Search Engine Implementation

CS6200: Information Retrieval

Northeastern University

Fall 2017, Prof. Nada Naji

By
Ritika Nair

Saptaparna Das

Grishma Thakkar

Table of Contents

1. Introduction:

1.1 Overview:

The goal of this project is to implement core concepts of Information Retrieval system by developing a search engine from scratch. The search engine uses four retrieval models, mentioned below and compares the results by evaluating the retrieval effectiveness.

Retrieval models used:

- BM25 Retrieval Model
- TF-IDF Retrieval Model
- Smoothed Query Likelihood Model
- Lucene's default retrieval System

Also, we have implemented query expansion technique and ran BM25 Retrieval Model with Pseudo Relevance Feedback. Additionally, BM25, TF-IDF and Query Likelihood models are run with Stopping and Stemming techniques.

To analyze the performance of these retrieval models, we evaluate them in terms of retrieval effectiveness, using the following techniques:

- Mean Average Precision (MAP)
- Mean Reciprocal Rank (MRR)
- P@K measures where k=5 and k=20
- Precision and Recall

Along with these, to display the result to the user, we have used Snippet Generation and Query Term Highlighting techniques on BM25 Retrieval results.

1.2 Contribution of team members:

- **Ritika Nair:** was responsible for implementing query expansion technique using Pseudo-relevance feedback and proximity enabled search. Additionally, she was involved in developing Snippet Generation module and BM25 retrieval model. Moreover, she made significant contribution in modularizing and integrating individual modules.
- **Saptaparna Das:** was responsible for developing QLM, TF-IDF retrieval models. She also implemented Stopping technique on the base line runs and was involved in designing and implementing Snippet Generation module. In addition, she made major contribution in documenting the design choices, findings and analysis.
- **Grishma Thakkar**: was responsible for implementing various measures of Evaluation for all the eight runs. Also, she developed Stemming techniques for top retrieval modules. She was responsible for implementing Lucene run on given data set as well as documenting the changes.

2. Literature and Resources:

2.1 Overview:

An overview of techniques used in the project is mentioned below:

- Phase 1: Indexing and Retrieval:
- Task1:
    - Indexer: We have used unigram indexer. It creates an inverted index in the form "Term : [DocID: TF]" where Term is a word in the document, DocID is name of the document and TF is frequency of term in the document.
    - BM25: For BM25 model, we have taken into consideration the relevance judgement by using 'cacm.rel' file and the values of 'k', 'k1', 'k2' are selected as per TREC standards.
    - TF-IDF: For TF-IDF we are calculating normalized value of term frequency multiplied by inverse document frequency.
    - Query Likelihood: For smoothed query likelihood model we have used Jelinek-Mercer method of smoothing with a constant of value 0.35.
    - Lucene: We have used standard Lucene open source library (version 4.7.2) to index and rank documents.
- Task2 : Pseudo Relevance Feedback:

  The pseudo relevance technique is implanted using Rocchio algorithm in our project.
- Task 3 : Stopping and stemming:

  For stopping, we have used 'common-words.txt' to skip indexing any word, that belong to the document.

  For stemming, we have used 'cacm_stem' as corpus and 'cacm_stem.query' as list of stemmed queries.
- Phase 2 : Displaying Results:

  For displaying the results, we have used Snippet Generation and Query Term Highlighting. We have used basic Luhn's algorithm to select significant words, with an improvement of selecting query terms as well to select a better span for the snippet.
- Phase 3 : Evaluation:

  Evaluation of retrieval effectiveness is finally done by using following measures: MAP, MRR, P@K and Precision & Recall.
- Proximity-enabled Search (**Extra Credit**):

  Modified the existing unigram indexer to store position of each term in document. We have used Buttcher's Scoring Model to boost the scores of documents in which query terms appear in max three words proximity and in same order as that of query. The search is performed with stopping and without stopping.

2.2 Third-party tools:

We have used the following third-party tools throughout the development of our project:

- Lucene libraries: We used below mentioned libraries:
    - lucene-core-VERSION.jar
    - lucene-queryparser-VERSION.jar
    - lucene-analyzers-common.jar
- Beautiful SOUP: Beautiful SOUP is used to process the documents of corpus and queries.

