Implementation of Search Engine

CS6200 : Information Retrieval
Fall 2018
Prof. Nada Naji

by
Amit Kulkarni (NUID : 001826355)
Aayush Jain (NUID : 001211069)

College of Computer and Information Science
Northeastern University
Boston, MA

Contents

Introduction

1.1 Project Overview

The goal of the project was to apply and implement core concepts of Information Retrieval learnt as part of CS6200 and build retrieval systems.Our implementation is specific to the CACM data
The following retrieval models / techniques were implemented as part to the project:

1. BM25 Retrieval Model
2. TF-IDF Retrieval Model
3. Jelinek-Mercer Smoothing Technique
4. Lucenes Retrieval System
5. Query Enrichment using Pseudo-Relevance Feedback (for BM25 Retrieval Model)
6. Retrieval Models with Stop Words
7. Retrieval Models using Stemming
8. Snippet Generation
9. Performance Effectiveness (Using techniques such as Mean Average Precision, Mean Reciprocal Rank, Precision and Recall, Precision @ K)

1.2 Contribution

We had a team of 2 members, Aayush Jain and Amit Kulkarni.The individual contributions are as follows:

- Amit Kulkarni: Implemented the entirety of Phase 1 and Extra Credits
- Aayush Jain: Implemented the entirety of Phase 2 and Phase 3

Literature and Resources

2.1 Overview of Techniques

2.1.1 Phase 1 :: Indexing and Retrieval

- Task 1
  - Corpus Cleaning: Cleaned the corpus by removing punctuation, applied case folding case-folding and ignored the table of numbers at the end of each cacm*.html file.
  - Indexing: Used unigram indexing to index the entire cleaned corpus
  - Retrieval Models: Applied standard formulae for BM25, TF-IDF and JM-QLM models and ranked documents based on their scores as per queries (provided in cacm.query.txt file.)
  - Lucene: Used Lucenes APIs to perform indexing and ranking of documents
- Task 2
  - Pseudo-Relevance Feedback: Implemented Rocchio algorithm
- Task 3
  - Stopping: Used stop words provided in the common_words file to perform indexing and ranking of documents
  - Stemming: Parsed and cleaned the stemmed version of the corpus (in cacm.stem.txt) and performed indexing and ranking of documents as per queries (in cacm_stem.query.txt file)

2.1.2 Phase 2 :: Displaying Results

For snippet generation we used Luhn's algorithm with significance factor. We used a sentence length of 50 and calculated the significance factor for the sentence based on the presence of significant words with a sliding window approach. We highlighted the significant words and presented them in an HTML

2.1.3 Phase 3 :: Evaluation

For evaluation of our retrieval systems, we used 5 metrics:

- MAP : Mean Average Precision
- MRR : Mean Reciprocal Rank
- P@K : Precision at rank K
- Precision and Recall table
- Plot of Recall vs Precision

2.1.4 Extra-Credit

1. **Exact Match**: The task entailed ranking documents based on queries such that the ranked documents contained all the query terms in the exact order as they appear in the query. This is basically a Boolean AND query with the addition of query term ordering. We created a positional inverted index which is of the form $\{term : \{docID : [freq, [pos_1\ term, pos_2\ term...]]\}\}$ and used this index to rank documents. The scoring function used here was to just sum up the frequency of query terms appearing in the document.

2. **Best Match**: In this scenario, a document was ranked even if 1 of the query terms was present in the document. This is a Boolean OR query. The normal inverted index was used for this task. The scoring function again was to sum up the frequency of terms in the document.

3. **Ordered Proximity Match within proximity N**: The documents were to be ranked in a similar way as the Exact Match task with the additional condition that no more than N terms were to be present between the query terms. Used positional index for this task. N (which refers to the window size) was accepted as an input from the user.

