Information Retrieval Project Report

CS 6200: Information Retrieval

Northeastern University

Fall 2017 (Semester 1), Prof. Nada Naji

<u>TEAM MEMBERS:</u>

Akshat Shukla

Parshva Shah

Virat Goradia

<u>ii. Introduction:</u>

The goal of this project is to design and build the information retrieval systems, evaluate and compare their performance in terms of retrieval effectiveness.
The project is divided into 3 phases as explained below:
(Each member's contribution is also mentioned alongside each task in brackets)

1) Phase 1: Indexing and Retrieval

It comprises of 3 Tasks as below:

1) <u>Task 1:</u> It comprises of 4 steps:
    1) <u>Step 1:</u> This step takes Raw HTML files (given corpus) as input, and returns same number of files but with each input file tokenized by case-folding and removing punctuations. (ViratG)
    2) <u>Step 2</u>: This step takes output of Step 1 as input and generates an inverted index and a document which maps Document ID and its length. (ViratG)
    3) <u>Step 3:</u> This step takes the given query file and cleans it to convert into a file containing a mapping between query id and corresponding query. (ParshvaS)
    4) <u>Step4:</u> This step constitutes implementing the following models for score calculation and finding top 100 documents for each retrieval models below:
        1. BM25 (Considering no relevance) (AkshatS)
        2. BM25 (Considering relevance) (ViratG)
        3. Lucene (ParshvaS)
        4. Query Likelihood (AkshatS)
        5. tf-idf (ParshvaS)
2) <u>Task 2:</u> In this task, Pseudo Relevance Model is used to re-compute scores calculated for BM25 Model which considered relevance. The output comprises of top 100 documents according to this model. (AkshatS & ViratG)
3) <u>Task 3:</u> This task comprises of 2 parts:
    1) <u>Part A:</u> This part consists of performing stopping on the corpus as well as queries (using common_words.txt) to perform scoring by performing 4 steps same as those in Task 1:
        1) <u>Step 1:</u> Tokenization (ParshvaS)
        2) <u>Step 2:</u> Inverted Index Generation (ParshvaS)
        3) <u>Step 3:</u> Query generation (AkshatS)
        4) <u>Step 4:</u> Score calculation and finding top 100 documents using all the 5 retrieval models. (AkshatS, ParshvaS & ViratG)

    2) <u>Part B:</u> This part is divided into 3 steps:
        1) <u>Step 1:</u> Tokenization of the given stemmed corpus (Using cacm_stem.txt) (ViratG)
        2) <u>Step 2:</u> Inverted List Creation (AkshatS)
        3) <u>Step 3:</u> Finding top 100 documents by calculation scores for each using using all the 5 retrieval models (AkshatS, ParshvaS & ViratG)

2) Phase 2: Displaying Results:

This phase consists of snippet generation for the top 5 documents found for each query for the models 1,3,4,5 (Stopped) (AkshatS & ParshvaS)

3) Phase 3: Evaluation:

This phase consists of evaluating the results found by calculating scores for each of the following retrieval models:

- BM25 (Considering Relevance)
- BM25 (Considering Relevance and Stopping)
- BM25 (Considering Relevance and PRF)
- Lucene
- Lucene (With Stopping)
- QLM
- tf-idf
- tf-idf (With Stopping)

It consists of 2 steps:

1) Step 1: This step consists of forming encoded dictionaries which are used by the Step 2 for evaluating the results generated by the above-mentioned models. (AkshatS & ViratG)
2) Step 2: This step consists of evaluating the results by calculating different evaluation measures for each result set like precision-recall tables, MAP, MRR, P@K.
(ParshvaS & ViratG)

NOTE: Used term-at-a-time evaluation for better efficiency. (CMS Page 168)

4) Extra-credit:

This phase consists of computing document scores considering the positions of two terms (proximity) in the same order. It consists of two parts:

1) Part A: No Stopping (AkshatS, ParshvaS & ViratG)
2) Part B: With Stopping (AkshatS, ParshvaS & ViratG)

iii) Literature and Resources:

Following approaches were used while performing each of the below mentioned tasks:

1) Phase 1 Task 1 Step 1:
While de-punctuating, occurrences of ‘-’, ‘:’, ‘,’ or ‘.’ within digits are retained and occurrence of ‘$’ before a number is retained

2) Phase 1 Task 1 Step 3:
For extracting queries from the given cacm_queries.txt, <ROOT> tag is surrounded around whole file and <QUERY> tag is surrounded around each of the 64 queries, followed by converting it into .xml and extracting query IDs and their corresponding queries.