2.3 Research article references:

The below mentioned articles were considered while implementing proximity enabled search

- https://pdfs.semanticscholar.org/472d/d74882ff85d89fc70b54e88fc9f40d3c1380.pdf

  Efficient Text Proximity Search by Ralf Schenkel1, Andreas Broschart1, Seungwon Hwang2, Martin Theobald3, and Gerhard Weikum1

The below mentioned articles were considered for implementing Pseudo Relevance feedback:

- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3553&rep=rep1&type=pdf

  Improving Retrieval Performance by Relevance Feedback. Gerard Salton; Chris Buckley. Journal of the American Society for Information Science (1986-1998); Jun 1990

3. Implementation and Discussion:

3.1 Baseline Runs:

- 3.1.1 BM25 Model: BM25 is a ranking algorithm that is an extended version of the binary independence model which includes query term weights to rank documents. Our implementation of BM25 uses the following formula:

  $\Sigma \log \{(r_i + 0.5) / (R - r_i + 0.5)\} / \{(n_i - r_i + 0.5) / (N - n_i - R + r_i + 0.5)\} * (k_1 + 1) f_i / (K + f_i) * (k_2 + 1) qf_i / (k_2 + qf_i)$

  Where:
    - r_i is number of relevant documents containing the term i
    - n_i is number of documents containing the term i
    - N is total number of documents in the corpus
    - R is total number of relevant documents for the query
    - f_i is frequency of the term i in the document
    - qf_i is frequency of the term i in the query
    - k_1 the value is chosen as 1.2

- k_2 the value is chosen as 100
- K -> K = k_1 ((1 – b) + b * dl /avdl) where dl is document length, avdl is average length of document in the collection.
- The value of b is chosen as 0.75

Documents are sorted in descending order of their scores and top 100 documents are printed in 'BM25RelevanceRun.txt' file in below format:

query_id Q0 doc_id rank BM25_score BM25RelevanceModel

- 3.1.2 TF-IDF Model: For TF-IDF measure we have multiplied normalized tf with idf.
  - tf is calculated as number of occurrences of term in a document divided by total number of terms in that document
  - idf is calculated by adding 1 to log of total number of documents divided by 1 + number of documents in which term appears. 1 is added to prevent it from becoming zero.

Documents are sorted in descending order of their scores and top 100 documents are printed in 'TFIDFRun.txt' file in below format:

query_id Q0 doc_id rank TF-IDF_score TFIDFModel

- 3.1.3 Query Likelihood Model: we rank documents by the probability that the query text could be generated by the document language model. In other words, we calculate the probability that we could pull the query words out of the "bucket" of words representing the document. This is calculated by following formula:

$$\log P(Q/D) = \Sigma_{ni=1} \log((1 - \lambda) f_{qi;D} /D + \lambda c_{qi}/C)$$

where

- λ is 0.35 as per TREC standard
- fq is number of times word q occurs in document
- D is number of words in doc
- Cq is number of times word q occurs in corpus
- C is total number of words in corpus

Documents are sorted in descending order of their scores and top 100 documents are printed in 'QLRun.txt' file in below format:

query_id Q0 doc_id rank TF-IDF_score QLModel

3.2 Pseudo relevance feedback:

Our implementation of pseudo-relevance feedback utilizes the 'Rocchio algorithm' explained in the course material. This algorithm modifies the initial weights in the query vector to produce an expanded query, by maximizing the difference between the average vector representing the relevant documents and that representing the non-relevant documents.

The formula used for generating the expanded query is as follows:

$$q'_j = \alpha.q_j + \beta.\frac{1}{|Rel|}\sum_{D_i \in Rel} d_{ij} - \gamma.\frac{1}{|Nonrel|}\sum_{D_i \in Nonrel} d_{ij}$$

Where qj is the initial weight of query term j, Rel is the set of identified relevant documents, Nonrel is the set of non-relevant documents, |.| denotes the size of a set, dij is the weight of the jth term in document I, and α, β, and γ are parameters that control the effect of each component.

For our project we have chosen the values α = 1.0, β = 0.75, and γ = 0.15.

These values were chosen based on the experiments described in the scholarly article "*Improving Retrieval Performance by Relevance Feedback." Gerard Salton; Chris Buckley. Journal of the American Society for Information Science (1986-1998); Jun 1990."*

The parameter γ was assigned the smallest value out of the three to reduce the contribution by non-relevant documents, and the parameter β was chosen to be higher to boost the contribution by the relevant documents.

