Search Engine for CACM Collection

CS 6200: Information Retrieval

ASKSHAY SINGH, CHAITANYA KAUL, SANJAY MURALI

Under Guidance of Professor Nada Naji

Fall 2017

I. Introduction

This project is designed to build a Search Engine for the CACM corpus. The goal is to design several retrieval systems using certain Indexing techniques and finally evaluate all of them to understand which one produces the best result.
The process involves, first building a unigram inverted index out of the pre-processed corpus. The query file is also processed at the same time and then along with the corpus, a top-100 ranking list is generated for each of the queries.
Finally, the result of all the retrieval systems is analyzed in the “Evaluation Stage”, helping us understand the efficiency/performance of each of the retrieval systems.

II. Contributions

Sanjay Murali

1. Implemented Baseline runs on BM25, TF-IDF and Lucene retrieval models.
2. Implemented proximity based search (Extra Credit) without stopping.
3. Developed Evaluation (Phase-3) for generating Precision, Recall, MAP and MRR

Chaitanya Kaul

1. Design and Implement Smoothed Query retrieval model based on Jelinek-mercer algorithm
2. Implemented Pseudo Relevance Feedback for BM25
3. Implemented Snippet Generation and query term highlighting Phase-2

Akshay Singh

1. Responsible for generating new corpus after removing the stopped words given in the “common_words.txt” file and generating Stemmed corpus from the given document “cacm_stem.txt”.
2. Performed 6 runs required for the completion of Task 3.
3. Convert the results for all runs in the project to Spreadsheets for easy access.

III. Literature and Resources

3.1 Literature

BM25 Algorithm

BM25 Retrieval algorithm is one of the most popular[1] retrieval algorithm which is used as a basis for a lot of popular search engines till date. It extends the binary independence model to include document and query term weights[1]. The values of parameters k1 = 1.2, b= 0.75 and k2 = 100 are set for maximum effectiveness as found in TREC experiments.

TF-IDF Algorithm

TF-IDF or simply, term frequency-inverse document frequency is yet another type of retrieval algorithms which is simply based on term and query frequency weights[1]. Our one of the baseline runs includes the implementation of this algorithm.

Jelinek-Mercer Algorithm

Jelinek-Mercer method is used for implementing Smoothed Query Likelihood model. As given in the document we have implemented the model with λ = 0.35.

Rocchio’s Algorithm

Rocchio's algorithm is an algorithm[1] which is used as part of performing pseudo-relevance feedback. Pseudo relevance feedback basically involves the user selecting a set of documents which are relevant and another set of documents which are irrelevant to a search query. This process improves the results obtained for the targeted queries.

Snippet Generation

Snippet generation is one of the techniques through a summary of the document can be created. Snippets basically show the part of documents which are more relevant to the query. We have used Luhn's algorithm[1] for calculating the significant factor for snippets in a document.

$$Significant\ factor = \frac{number\ of\ significant\ words^2}{Bracket\ size}$$

3.2 Resources:

Lucene

Lucene[2] is part of the Apache project which basically gives us three jar file to perform functionalities like indexing, parsing and retrieval of documents based on given queries. It is used as one of the baseline runs for our project. Jar files:

1. lucene-core-VERSION.jar
2. lucene-queryparser-VERSION.jar
3. lucene-analyzers-common-VERSION.jar.

Beautiful Soup

BeautifulSoup[3] is one of the very popular libraries used for reading, parsing and extracting contents out of HTML documents by treating them as XML related documents. BeautifulSoup is used in our project for extracting and parsing the corpus

IV. Implementation and Discussion

Phase 1 Task 1

Indexing and Query Cleaning

The task 1 in phase 1 involves cleaning of the corpus, meaning tokenizing all the corpus documents, converting the case from upper to lowercase wherever needed and generating text(.txt) based files for each of the documents using BeautifulSoup. These input files are then used for generating a unigram based inverted index. The script which performs all these operations for the corpus is *gen-cacm-corpus.py*.
The queries, at the same are also parsed and extracted from the HTML style syntax and dumped to a single text file. For queries also, tokenization is performed. The script which performs these operations is *queryBreakdown.py*.

BM25 Ranking Algorithm

BM25 retrieval algorithm, a very popular form of retrieval model extends the scoring function[1] for the binary independence model to include document and query weights. BM25 in a lot of experiments has performed really well in a lot of experiments carried out by TREC. The formula for calculating BM25 is as followed:

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

The value for k_1, k_2 and K is set empirically and it's found that the typical values are for k_1=1.2 and k_2 = 100[1]. The value for K is calculated using the below formula

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

where b indicates the amount of normalization performed, i.e 1 being full normalization and 0 being no normalization. The script BM25.py performs the calculation of BM25.

