Search Engine Implementation

CS6200: Information Retrieval

Northeastern University

Spring 2019, Prof. Rukmini Vijaykumar

By
Shruti Parpattedar
Poorva Sonparote
Nanditha Sundararajan

Table of Contents

1. Introduction:

1.1 Overview:

For our final project where we build our own retrieval systems, we implemented four retrieval models and evaluated their performance. We used the CACM corpus, stemmed it and removed the stop words using the stoplist provided to us and analyzed each model's performance. Additionally, we also generated document rankings for queries using Lucene. The Retrieval models used are:

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

We implemented query expansion technique and ran BM25 with Pseudo Relevance Feedback.BM25, TF-IDF and Query Likelihood models are run with Stopping and Stemming techniques as a part of our project.

The performances of the above retrieval models are evaluated in terms of their retrieval effectiveness using the below mentioned 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 Statement of Contribution:

- **Nanditha Sundararajan :** Was responsible for implementing Task1. Was involved in implementing the Pseudo Relevance Feedback with KL-Divergence scoring and made a major contribution in documentation.
- **Poorva Sonparote:** Was responsible for implementing Task 2 ,Task 3and made major contribution in documentation.
- **Shruti Parpattedar**: Was responsible for implementing Phase 2 and Phase 3. Made major contribution in documenting changes.

2.Literature and Resources:

2.1 Overview:

The techniques used in the project are as follows:

- Phase 1: Indexing and Retrieval:
- Task1:
  - Indexer: The data structure for inverted index is a dictionary in which index term is the key and the value is a dictionary of the type (document_id : count)..The inverted indices are generated by the Indexer.py file.
  - TF-IDF: For TF-IDF we are calculating the 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.
  - BM25: The relevance judgments are taken into consideration by using 'cacm.rel' file and the values of 'k', 'k1', 'k2' are selected as per TREC standards.
  - 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 implemented along with Dice's Coefficient to rank the top 'k' retrieved documents.
- Task 3 : Stopping and stemming:

  Stopping: 'common-words.txt'is used to skip indexing any word, that belong to the document.
  Stemming:'cacm_stem' is used as corpus and 'cacm_stem.query' is used as list of stemmed queries.
- Phase 2 : Displaying Results:

  We have used Snippet generation and Query highlighting for displaying results. For snippet generation, we have used basic Luhn's algorithm with minor variation in choosing significant words to improve upon it.
- Phase 3 : Evaluation:

  Evaluation of retrieval effectiveness is finally done by using following measures: MAP, MRR, P@K and Precision & Recall.
- Relevance Model with Pseudo Relevance feedback and KL Divergence for scoring (**Extra Credit**):

  Developed a model using the Pseudo Relevance feedback and selected the top 7 terms using the Dice's coefficient scores for query expansion. The documents are scored using the KL Divergence algorithm for each of the expanded query. The results are sorted in descending order and stored in a text file.

2.2 Other Tools:

For developing our project, we have used the below third-party tools:

- Lucene 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 Resources:

For implementing the retrieval models and for Pseudo Relevance Feedback we referred:

- Search Engines Information Retrieval in Practice by W. Bruce Croft, Donald Metzler, Trevor Strohman.

For implementing BM25 model, we referred:

- https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model1.html

3. Implementation and Discussion:

3.1 Baseline Runs:

- 3.1.1 TF-IDF Model:
    - TF(t) is nothing but the number of occurrences of term in a document divided by total number of terms in that document
    - IDF(t): To attenuate the effect of terms that occur too often in the collection to be meaningful for relevance determination, we scale down the term weights of terms with high *collection frequency,* defined to be the total number of occurrences of a term in the collection. The idea would be to reduce the TF weight of a term by a factor that grows with its collection frequency
      IDF (t) = log_e (Total number of documents / Number of documents with term t in it).

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

- 3.1.2 Query Likelihood Model: The documents are ranked by the probability that the query text could be generated by the document language model. Smoothing is used to overcome data sparsity by lowering or discounting the probability estimates for the words that are seen in the document text and assign the leftover probability estimates for the words that are not present in the document. This is calculated by following formula:

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

- λ is 0.35 as per TREC standard
- f_{qi} is number of times word q occurs in document
- D is number of words in document
- c_{qi} 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

- 3.1.3 BM25 Model: BM25 is an extended/updated version of binary independence model that uses query term weights to rank the documents. We use the below formula to calculate the scores of the document.

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

where,

- k1 =1.2, k2= 100
- K: K = k1((1-b) + b*dl / avdl) where dl is the document length and avdl is the average length of document in the collection.
- The value of b is chosen as 0.75
- qf_i is the frequency of the term within query frequency
- f is the document frequency of the term
- n is the number of documents in the collection indexed by this term
- N is the total number of documents in the collection
- r is the number of relevant documents indexed by this term
- R is the total number of relevant documents

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

query_id Q0 doc_id rank TF-IDF_score BM25RelevanceModel

- 3.1.4 Lucene Model: Provides Java-based indexing and search technology along with well as spell checking, hit highlighting and advanced analysis/tokenization capabilities. It is an open source project and can be accessed here - https://lucene.apache.org/. Lucene scoring uses a combination of the Vector Space Model (VSM) of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query

Documents are sorted in descending order of their scores and top 100 documents are printed in the 'LuceneRun.txt' in the format:

query_id Q0 doc_id rank TF-IDF_score Lucene

3.2 Query Expansion:

The implementation of query expansion takes the following course:

- 3.2.1 Query Time Stemming:
  - The original 64 queries, given in cacm_stem.query.txt file is tokenized by running the file task2-QueryExpansion.py file.
  - To retrieve the synonyms and the derivative variants of the terms in the tokenized queries, we call the Words API by in the file task2-QueryExpansion.py. This results in the generation of expanded queries based on derivatives for every term in the query as returned by the API. The expanded queries are stored in new_queries.txt.

