Project Report

DESIGN AND IMPLEMENTATION OF INFORMATION SYSTEMS INFORMATION RETREIVAL

Professor: Dr. Nada Naji

Surupa Chatterjee
Srashti Badjatya
Nipesh Roy

Table of Contents

1. Introduction

1.1. Project Description:

Goals: Design and build your information retrieval systems (using core information retrieval concepts and processes), evaluate and compare their performance levels in terms of retrieval effectiveness on the CACM test-collection dataset.

The project was implemented on the dataset in three phases:

- **Phase 1 – Indexing and Retrieval**

As a part of this phase, we performed 3 tasks:

- **Task 1**: In this task, we implemented four retrieval models on a word unigram indexer.
  - *BM25*
  - *Smoothed Query Likelihood Model (λ = 0.35)*
  - *tf-idf*
  - *Lucene's default retrieval model*

The top 100 retrieved ranked lists by each retrieval model were generated as a result of this task.

- **Task 2**: In this task, we implemented *pseudo relevance feedback* for performing *query enrichment* on the results generate in Task 1 for BM25 retrieval model.

As result enriched queries where generated in this task.

- **Task 3**: In this task, we applied the text transformation techniques of *Stopping* and *Stemming* on runs of BM25, Smoothed Query Likelihood Model (λ = 0.35) and tf.idf.

As a result, 6 runs with variation of stopping and stemming on each baseline we generated.

- **Phase 2 – Displaying Results**

In this phase, we implemented a *snippet generation* technique and performed *query term highlighting* on the top 100 ranked results of BM25 retrieval model.

- **Phase 3 – Evaluation**

In this phase, we assessed the performance of the retrieval systems implemented in Phase 1, in terms of their effectiveness.

The evaluation was performed using following parameters:

1. MAP
2. MRR
3. P@K, K=5 and 20 (Precision at rank 5 and 20)
4. Precision and Recall

- **Extra Credits**

For this we implemented a synthetic spelling-error generator and a soft-matching query handler.

1.2. Team Member Contributions:

- Srashti Badjatya: Responsible for implementation of Phase1 (Task1) and Phase 2.

- Surupa Chatterjee: Responsible for implementation of Phase1 (Task2) and Extra Credit 1.

- Nipesh Roy: Responsible for implementation of Phase1 (Task3), Phase3 and Extra Credit 2.

All the team members contributed equally for documenting their implementation.

2. Literature and Resources

2.1. Techniques Used

Overview of retrieval models implemented in the project:

- *Retrieval Models*
  - **BM25 Model**: We used the scoring function of BM25 retrieval model to calculate the document scores. The values for *k1* and *k2* and *b* in the scoring function as per *TREC* standards.
  - **tf.idf Model**: We used the product of normalized value of term frequency and inversed document frequency to calculate the document scores *(tf * idf)*.
  - **Smoothed Query Likelihood Model (*λ = 0.35*)**: We used Bayer's rule for determining probability of each term,

    $$P(D|Q) = P(Q|D) * P(D)$$

    And have used *Jelinek-Mercer Smoothing Technique* with *λ = 0.35* for smoothing.
  - **Lucene's Default Retrieval Model**: We used *Lucene Open Source Library v4.7.2* to determine the ranking for the documents using its standard indexer and *SimpleAnalyzer* as analyzer.
  - **Stopping**: For performing stopping, we remove the common words (stop-words) from the stream of tokens which are to be indexed. The most common words are typically function words that help form sentence structure but contribute little on their own to the description of the topics covered by the text. Examples are "the", "of" , "to", and "for". We have used the given *common_words.txt* for determining the stop-words.
  - **Stemming**: In stemming we group the words that originate from the same stem. We have used the *cacm-stemmed corpus* as the stemmed version of the corpus.
  - **Query Refinement Approach**: The corpus and query should be refined similarly. We have given an option of refinement of queries and corpus by case-folding and punctuation handling.

- **Query Expansion**: We have used Pseudo Relevance Feedback technique for expanding the query.

- **Snippet Generation**: We initially check if tri-grams of the query exists in the documents in the corpus. If it does then we display that tri-gram (as highlighted text) along with 50 characters before and after the tri-gram as document snippet. If no match for tri-grams is found then we repeat the process using bi-grams and then uni-grams of the query.