3) Phase 1 Task 1 Step 4:

- Two versions of BM25 are generated, one with assumption of no relevance consideration, while other with relevance considered. Reason for keeping BM25 without relevance is that it is used in Phase 2 (Snippet Generation) for generating snippets for all 64 queries (BM25 with relevance would have given snippets only for 52 queries whose relevance information is given according to cacm.rel.txt). The values for 'K', 'k1', 'k2' are chosen as per TREC standards. Formula used:

  ((k2+1)q)/((k2+q))*((k1+1)f)/((K+f))*log((r+0.5)(N-n-R+r+0.5))/((n-r+0.5)*(R-r+0.5))
  where;
  r = no. of relevant documents containing the query term
  R = no. of relevant documents for that query
  N = total no. of documents in the collection
  q = frequency of the query term in the query
  K = k1(b*L+(1-b))
  f = term frequency in that document
  n = total number of documents in which the term appears
  L = doc length / average doc length

- The Lucene model uses Standard Analyzer for tokenizing and analyzing text.

- The smoothing parameter (λ) is set to 0.35 for computing scores using Query Likelihood Model. Formula used:
  score=(1- λ)*(tf/D)+(λ*(tf/C))
  where:
  λ's value set to 0.35 as per TREC standards
  D: is the document length
  C: corpus length
  tf: term frequency

- Normalization is performed to the scores computed by tf-idf retrieval model by:
  1) normalized_tf=tf/D
  2) idf=1+math.log(N/df+1);
     document score=normalized_tf*idf

  where: tf: term frequency, df: document frequency, N: total number of documents in the corpus, D: document length
  NOTE: We add 1 to df to prevent divide by 0 error and add 1 to the log value to prevent the idf value to become 0, if log value is 0.

4) Phase 1 Task 2:
In pseudo-relevance feedback model, words occurring frequently in top documents are added in query to perform query expansion. (CMS Page 208).

5) Phase 2:
   While dividing each of the top 5 documents' contents into sentences, an assumption is made that each sentence consists of a maximum of 10 tokens. The algorithms for snippet generation include a Luhn's Approach of calculating significant factor to select top sentences for summary (CMS Page 216). To find whether a word is significant or not, a modified version of frequency-based criterion has been used (CMS Page 216, 217) as below:
   A word is significant iff

f(d,w)>=4-0.1*(25-sd)	if sd<25 or
f(d,w)>=4	if 25<=sd<=40 or
f(d,w)>=4+0.1*(sd-40)	otherwise

   where,
   f(d,w) is the frequency of word w in document d and w is not a stop-word
   sd is the number of sentences in document d.

   This algorithm is used to calculate significance factor of all the sentences and top 3 sentences are chosen to be included in snippet.

6) Phase 3: For a uniform test for all runs, only the queries with relevance information given have been used, rest have been excluded i.e. 52 queries have been evaluated for each model in this case.

OTHER TOOLS USED:

1) Beautiful SOUP: Used for parsing the documents in the given corpus.
2) Lucene libraries: lucene-core-4.7.2.jar
   lucene-queryparser-4.7.2.jar
   lucene-analyzers-common-4.7.2.jar

iv) Implementation and Discussion:

Thorough descriptions for tasks are as below:

1) Phase 1 Task 2:
   First, consider top "k" words from each of the top "n" documents. Consider these factors (k, n) each to be 5. Thus, 5*5=25 words are to be added to each given query, thereby performing query expansion. Since the top 5 words from each document can be a "stop word", do not include the stop words as per 'common_words.txt'. Also, remove the words that are present in this expansion term list, which is already present in the original query. Run the bm25 (with relevance) model for these expanded queries.

2) <u>Extra Credit:</u>

   a) Generate an inverted index dictionary with term as key and (document_id, position_list) as it's corresponding value.

   <u>NOTE:</u> position_list denotes the list of all positions of that term in that document, denoted by its document_id.

   b) Generate bigram terms for the query. Split every bigram term and store it in an array having first bigram term as the first element and second bigram term as the second element. From the inverted index dictionary, get the value part of the first bigram term. This value part will have the documents you want, it will be a tuple (document_name, position_list). Get this second part of tuple, which is the position list.

   c) Repeat step (b) for the second bigram term too.

   d) For the position_list of the first bigram term, subtract each of its element by each element of position_list of the second bigram term. For each such pair having a difference in the range [4,0) which ensures no more than 3 words between these two, score will be calculated as:
   Score = difference + 5 (More score for closer words)
   and added to the total document score.

   e) Documents are ordered in decreasing order of scores for each query and top 100 are shown.

<u>**Query by Query Analysis:**</u>

Three queries we chose are:

1) portabl oper system (Stemmed Version)
   portable operating systems (Non-Stemmed Version)
2) parallel algorithm (Stemmed Version)
   Parallel algorithms (Non-Stemmed Version)
3) appli stochast process (Stemmed Version)
   Applied stochastic processes (Non-Stemmed Version)

In the first query, the query terms 'portable', 'operating' and 'systems' are stemmed to 'potabl', 'oper' and 'system' respectively. In the second query, only the term 'algorithms' is stemmed to 'algorithm'. The term 'Parallel' is not stemmed. In the third query, the query terms 'Applied', 'stochastic' and 'processes' are stemmed to 'appli', 'stochast' and 'process' respectively.