The steps followed to implement the same is as follows:

- Retrieval of documents was performed for the initial query using BM25 model.
- The top 'k' documents (k=10) was chosen to be included in the relevant set of documents.
- The Rocchio algorithm formula was applied and the top 20 terms with highest weights were appended to the query.
- Retrieval was performed for the new query with the BM25 model.

3.3 Snippet generation and Query term highlighting:

For snippet generation, we have used basic Luhn's algorithm with minor variation in choosing significant words to improve upon it. The steps followed to implement the same is as follows:

- For each document, assign significance factor to each sentence. Calculate the significant words by the formula:

$$f_{d,w} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), & \text{if } s_d < 25 \\ 7, & \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), & \text{otherwise,} \end{cases}$$

Where sd is number of sentences in document, f is frequency of word w in document d.

- The words in query are also added to list of significant words to consider relevance and produce better quality results.
- The significance factor for each sentence is calculated by generating a span of words from first occurrence of significant word to last occurrence. Now, factor is calculated by taking square of number of significant words in the span divided by total number of words in the span.

- Sentences are sorted in descending order as per their scores and top 4 are displayed per document with query terms highlighted. Stop words are filtered out while getting significant words as well as while highlighting the same.

3.4 Evaluation:

Evaluation of corpus is essential to get a sense of the efficiency and effectiveness of an Information retrieval system. For our system, we have used precision, recall, MAP, MRR, P@5, P@20.

- Precision:
  The ratio of the number of the relevant documents that the system was able to retrieve for a given query, to the number the retrieved documents.

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

- Recall:
  the ratio of the number of the relevant documents that the system was able to retrieve for a given query, to the number of the relevant documents for that query (retrieved or le` out).

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

- MAP:
  MAP stands for Mean Average Precision, i.e. it is the mean of all the average precisions for all the queries. Average Precision for a query, is the ratio of sum of precision values when relevant document is found to the total number of relevant documents for that query.
  It is used to demonstrate and evaluate how effectively the documents have been ranked for every query.

- Precision at K =5 and K=20:
  Precision at K=5, and K=20 measures the performance of the search engine, i.e. how many relevant documents have been retrieved at higher ranks, higher precision values at higher ranks translates to more relevant documents being retrieved at higher ranks, which means that the Information Retrieval system is highly efficient.

- MRR:
  MRR stands for Mean reciprocal rank, it is the mean of all reciprocal ranks for all the queries. Reciprocal rank is defined as the reciprocal of the rank at which the first relevant document is retrieved.

3.5 Query by query analysis:

The two queries we have selected for analysis are:

- Parallel algorithms (un-stemmed) / parallel algorithm (stemmed)
- Performance evaluation and modelling of computer systems (un-stemmed)/ perform evalu and model of comput system (stemmed)
- Parallel processors in information retrieval (un-stemmed) / parallel processor in inform retriev (stemmed)

The first query is chosen because both the stemmed and un-stemmed versions are having almost same words except for (algorithms/algorithm). So, there should be high degree of overlap between baseline runs. When we considered BM25 run with and without stemming, there are 13 common documents out of top 20 results namely:

In top20: CACM-2973 , CACM-1601 , CACM-0950, CACM-2557 , CACM-2714, CACM-2685, CACM-1262, CACM-2700, CACM-2896, CACM-1158, CACM-0141, CACM-2785, CACM-2433

For the second query, we can see there are five words that's changed in stemming. Words like 'evaluation', 'evaluations', 'evaluate',' evaluates', 'evaluative', 'evaluated' etc. will be replaced by stem class 'evalu'. So, all the documents having these words also will be taken into consideration as relevant documents and hence, the score will be boosted with stemming.

If we compare, BM25 results of the query with and without stemming, we can see there only 6 common documents namely:

In Top 20: CACM-2318, CACM-3070, CACM-2988, CACM-2741, CACM-2319, CACM-2812

In the third query, there are again three words that changed after stemming. So, BM25 result of the query with and without stemming shouldn't have high overlap. When we ran the results, there is only 1 common document namely:

In Top 20: CACM-2714

3.6 Proximity-enabled search **(Extra Credit)**:

For Proximity-enabled search we have modified the unigram indexer to store position of terms. So, the scores of the documents in which query terms are in close proximity (not more than 3 words) are boosted. This score is integrated with our BM25 model scores to produce final ranking.