Overview of the evaluation techniques used to evaluate the performance of retrieval models implemented in the project:

- *Evaluation Techniques*
  - **Precision**: It is the proportion of retrieved documents that are relevant.

$$Precision = \frac{|A \cap B|}{|B|}$$

  - **Recall**: It is the proportion of relevant documents that are retrieved

$$Recall = \frac{|A \cap B|}{|A|}$$

  - **Mean Average Precision (MAP)**: It is the average of average precision of relevant documents for all queries.

  - **Mean Reciprocal Rank (MRR)**: It is the average of reciprocal rank of the queries.

  - **P@K**: It is the precision at k^{th} rank. In this project, we have generated results for k = 5 and 20.

2.2. Research and Articles

- http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/chap11.htm
- http://sigir.org/files/museum/pub-09/VIII-1.pdf
- https://en.wikipedia.org/wiki/Rocchio_algorithm
- https://norvig.com/spell-correct.html
- https://en.wikipedia.org/wiki/Levenshtein_distance

3.Implementation and Discussions

3.1. BM25 Retrieval Model

BM25 extends the scoring function for the binary independence model to include document and query term weights. The extension is based on probabilistic arguments and experimental validation, but it is not a formal model.
There are some variations of the scoring function for BM25, but the most common form is:

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

where k1, k2 and b are parameters whose values are set empirically.
In this implementation k1 = 1.2, k2 = 100 and b = 0.75.
Based on these values, K is calculated by the following formula:

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

where,
dl = document length.
avdl = average document length in the corpus.
qfi = frequency of query words.
N = size of the corpus.
R = number of relevant documents for query (Given R = 0)
ri = number of relevant documents containing the term i. (Given ri = 0)
fi = frequency of term in i^{th} document.

After calculating the scores using the above-mentioned formula, sort the documents as per their scores and return the top 100 documents.

3.2. tf.idf Retrieval Model

Score = (fi/dl) * (1+ log (N/(ni+1)))

where,
dl = document length.
N = size of the corpus.
ni = Number of documents containing i^{th} term.
fi = frequency of i^{th} term in the document.

After calculating the scores using the above-mentioned formula, sort the documents as per their scores and return the top 100 documents.

3.3. Smoothing Query Likelihood Retrieval Model

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

where,

fi = frequency of i^{th} term in the document.
|D| = length of the document
λ = 0.35 (Given)
cqi = number of times the term i occurs in entire corpus.
|*C*| = corpus size

After calculating the scores using the above-mentioned formula, sort the documents as per their scores and return the top 100 documents.

3.4. Pseudo-Relevance Feedback

We have implemented following algorithm for *pseudo relevance feedback:*

- We consider the top 10 documents as relevant documents from the results generated by the BM25 retrieval model.
- We then run *Rocchio's Algorithm* to determine the list of words to be added to query from previously selected documents for query enrichment.

$$\vec{q}_m = \alpha\vec{q}_0 + \beta\frac{1}{|D_r|}\sum_{\vec{d}_j \in D_r}\vec{d}_j - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d}_j \in D_{nr}}\vec{d}_j$$

For optimum results, we did the following:

- Dr,Dnr : Set of relevant and non relevant documents obtained from baseline run of BM25
- q0,qm: q0 is the original query, qm is the modified query after expansion
- The constants α, β, and γ in the *Rocchio's Algorithm* are as per TREC standards i.e. α = 1 β = 0.75 and γ = 0.15.
- The words add to query for enrichment exclude stop-words and numbers at the end of the documents in CACM data collection.
- we expand the query only if it has less than 20 words as adding more terms to an already long query will make it further generic and thus making the results lesser relevant.

3.5. Extra Credits

Task 1: Spelling Error Generator
Here we are trying to generate error in the query term. The strategy used for achieving the output, involved introducing errors randomly at query term level up to 40% of the query length. Terms with longer lengths are affected more with noise than shorter terms. The introduction of spelling errors was done by randomly shuffling characters in a query term. Evaluation was done by comparing overall effectiveness levels when running noisy queries against the noise-free baseline.

Task 2: Soft Query Matching Soft Query Matching
Strives at minimizing the impact on effectiveness introduced by synthetic noise. Assuming the corpus to be in English language, here we have used PyEnchant - a python library, which internally implements Levenshtein's distance measure to fetch a list of probable corrections for the given query term. If the list contains a term that appears in the corpus, we replace that query term with the first probable correction appearing in the corpus for evaluation.