  - Two baseline runs (TFIDF and BM25) are performed on the reformulated queries again and the output is stored in the Outputs folder.

- 3.2.2 Pseudo Relevance Feedback:
  - The Dice's coefficient is calculated on the original 64 queries, given in cacm.query.txt file and the top seven terms with the highest association measure are added to the query.
  - BM25 baseline run is performed on the reformulated queries again and the output is stored in the Outputs folder.

  - This part of the project can be implemented using Task2-PseudoRelevance.py and the output is stored in the Outputs folder under the name PseudoRelevanceWithBM25.txt.

3.3 Query Analysis:

For query analysis, we have chosen the below queries and discussed the results.

3.3.1 "portabl oper system":

- The top most document CACM-3127 was relevant to the query. This document (CACM-3127) contains information about "Thoth, a Portable Real-Time Operating System". It has all the words in the given query, so it is topically relevant to the query. However, the ranked documents after CACM3127 are not purely relevant to the given stemmed query.
- Document CACM-2319 (talks about "Operating System Performance") is partially relevant. Similarly, document CACM-3068 talks about "A Model for Verification of Data Security in Operating Systems", CACM-2379 talks about "The Design of the Venus Operating Systems" etc. are partially relevant as they do contain the query terms "oper" and "systems", but documents like CACM-1591 talks about "A Model for a Multifunctional Teaching System", CACM-1680 talks about "A General-Purpose Display Processing and Tutorial System" and still exist in the ranking list.
- The reason for these documents occurring in this ranking list is there are words like "operations", "operate" etc. present in them and thus, increasing the relevance of a term in a query for document.

3.3.2 “code optim for space effici”:

- The top most document in the ranking list for BM25 Retrieval model for stemmed corpus is CACM-2748 which talks about “Indirect threaded code” which is not relevant to the query, topically. This document is not present in the ranking list of BM25 Retrieval Model for un-stemmed corpus.
- CACM-2491 talks about “Threaded code” and “space” and hence is relevant. Similarly, documents like CACM-2680, CACM-2863, CACM-3129, and other contain words like “coded”, “optimal”, “space” and hence exist in the ranking list but are not topically relevant. Despite not being relevant to the topic, these documents were retrieved.
- For tf-idf, the most relevant document is CACM-1064 which is at 102th position for BM25 relevance. The second most relevant document is CACM- 2748 which is the most relevant document for BM25

3.3.3 “parallel processor in inform retriev”:

- The top - rated document using BM25 is CACM-2264 which talks about “Derived Semantics for Some Programming Language Constructs”.
- The top document for TF-IDF is CACM-1367 which is “Character Structure and Character Parity Sense for Parallel-by-Bit Data Communication in ASCII” which only has the term “parallel” in common.

3.4 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 are:

- 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.

- This phase of the project can be implemented by running Phase2Run.py, which will internally call Indexer.py to parse and index the corpus and snippetGeneration.py.
- The output is displayed on the console with query terms highlighted.

3.4.1 Query Expansion using Pseudo Relevance Feedback with Stopping:

- Dice's coefficient is calculated on the original 64 queries, given in cacm.query.txt file and the top seven terms with the highest association measure are added to the query. Only those terms not already present in the query were added to the query.
- Next, the common words from common_words file are removed from the expanded query.
- BM25 baseline run is performed on the reformulated queries again and the output is stored in the Outputs folder as Phase3RunOutput.txt.

3.5 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.

3.6 :Relevance Model with Pseudo Relevance feedback and KL Divergence for scoring (**Extra Credit**):

- Developed a model using the Pseudo Relevance feedback and selected the top 7 terms using the Dice's coefficient scores for query expansion. The documents are scored using the KL Divergence algorithm for each of the expanded query. The results are sorted in descending order and stored in a text file.

$$\sum_{w \in V} \frac{f_{w,Q}}{|Q|} \log P(w|D)$$

4 Results:

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

Evaluation_output.xlsx

5. Conclusion:

By evaluating these models, we found that:
On the basis of the results, we can conclude that the BM25 retrieval model is better than the TF.IDF and Lucene. For baseline runs it was observed that BM25 performed the best for all the evaluation parameters.
It has a higher mean average precision(MAP), mean reciprocal rank(MRR), P@5 and P@20.
Therefore, it retrieved more relevant documents earlier and at higher ranks. After query expansion, we observed that TF-IDF and BM25 were like their respective base runs. QLM has higher MRR and MAP scores in most cases.

5 Bibliography:

5.1Books:

- 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.*

5.2Scholarly 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

5.3 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