***Information Retrieval***

CS 6200: Information Retrieval

Northeastern University

Spring 2018, Prof. Nada Naji

***TEAM MEMBERS:***

*Kevin Shah*

*Manoj Venkatesan*

*Twisha Vyas*

**ii. Introduction:**

The main aim of this project is to design and implement an information retrieval system using models such as BM25, Lucene, Query Likelihood and TF-IDF and evaluate their effectiveness.

The project is divided into 4 parts including extra credit as explained below:

(The contribution of each team member is written beside the steps)

1. **Phase 1: Indexing and Retrieval**

It comprises of 3 Tasks as below:

1. Task 1: It includes 4 steps:
2. Step 1-Corpus Generation: 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. (Kevin)
3. Step 2-Index Generation: This step takes the clean documents created in step 1 as input and generates an inverted index of the form Term 🡪[(Document\_id, Term Frequency)]. (Kevin)
4. Step 3-Query Cleaning: The queries provided are processed in the same way as documents and the cleaned queries are written into a file. (Twisha)
5. Step4-Retrieval Models: There are 4 parts in this step. 4 retrieval models are implemented using the clean queries and index generated in steps 3 and 2. They are as follows:
6. BM25:

BM25 considering no relevance (Kevin)

BM25 Considering relevance (Kevin)

1. Lucene (Kevin)
2. Smoothed Query Likelihood (Twisha)
3. TF-IDF (Manoj)
4. Task 2: We have used pseudo relevance feedback for query enrichment. Pseudo Relevance feedback is used to re-compute scores calculated for BM25 Model which considered no relevance. The output of this task contains top 100 documents after expanding the query with most frequent (Kevin)
5. Task 3: This task comprises of 2 parts:
6. Part A: In this part, a new corpus is created by removing the stop words provided. The stop words are also removed from the queries.
7. Step 1: Corpus Generation (Manoj)
8. Step 2: Index Generation (Manoj)
9. Step 3: Query Cleaning (Manoj)
10. Step 4: Retrieval Models. (Manoj)
11. Part B: In this part, Stemming is performed on the terms in the corpus as well as the queries.
12. Step 1: Corpus Generation (Manoj)
13. Step 2: Index Generation (Manoj)
14. Step 3: Query Cleaning (Manoj)
15. Step 4: Retrieval Models. (Manoj)
16. **Phase 2: Displaying Results:**

This part of the project consists of snippet generation and query term highlighting. (Twisha)

1. **Phase 3: Evaluation:**

(Kevin)

In this phase, we have performed the following operations:

1. MAP

2. MRR

3. P@K, K = 5 and 20

4. Precision & Recall

For the following models:

Decide the model on which you want to perform evaluation:

1. Baseline BM25

2. Baseline Lucene

3. Baseline Smoothed Query Likelihood

4. Baseline TF-IDF

5. BM25 Pseudo-relevance Feedback

6. Stopped BM25

7. Stopped Smoothed Query Likelihood

8. Stopped Lucene

1. **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

………………………………………

Step 3- Query cleaning:

* Remove the tags <doc> and <doc\_no> from cacm.query.text. –
* We then break the query into a dictionary with the mapping Query\_id(obtained form <doc\_no>) 🡪 Query(from <doc>).
* Perform casefolding on the terms in the query
* Remove punctuations and extra spaces from the query except ‘-‘. Keep ‘.’ between digits.
* Write the clean queries to a file

Step 4- Retrieval Models:

BM25:

We have code for both, BM25 using Relevance(using cacm.rel.txt, we got the relevant documents for each query and plugged in values for R and r using the same) and without using relevance.

We are using the formula from the book *‘Search Engines: Information Retrieval in Practice’*[1].

((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

Lucene:

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

Smoothed Query Likelihood Model:

The smoothing parameter (λ) is set to 0.35 for computing scores using Query Likelihood Model. We have used the following formula from *‘Search Engines: Information Retrieval in Practice’*[1].

score=(1- λ)\*(tf/D)+(λ\*(cf/C))

where:

λ’s value set to 0.35 as per TREC standards

D: is the document length

C: corpus length

tf: term frequency in the document

cf: frequency of the term in the corpus

TF-IDF

We have used the following formula for calculating scores for documents using TF-IDF:

……………………………

We have performed normalization to the scores computed by tf-idf retrieval model using the following steps:

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

Task 2:

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

……………………………………………………

1. 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.

1. 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.

1. Extra Credit:
2. Generate an inverted index dictionary with term as key and

(document\_id, position\_list) as it’s corresponding value.

NOTE: position\_list denotes the list of all positions of that term in that document, denoted by its document\_id.

1. 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.
2. Repeat step (b) for the second bigram term too.
3. 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.

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

**Query by Query Analysis:**

Three queries we chose are:

1. portabl oper system (Stemmed Version)

portable operating systems (Non-Stemmed Version)

1. parallel algorithm (Stemmed Version)

Parallel algorithms (Non-Stemmed Version)

1. 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.

NOTE: 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 (Without Relevance) | 0.435550 | 0.695191 |
| BM25 (Without Relevance With PRF) | 0.396661 | 0.565076 |
| Lucene | 0.412352 | 0.701790 |
| TF-IDF | 0.258652 | 0.449084 |
| Smoothed Query Likelihood | 0.382881 | 0.649601 |
| BM25 (Without Stopping) | 0.474866 | 0.695006 |
| Lucene (With stopping) | 0.433673 | 0.734005 |
| Smoothed Query Likelihood (With Stopping) | 0.426498 | 0.737010 |

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 and References:**

[1] Search Engines: Information Retrieval in Practice by Croft, Metzler, Strohman

(For concepts and logic behind implementations)