**INFORMATION RETRIEVAL**

**Fall 2018**

**PROJECT REPORT**

# Group Members:

Darshit Chanpura

Dharmish Shah

Soumyadeep Sinha

**Instructor:**

Prof. Nada Naji

2. 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**

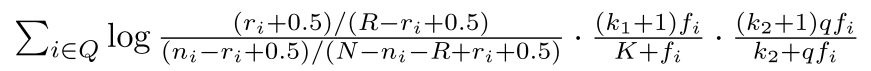
The course textbook and the slides were the primary source of reference. For programming related questions, we used the Python documentation and BeautifulSoup documentation.

**Implementation and Discussion**

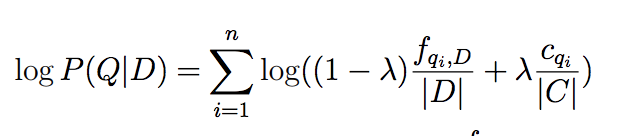
**Phase 1: Indexing and Retrieval**

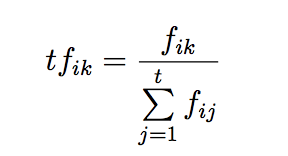
Task 1:

* To start with, we implemented the 4 Retrieval Systems, 3 of which (BM\_25, JM\_Querylikelihood, TF\_IDF) were coded from scratch and 4th was Lucene’s default model.
* We created a cleaned corpus using the corpus generator and then ran the 3 systems on this cleaned corpus.
* BM\_25, the first system: the assumptions – R = 0, ri = 0, b = 0.75, k1 = 1.2, k2 = 100.
* We loop over all the queries.
* We split each query into query terms and remove stop words in the case of stop words.
* Firstly, we calculate K as shown in the formula below.
* While looping over each query term, we find the idf component; and we calculate the tf component for each document, using the formula given below. As mentioned earlier consider relevance data.

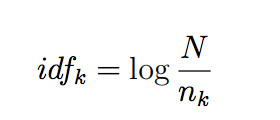


  
where the summation is now over all terms in the query; and N is the set of non-relevant documents, R is the set of relevant documents, ni is the number of non-relevant documents in which the term i is found, and ri is the number of relevant documents in which the term i is found. r and R are set to 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.

* We calculated the partial score using the basic ranking function.
* We have 3 types of runs in this system, one in normal, one is for stemmed corpus and last one is for corpus without stop-words.
* 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.
* Just like the previous system, we start by refining the query and splitting it into query terms. It has 3 versions: normal, with stopping and with stemming.
* We then get the list of docs in which each query term appears.
* We calculate term count per document for all documents, and the words count 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:
* Just like the other two, 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:

1. BM25 with no stemming and no stopping: src/results/bm\_25
2. JM smoothing with no stemming and no stopping: src/results/jm\_query\_likelihood
3. TF.IDF with no stemming and no stopping: 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 on next page:

Task 3:

Part -A:

* We created a list of Stop words based on the common\_words.txt and cleaned the corpus by removing these stop words.
* We then created index of the corpus after removal of stop words.
* Then we performed and retrieved the results for the queries in cacm.query.txt for 3 baseline runs on this newly generated index, namely: BM25, tf.idf and JM Smoothing.

Part-B:

* We first created stemmed corpus of all the 3204 files using camc\_stem.txt and indexed them.
* We performed retrieval on this index by using the stemmed queries given in camc\_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. JM smoothing with stemming: src/results/jm\_query\_likelihood \_stemmed
4. JM smoothing 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 particular 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. JM Smoothing: src/results/jm\_query\_likelihood
5. JM Smoothing stemmed: src/results/jm\_query\_likelihood\_stemmed
6. JM Smoothing 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@5 stored as ‘p\_at\_5.txt’ and P@20 stored as ‘p\_at\_20.txt’.

All of these outputs can be found in “src/evaluation” folder.