3.6. Query-by-Query Analysis

3.6.1. Phase 3.2

For query-to-query analysis we considered the following 3 stemmed queries:

- Query 1: *parallel algorithm*

Analysis: When we compare the rankings generated by BM25 model with stemming and without stemming versions, we see drastic changes in ranking.

Document	Rank in BM25		Term count in stemmed document		
	w/o Stemming	Stemming	*parallel*	*algorithm*	*algorithms*
CACM-2714	10	1	7	5	0
CACM-2973	1	2	4	3	2

Due to stemming the term ***algorithms*** is grouped with the word *algorithm*. Because of which in BM25 (stemming version) the documents with term *algorithm* are also considered unlike in in BM25 (without stemming version) which only considered documents having the term *algorithms*. Due to increased frequency of term *algorithm (due to stemming)* in CACM-2714 its ranking shots up.

- Query 2: *code optim for space effici*

Analysis: When we compare the rankings generated by BM25 model with stemming and without stemming versions, we see drastic changes in ranking.

Document	Rank in BM25		Term count in stemmed document			
	w/o Stemming	Stemming	*code (372)*	*space (165)*	*effici (282)*	*optim (299)*
CACM-2748	4	1	5	1	1	0
CACM-1947	1	4	2	0	2	2

Note: The value in brackets under each term is their corpus frequency, which is used to determine if a word is more important than other. The word with least corpus frequency is considered more important.

The document CACM- 2748 does not contain *efficiency* as a term instead it has the term *efficient*. Since stemming groups all the terms with same stem, after stemming the term *efficient* in CACM- 2748 is stemmed to *effici* which matches the term *effici* in stemmed query. Also, the document contains the more of important terms than CACM- 1947. Therefore, the ranking of document CACM- 2748 improves.

The ranking of document CACM-1947 is also affected by the fact that term *optimization* is reduced to stem word *optim*. Thus, CACM-1947 now competes with documents which contains terms which have the stem *optim*, like: optimized, optimal etc unlike before stemming when it had to compete against only those documents that had the term *optimization.*

- Query 3: *parallel processor in inform retriev*

Analysis: When we compare the rankings generated by BM25 model with stemming and without stemming versions, we see drastic changes in ranking.

Document	Rank in BM25		Term count in stemmed document			
	w/o Stemming	Stemming	*parallel (168)*	*processor (178)*	*inform (521)*	*retriev (275)*
CACM-1811	4	1	5	5	0	0

CACM-2714	1	3	7	3	0	0

Note: The value in brackets under each term is their corpus frequency, which is used to determine if a word is more important than other. The word with least corpus frequency is considered more important.

Due to stemming the term ***processors*** is grouped with the word *processor*. Because of which in BM25 (stemming version) the documents with term *processor* are also considered unlike in in BM25 (without stemming version) which only considered documents having the term *processors*. Due to increased frequency of term *processor (due to stemming)* in CACM-1811 its ranking shots up.

4. Results

4.1. Query Runs and Evaluation Measures Tables

- The results for all the baseline retrieval models are available as text files which can be located in their respective phases and tasks folders.
- All the *precision* and *recall* tables for the 8 distinct runs can be found in the *evaluation_results* folder in the Phase3(Evaluation) folder.