2.2 References

Pseudo Relevance Feedback - Rocchio Algorithm

Implementation and Discussions

3.1 Phase 1 - Baseline Runs

1. **BM25**
   Following formula was used to calculate the BM25 score of the documents

$$\sum_{i \in Q} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} * \frac{(k_1 + 1)f_i}{K + f_i} * \frac{(k_2 + 1)qf_i}{k_2 + qf_i} \tag{3.1}$$

   where R : Total number of documents for query
   r_i : The number of relevant documents containing the query term
   N : Total number of documents in the collection
   n_i : Number of documents containing the query term
   f_i : frequency of query term i in the document
   qf_i : Frequency of query term i in the query
   k_1, b, k_2 = 1.2, 0.75, 100

$$K = k_1((1 - b) + b * \frac{dl}{avdl} \tag{3.2}$$

   dl : Document length
   $avdl$: Average document length in the entire collection

2. **TF-IDF**
   Used the following formula to calculate scores for each document

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}} \tag{3.3}$$

$$idf_k = log \frac{N}{n_k} \tag{3.4}$$

3. **Jelinek-Mercer Smoothing**

$$logP(Q|D) = \sum_{i=1}^{n} log((1 - \lambda)\frac{f_{qi,D}}{|D|} + \lambda \frac{c_{qi}}{|C|}) \tag{3.5}$$

$f_{qi,D}$: Frequency of query term i in document D
$|D|$: Length of document D
$|C|$: The length of the entire collection
c_{qi} : Frequency of query term i in collection
λ : Smoothing constant (small values of lambda produce less smoothing and the query behaves like a boolean AND)

3.2 Phase 1 - Pseudo Relevance Feedback

Relevance Feedback generally improves recall and precision. The idea of pseudo-relevance feedback is to involve the user in the retrieval process to improve the final set. Our implementation used the Rocchio algorithm (2) with the formula as below:

$$\vec{q_m} = \alpha\vec{q_0} + \beta\frac{1}{|D_r|}\sum_{\vec{d_j}\in D_r}\vec{d_j} - \gamma\frac{1}{|D_{nr}|}\sum_{\vec{d_j}\in D_r}\vec{d_j} \tag{3.6}$$

where,
q_m : modified query vector
q_0 : original query vector
$|D_r|$: magnitude of the relevance vector
$|D_{nr}|$: magnitude of non-relevance vector
α, β, γ are parameters which control the effect of each component
In our implementation, we chose the following values: $\alpha, \beta, \gamma = 1$, 0.75 and 0.15

Logically this seems right as we are assigning more weight (β) to relevant documents and less weight (γ) to non-relevant documents

The following steps were followed to incorporate a relevance feedback mechanism:

- Rank documents according to BM25 model
- Include k (= 10) documents in relevant set
- Generate query vector, relevance vector and non-relevance vector
- Apply Rocchio algorithm to create a new expanded query with top 30 terms being appended to the original query
- Score the documents as per these new queries

3.3 Phase 1 - Query-by-Query Analysis

The task here was to select 3 queries (un-stemmed and corresponding stemmed) queries and examine the results.
The 3 queries that we decided to choose were:

1. "Parallel algorithms" (un-stemmed) vs "parallel algorithm"(stemmed version)
    - The reason for choosing these 2 queries were because the original query matched very well with the stem queries. Ideally there should be high number of matches in the retrieved documents for both the queries.
    - There were a total of 6 documents that overlapped of the top 10 results for both the queries. (CACM-2266, CACM-2973, CACM-3075, CACM-3156, CACM-0950, CACM-2289)
2. "portable operating systems" (un-stemmed) vs "portabl oper system" (stemmed version)
    - The reason for choosing this query was that 2 / 3 words changed in the stemmed version. The number of documents overlap shouldnt be either too high or too low
    - There were a total of 5 documents that overlapped in the top 10 results using the BM25 ranking (CACM-3127, CACM-2246, CACM-1930, CACM-3196, CACM-3068)
3. "Performance evaluation and modelling of computer systems" (un-stemmed version) vs "perform evalu and model of comput system" (stemmed version)
    - The reason for choosing these queries was the high degree of change in the query terms between un-stemmed and stemmed version of the query. This should result in a very low number of documents overlap.
    - There were a total of 4 documents that overlapped for the top 10 results of both the queries using the BM25 ranking (CACM-2318, CACM-3070, CACM-2741, CACM-2319)

