**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**

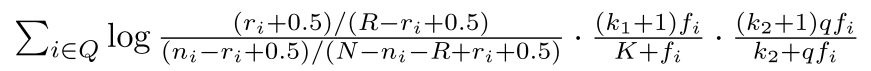
The course textbook and the slides were the primary source of reference. For programming related questions, we used the Python documentation and BeautifulSoup documentation.

**Implementation and Discussion**

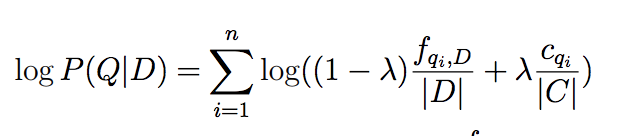
**Phase 1: Indexing and Retrieval**

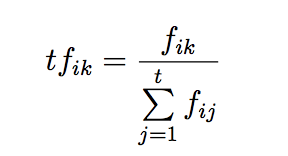
Task 1:

* 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.
* We created a cleaned corpus using the Generator.py and then we use this corpus for all the 4 systems.
* BM\_25, the first system: the assumptions – R = 0, ri = 0, b = 0.75, k1 = 1.2, k2 = 100.
* We start by refining the query (stopping and stemming) and split it into terms.
* We calculate K as shown in the formula below.
* We calculated the partial score using the basic ranking function.
* While looping over each query term, we find its idf component; and calculate the tf component for each document, using the formula given below.

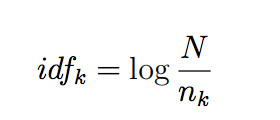


  
where the summation is now over all terms in the query; and N is the set of non-relevant documents, R is the set of relevant documents, ni is the number of non-relevant documents in which the term *i* is found, and ri is the number of relevant documents in which the term *i* is found*. r and R are set to zero if there is no relevance information*; fi is the frequency of query term *i* in the document; qfi 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.

* Jelinek Mercer Likelihood model, the second retrieval system. We used JM as our smoothing function. We were given λ = 0.35.
* Since it is a language model it’d score all relevant documents even if query term doesn’t exist int the document.
* We start by refining the query and split it into terms.
* We then get the list of docs in which each query term appears.
* We calculate term count per document for all documents, and the total words in the collection.
* We then calculate score for each document using the following formula: 
* We sort the score in descending order, with highest relevant document ranked first.
* TF\_IDF, the third system:
* We start by refining the query and split it into terms.
* We calculate the *tf* and the *idf* component using the following formula:



where *tfik* is the term frequency weight of term *k* in document *Di*, and *fik* is the number of occurrences of term *k* in the document.



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.

* We then multiplied both the values to obtain the final score.
* We used the same corpus as the other two.
* For Lucene, the last retrieval system:
* We used the same corpus as the last three systems.
* 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:

* 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

* 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:

* We created a list of Stop words from the “common\_words.txt”, cleaned the corpus and then generated index of the same.
* 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:

* We first created stemmed corpus of all the 3204 files using “cacm\_stem.txt” and indexed them.
* 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**

* We started by reading the file from the clean corpus based on the ranked document names in query\_id.txt.
* We did this for each retrieval model and over each query.
* 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….)
* We then calculated the number of query terms in that window.
* More the number of query terms, higher the rank of the sentence.
* We then extracted top 3 sentences.
* 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:

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

The evaluation criteria:

1. MAP
2. MRR
3. P@K
4. Precision & Recall
5. Average Precision
6. Reciprocal Rank
7. Average Recall

* We started by retrieving all the relevant document names for each query id given in ‘cacm-rel.txt’.
* We then calculate precision and recall values for each query. Finally, we calculate MAP and MRR. We then repeat this step for each run.
* 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 |

