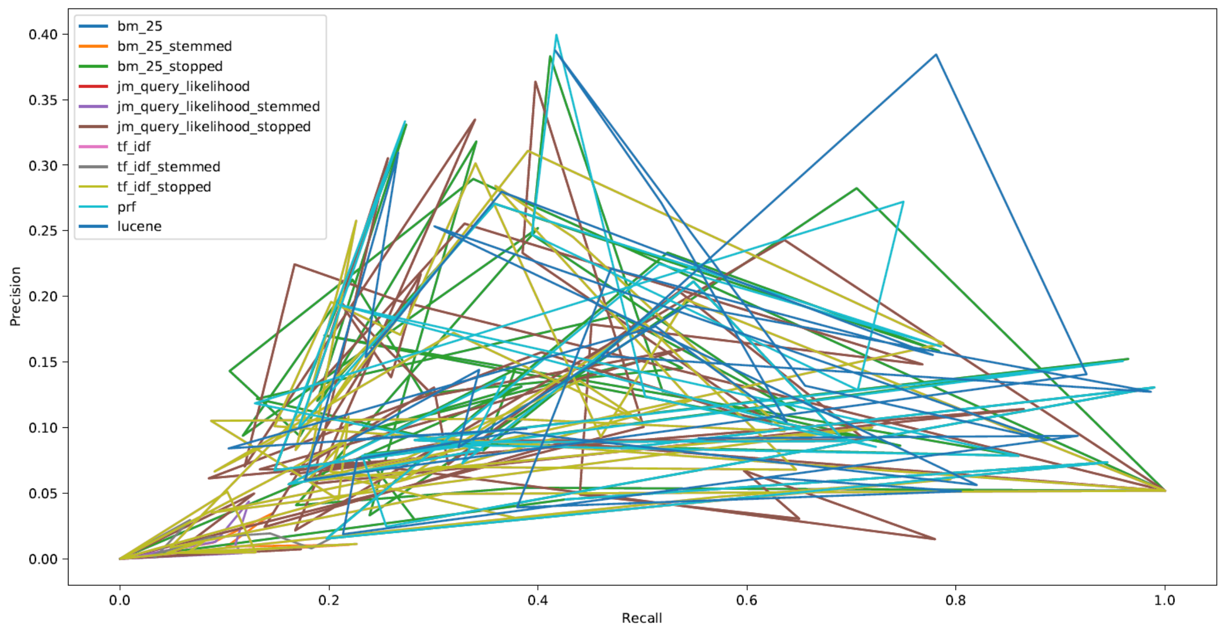
Hence, the ranking generated by our code is pretty reliable for all the three advance searches.

<u>**Results**</u>

Retrieval Model	Mean Average Precision	Mean Reciprocal Rank
BM 25 (no stemming and no stopping)	0.13363730769230767	0.6227060866483943
BM 25 with stemming	0.010188749999999998	0.027007697286952605
BM 25 with stopping	0.1324013461538461	0.607941283756501
JM Smoothing (no stemming and no stopping)	0.11013634615384611	0.49064407370166163
JM Smoothing with stemming	0.009879999999999995	0.054096320346320344
JM Smoothing with stopping	0.11013634615384611	0.49064407370166163
TF IDF (no stemming and no stopping)	0.0975444230769231	0.43306235088452255
TF IDF with stemming	0.011709999999999998	0.02821739564396196
TF IDF with stopping	0.0975444230769231	0.43306235088452255
BM 25 after Pseudo Relevance Feedback	0.13551499999999997	0.6080827172939067
Lucene	0.1463828125	0.6364752435064935

Table 1: Mean Average Precision and Mean Reciprocal Rank

Precision-Recall Plot



Conclusion

Observing the results, we can say that Lucene is the best retrieval model in this project. It has higher MAP indicating that high number of relevant documents are retrieved and higher MRR indicates these documents are retrieved from top k documents itself.

Outlook

Smoothed query likelihood model would improve performance in our case, along with the stop words removed and query expanded properly. Also, term positions would result in better matches rather than term frequency as it also matches proximity. Smoothed model has higher probability of returning some documents that are not even relevant but still might improve chances of getting relevant documents, in case of smaller corpus.

Bibliography:

1. Croft W. B., Metzler D., & Strohmann T. (2010). Search engines. Pearson Education.
2. Beautiful Soup (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)
3. Official Python Documentation (https://docs.python.org/3/)