3.4 Phase 2 - Snippet Generation and Query Term Highlighting

Luhn proposed an approach to generate the snippets using a significance factor (1). This significance factor can be based on frequency of query terms or on the presence of query terms alone. We have implemented Luhn's approach with query terms as

the significant terms. Our sentence selection criteria is one where a sentence is either 50 words long or one thirds of the total length of document, whichever is smaller. The steps are discussed below:

1. We have a sliding window of size mentioned above
2. Significant words are identified in the window
3. A significance score is calculated by the formula Luhn proposed $\frac{(significant\ terms)^2}{window\ length}$
4. Window is slid ahead by one word and the significance score is generated for the next window.
5. Window with the maximum score is selected as the snippet

For Query Term Highlighting, we present our snippets in an html file, where all the significant terms are bold and can easily be identified.

3.5 Evaluation

To measure the quality of our retrieval system, we use five metrics for evaluation. MAP, MRR, P@K, Precison-Recall table, Precision-Recall Plot

1. Precision
   Precision measure how well a retrieval system is doing at rejecting non-relevant documents. It is defined as the number of relevant document that the system retrieved divided by the number of documents retrieved
   $$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|} \tag{3.7}$$
2. Recall
   Measures how well a search engine is doing at finding the relevant document
   $$Precision = \frac{|Relevant \cap Retrieved|}{|Relevant|} \tag{3.8}$$
3. Mean Average Precision
   It is the mean of all average precision for all queries. It is most commonly used measure in research papers. It assumes user is interested in finding many relevant document for each query
4. Mean Reciprocal Rank
   Rank is the position at which the first relevant document occurs. Reciprocal rank is the inverse of rank and mean reciprocal rank if the average of all reciprocal ranks for all the queries

5. Precision at K = 5 and K = 20

   It evaluates at performance of a system by measuring the precision at higher rank. More relevant document in higher rank would signify higher better retrieval and hence better P@K

3.6 Extra Credits

For both Exact Match and Ordered Proximity Match tasks a positional inverted index was used. The positional inverted index is of the form $\{term : \{docID : [freq, [pos_1\ term, pos_2\ term.]]\}\}$. The pseudo code to determine exact match was along the below lines:

```
for doc in collection:
        for i in len(query - 1):
        term_i, term[i+1] = query[i], query[i+1]
        if term_i not in POSITION_INVERTED_INDEX:
            Do not rank documents based on such a query
         else:
            if doc not in POSITION_INVERTED_INDEX[term_i]:
                Do not rank such a document
            else:
                # Get the position of term_i and term_i+1
                if POSITION_INVERTED_INDEX[term_i][doc][1][0] >
                   POSITION_INVERTED_INDEX[term_i+1][doc][1][0]:
                    Do not rank such a document
                else:
                    # Everything went through fine
                    score[doc] += freq[term_i]
```

The ordered proximity match also worked on the similar lines (with an additional condition that distance between query terms should be less than window size).

The query "parallel algorithms" produced a high number of ranked documents for Exact Match and Ordered Proximity match, with the reason being that the query is short in nature (boolean AND works well here). A "No Results Found" message was the output for queries where there were no ranked documents.