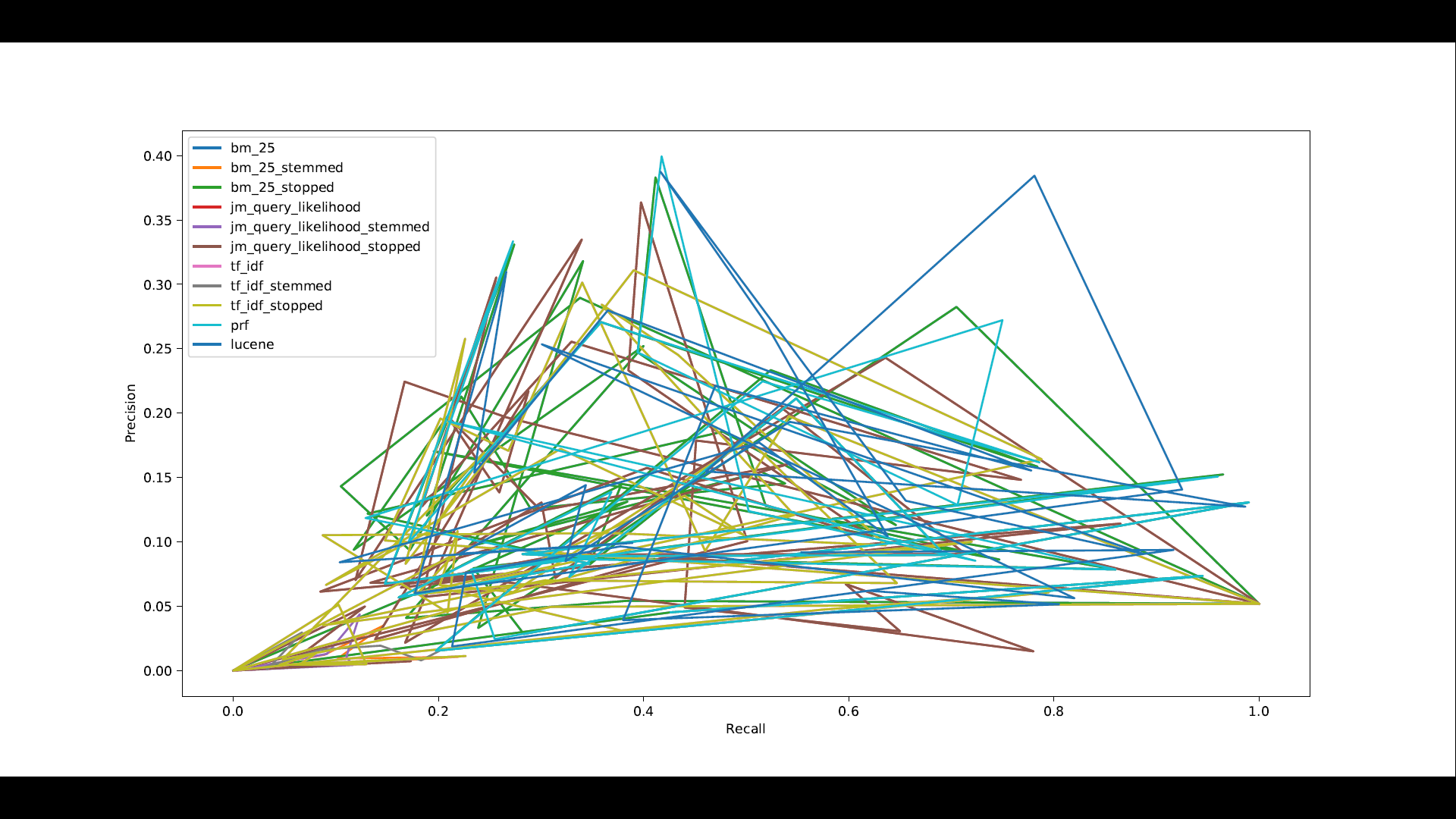
As we can see from the above data, the top document of the BM25 and TFIDF are exactly same. However, for JM smoothing the same document is ranked 2 which is not that drastic of a change. We can also see that the few of the documents that showed up in BM25 is also there in JMSmoothing as well as TFIDF.

**Results**

|  |  |  |
| --- | --- | --- |
| 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/)](https://www.crummy.com/software/BeautifulSoup/))

3. Official Python Documentation (<https://docs.python.org/3/)>