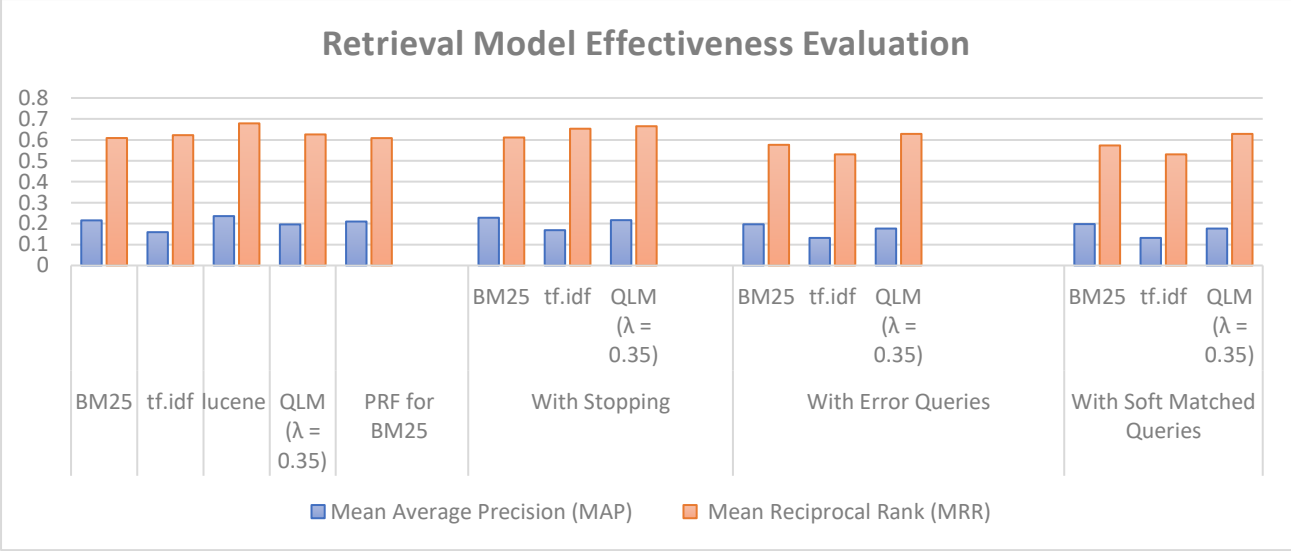
4.2 Comparing Evaluation Results

A comparison of *Mean Average Precision* (MAP) and *Mean Reciprocal Rank* (MRR) for different retrieval models:

- with/without stopping
- with/without query enrichment
- with/without query with errors
- with/without soft matching queries

Retrieval Model		Mean Average Precision (MAP)	Mean Reciprocal Rank (MRR)
BM25		0.215475655	0.608763443
tf.idf		0.159177887	0.622520032
lucene		0.235867631	0.678788503
QLM (λ = 0.35)		0.19614758	0.625911168
PRF for BM25		0.209974554	0.608363287
With Stopping	BM25	0.228065482	0.611381153
	tf.idf	0.168652764	0.653231946
	QLM (λ = 0.35)	0.216545719	0.665158132
With Error Queries	BM25	0.19678668	0.57632081
	tf.idf	0.131714953	0.53075319
	QLM (λ = 0.35)	0.176437945	0.628535674
With Soft Matched Queries	BM25	0.197726025	0.573288781
	tf.idf	0.131714953	0.53075319
	QLM (λ = 0.35)	0.176437945	0.628535674
		PRF	Psuedo Relevance Feedback
		QLM	Query Likelihood Model

A graph is plotted using MAP and MRR values for each run for providing further information.



5. Conclusion and Outlook

5.1 Conclusion

As from the evaluations and results, we can see that BM25 retrieval model provides a better ranking for relevant documents w.r.t any given query as well as it fetches most of the relevant documents (as per the relevance information given in *cacm.rel.txt*) unlike other models.

Besides this, BM25 model has better scores for both MAP and MRR, unlike other models which have better score for only one of the parameters.

Therefore, we concluded that BM25 retrieval model performs better than other three models.

5.2 Outlook

The project implements several features of the IR system. However, there are areas of improvement and it can be made better by including following features:

- In the document scoring functions of each model we could include PageRank algorithm in-order to perform document based early termination. By leaving out the lower ranking documents in search the efficiency of the system can be improved.
- In order to reduce space usage, we can use different lossless compression techniques for storing inverted indexes.
- In snippet generation, we do not currently check if the snippet is a meaningful sentence or not. In future, we can try and display only meaningful sentences.

6. Bibliography

6.1. Books

- *Search Engines Information Retrieval in Practice* - W. Bruce Cro Donald Metzler Trevor Strohman
- *An Introduction to Information Retrieval* - Christopher D. Manning Prabhakar Raghavan Hinrich Schütze

6.2. Articles, Papers and Websites

- https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- http://docs.python-requests.org/en/master/user/quickstart/
- https://nlp.stanford.edu/IR-book/html/htmledition/relevance-feedback-and-query-expansion-1.html
- http://www.pythonforbeginners.com/requests/the-awesome-requests-module
- http://www.pythonforbeginners.com/beautifulsoup/beautifulsoup-4-python
- https://www.codecademy.com/learn/learn-python
- https://docs.python.org/2/library/traceback.html
- https://docs.python.org/2/library/os.html
- https://docs.python.org/3/library/collections.html
- https://docs.python.org/2/library/sets.html
- Stackoverflow for syntax, understanding certain concepts, like retrieval model formulae, application of python dictionaries, string library of python etc.