Results

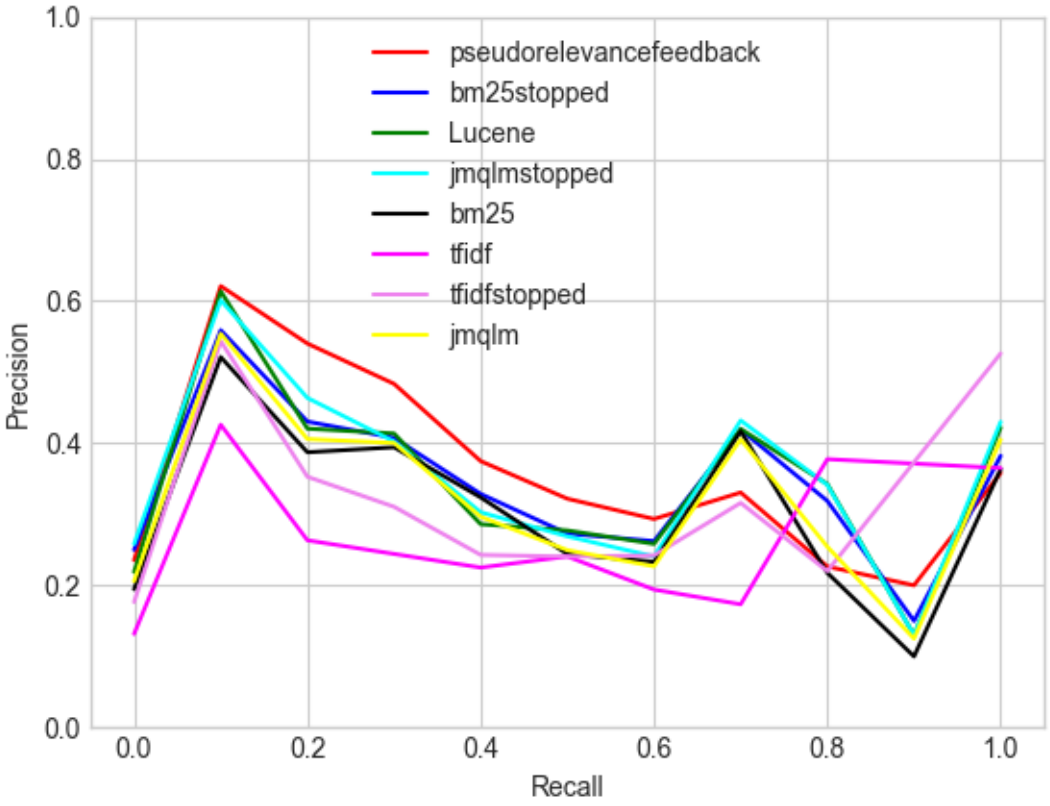
Results for baseline runs, enhancement runs and stopping runs are attached in the folder with name Output. Precision_recall tables have MAP and MRR at end of the file

Conclusions and Outlook

5.1 Conclusion

On performing the evaluations we have the following observations:

- BM25 with pseudo relevance feedback performed the best with highest MAP and MRR.
- ifidf was the worst performer out of all with lowest MAP and MRR
- Query Likelihood Model with JM smoothing performed on par with the BM25 model
- Stopping boosted performance of all the models with significant improvement in the MAP value
- BM25 with pseudo relevance feedback retrieved relevant documents at higher rank with highest MRR. tfidf had the lowest, JMQLM on par with Lucene
- From the plot we can verify the trend that are reflected in the MAP and MRR values



5.2 Outlook

The following actions could be taken to improve the project:

1. A GUI for the user to enter the query and retrieve Top N documents
2. Actual asking for user for relevance feedback or inherently inferring relevance feedback by following user actions through query logs.
3. Incorporate Spell-Check and Did You Mean feature based on edit-distance
4. Add a greater number of stop words (Come up with a generalize approach to classify words as being stop words)

Bibliography

[1] W. Bruce Croft Donald Metzler Trevor Strohman *Search Engines Information Retrieval in Practice*. Pearson Education, Inc. 2015

[2] Rocchio Algorithm *https://en.wikipedia.org/wiki/Rocchio_algorithm https://nlp.stanford.edu/IR-book/html/htmledition/the-rocchio71-algorithm-1.html*

[3] Lucene *https://lucene.apache.org/core/quickstart.html*

[4] Beautiful Soup *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*

[5] BM25 *https://en.wikipedia.org/wiki/Okapi_BM25*