The boosting factor is calculated by using basic '**Buttcher's Scoring Model**' according to the below formula:

$$acc_d(t_k) = \sum_{(i,j) \in A_d(q): p_i = t_k} \frac{idf(p_j)}{(i-j)^2} + \sum_{(i,j) \in A_d(q): p_j = t_k} \frac{idf(p_i)}{(i-j)^2}$$

Where

- acc is accumulated interim score for each query term that depends on the distance of this term's occurrences to other, adjacent query term occurrences
- pj is j th query term, pi is ith query term.
- idf is the inverse document frequency

idf is used to normalize the scores and it is divided by square of index difference (proximity) to boost the score so that the accumulated proximity score increases for terms that are more adjacent to each other and the less frequent the neighboring term is in the collection. We have implemented this by following the below steps:

- Score for a document is calculated by considering the occurrence of all pairs of query terms in the document. If any pair of query terms is present in the document in the same order, we calculate the smallest difference between their positions.
- If the smallest difference shows that they are in close proximity (no more than 3 words apart), then we calculate the score using above mentioned formula.
- These scores are accumulated in a dictionary for all documents.
- We populate the same dictionary to accumulate BM25 scores as well for all the documents.
- Hence, the final ranking of the documents will have BM25 scores boosted with proximity information.

4. Results:

Results for all the baseline runs, along with stopping stemming, pseudo relevance feedback and retrieval effectiveness measures are saved in spread sheet and attached below:

EVALUATION OUTPUT.xlsx

5. Conclusions and Outlook:

5.1 Conclusion:

By evaluating these models, we found that:

- QLM model performed poorly when compared to the other models.
- tf-idf performed better than QLN but not still wasn't as effective as the other models.
- Stopping boosted tf-idf model and QLM model, however, it negatively impacted the performance of BM25.
- BM25 retrieved relevant documents at higher rank and hence its performance was very efficient and effective as its MAP was the highest.
- BM25's performance is better than BM25 with pseudo relevance feedback. Pseudo relevance feedback assumes that all the top k documents are relevant, and the documents are re-ranked based on that assumption, which is not always true especially if non-relevant documents are retrieved at higher ranks.

5.2 Outlook:

- This prototype information retrieval model can be enhanced by:
- Accumulating the query logs and relevance feedback from the user.
- Customized generation of stop words based on the application.
- A spell-check can be incorporated in the project.
- A UI will make the information retrieval system more user friendly.
- Using multiple collections to evaluate the performance of the information retrieval system.
- Using supervised and unsupervised learning algorithms like classifications and clustering can further improve the efficiency of the system.
- Snipped generation we can incorporate importance of placement of text inside various tags, for example, text inside the head tag may be more significant that the text inside the body.
- Design the system to incorporate better results for longer queries and queries that are questions.

6. Bibliography:

6.1 Books:

- Croft, W.Bruce; Metzler, Donald; Strohman, Trevor *Search Engines: Information Retrieval in Practice.* Pearson Education 2015
- Manning, Christopher D; Raghavan, Prabhakar; Schutze Hinrich *An Introduction to Information Retrieval.*

6.2 Scholarly articles:

- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3553&rep=rep1&type=pdf

  Improving Retrieval Performance by Relevance Feedback. Gerard Salton; Chris Buckley. Journal of the American Society for Information Science (1986-1998); Jun 1990
- https://pdfs.semanticscholar.org/472d/d74882ff85d89fc70b54e88fc9f40d3c1380.pdf

  Efficient Text Proximity Search by Ralf Schenkel1, Andreas Broschart1, Seungwon Hwang2, Martin Theobald3, and Gerhard Weikum1
- https://arxiv.org/ftp/arxiv/papers/1502/1502.05168.pdf

6.3 Websites:

- https://www.udacity.com/course/intro-to-computer-science--cs101 : Basics of Python Programming and web crawling
- https://www.crummy.com/software/BeautifulSoup/ : BeautifulSoup has been used for extracting links from web pages
- https://learnpythonthehardway.org/book/ : Python Programming
- https://en.wikipedia.org/wiki/Rocchio_algorithm
- http://maroo.cs.umass.edu/getpdf.php?id=630