In case of stemming, all the words that belong to the same stem class get stemmed to that one stem root. Eg., in the first query, the words 'operation', 'operable', 'operating','operand',etc. all get stemmed to single stem 'oper'. Similarly, in the third query, the words 'application','applied','applying','applicable',etc get stemmed to single stem 'appli'. Hence, documents having words stemming down to a single word would score lesser, since previously, individual unique terms would be considered and now only a single stemmed word for all those stem class words would be considered.

If we consider the first query, only six documents match when we compare the top 20 documents obtained by the bm25 model with stemming and of that without stemming, namely, 'CACM-3127', 'CACM-2541','CACM-2246','CACM-3068','CACM-2740' and 'CACM-1750'. Reason for lesser matches is that stemming would have had a high impact on the query terms.

If we do the same analysis for the second query between the same two versions of the aforementioned model, 14 documents match, namely 'CACM-2714', 'CACM-2973', 'CACM-0950', 'CACM-2433', 'CACM-2785', 'CACM-2266', 'CACM-1262', 'CACM-2700', 'CACM-2685', 'CACM-3156', 'CACM-1158', 'CACM-3075', 'CACM-1828' and 'CACM-2289'. Reason for higher number of matches is that stemming of just one query term 'algorithms' to 'algorithm' has a very small impact on the query terms.

If we do the same analysis for the second query between the same two versions of the aforementioned model, 7 documents match, namely,'CACM-1696', 'CACM-0268', 'CACM-1410', 'CACM-2882', 'CACM-1540', 'CACM-1194' and 'CACM-3120'. Reason for lesser matches is that stemming would have had a high impact on the query terms.

<u>NOTE:</u> A relatively different query-by-query analysis is placed in "Query-By-Query-Analysis.txt" which contains the analysis for top 5 documents obtained by the three baseline runs- Lucene;s retrieval model, BM25(with relevance) retrieval model and tf-idf retrieval model. We talk about the drops in scores between the ranks, common documents obtained by the three aforementioned retrieval models, etc. and provide a generalized speculation, per query, for three queries we found interesting.

v) Results:

Final results can be found as the following text files:

1) Phase 1/Task 1/Step 4/BM25/BM25_Relevance_Top100_Pages.txt
2) Phase 1/Task 1/Step 4/Lucene/Lucene_Top100_Pages.txt
3) Phase 1/Task 1/Step 4/QLM/QLM _Top100_Pages.txt
4) Phase 1/Task 1/Step 4/TF-IDF/TF_IDF_Normalized_Top100_Pages.txt
5) Phase 1/Task 2/Step 4/BM25_Relevance_PRF_Top100_Pages.txt
6) Phase 1/Task 3/Part A/Step 4/BM25 (Stopped)/Stopped_BM25_Relevance_Top100_Pages.txt
7) Phase 1/Task 3/Part A/Step 4/Lucene (Stopped)/Stopped_Lucene_Top100_Pages.txt
8) Phase 1/Task 3/Part A/Step 4/TF-IDF (Stopped) /Stopped_TF_IDF_Normalized_Top100_Pages.txt

vi. Conclusion and outlook:

Retrieval Model	**Mean Average Precision**	**Mean Reciprocal Rank**
BM25 (With Relevance)	0.546	0.804
BM25 (With Relevance With PRF)	0.540	0.800
Lucene	0.421	0.703
TF-IDF	0.145	0.244
Query Likelihood	0.101	0.133
BM25 (With Relevance and Stopping)	0.554	0.824
Lucene (with stopping)	0.212	0.332
TF-IDF (With Stopping)	0.348	0.587

NOTE: PRF stands for Pseudo Relevance Feedback

- After analyzing the evaluation of top 100 documents of each query by all the models, a conclusion can be made that for the given combination of corpus and queries, BM25 with relevance considered and with stopping, gave the best results (Mean Average Precision of 0.554 and Mean Reciprocal Rank of 0.824)
- As BM25 with Relevance and Stopping harnessed the relevance information provided in cacm.rel.txt unlike other models, it tends to show better results for this test collection. Also, it also used stopping to improve its score.
- If we were provided with any user query logs information (such as session history, etc.), the query refinement technique which involved query expansion could use query logs as its source for better results for expanded queries than using pseudo relevance feedback. PRF is only as we have no query log information which is the best source for a knowing effective context of the query.

vii. Bibliography:

- Search Engines: Information Retrieval in Practice by Croft, Metzler, Strohman
  (For concepts and logic behind implementations)
- www.stackoverflow.com
  (For syntactical assistance)