TF-IDF Retrieval Algorithm

TF-IDF retrieval algorithm is one of the most straightforward algorithms. tf indicates the importance of a term in document Di and idf reflects the importance of term in a collection of documents[1]. The formula for calculating tf component is as followed:

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

Similarly, we can also define the formula for idf component as followed:

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

Here, N indicates total number of documents N and n_k indicates the documents containing the term k. The script *tf-idf.py* performs the computation of tf-idf scores for us.

Smoothed Query Likelihood Model

The Smoothed Query Likelihood Model is implemented using the Jelinek – Mercer Method given below:

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

Here,
$f_{qi,D}$ = The number of times word qi occurs in the document D.
|D| = The number of words in D.
C_{qi} = The number of times a query word occurs in the collection of documents
|C| = The total number of words occurrences in the collection.

Lucene Retrieval Algorithm

For Lucene, we were supplied with three jar files containing the core modules as well as the default retrieval models. For our project, we focused on SimpleAnalyzer[2] which parsed the query document and accordingly built the inverted index out of the corpus before finally calculating the retrieval scores.

Phase 1 Task 2

Pseudo Relevance Feedback

For performing query expansion, we selected BM25. Next, we implemented a combination of Pseudo-Relevance feedback and Rocchio's[1] Algorithm to expand the size of the query. The formula for Rocchio's algorithm can be defined as below[1]:

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

In the above formula, we used the value of α=8, β=16, γ=4[1]. We also considered that the top 10 ranked documents are relevant and the other 90 documents were non-relevant for our calculation of new query vector q_j. The decision was based on the fact that if we increased the number of relevant documents, we actually, added more terms to the query which resulted in bad results.

Phase 1 Task 3

Stopping Without Stemming on baseline runs

For Stopping without stemming, we used the *common_words.txt* to remove the common stop words like a, an, the etc. from our corpus. We then ran BM25[1], Jelinek-Mercer[1] and TF-IDF[1] on this processed corpus.

Stemming on baseline runs

For this process, we are given with a stemmed corpus and a stemmed query list. The corpus is tokenized and the new inverted index is processed and fed to the BM25 ranking algorithm which gives us the top ranked documents for the given queries.

Query-by-Query analysis

For the task 3, where stemmed queries are used for running baseline runs for models BM25, Smoothed Query Likelihood and tf-idf on the given stemmed corpus.
We consider below three queries for analysis:

1. STEMMED: portabl oper system, ORIGINAL: portable operating systems
2. STEMMED: parallel algorithm, ORIGINAL: parallel algorithms
3. STEMMED: perform evalu and model of comput system, ORIGINAL: performance evaluation and modelling of computer systems

For the first query, it is observed that all the terms in original query have been transformed in the stemmed version of the query i.e., 'portable' ➔ 'portabl', 'operating' ➔ 'oper' and 'systems' ➔ 'system'. In case of the second query, there is only one difference between original and stemmed version i.e., 'algorithms' ➔ 'algorithm'. For the third query, there are 5 differences between the original and stemmed version i.e., 'performance' ➔ 'perform', 'evaluation' ➔ 'evalu', 'modelling' ➔ 'model', 'computer' ➔ 'comput' and 'systems' ➔ 'system'

For the stemmed run, the stemmed corpus is generated, which includes same stem for all the words which belong to same stem class. For instance, "portable, portably, portables" will be replaced by word "portabl" in the stemmed corpus. Similarly, "opera, operabilities, operability, operable, operation, operating etc." will be replaced by word "oper" in the stemmed corpus. Thus, the documents having more words of the same stem class, will be more relevant to stem word present in the stemmed query.
For the original run, the relevance of the document will be measured by the original word in the query and the same word in the document, for instance, the words "Portable" and "Portably" will be treated differently.

Comparison of Results for Original Queries:

Original Queries (On Stopped Corpus)	**Results**
portable operating systems (Query number = 12)	**BM25** CACM-3127.txt CACM-2246.txt CACM-3068.txt CACM-1930.txt CACM-1462.txt **Smoothed Query Likelihood** CACM-3127.txt CACM-1461.txt CACM-3068.txt CACM-1462.txt CACM-2111.txt **TF-IDF** CACM-3127.txt CACM-1461.txt CACM-3068.txt CACM-2246.txt CACM-2319.txt
parallel algorithms (Query number = 19)	**BM25** CACM-2973.txt CACM-3075.txt CACM-0950.txt CACM-1601.txt CACM-2266.txt **Smoothed Query Likelihood** CACM-2973.txt CACM-3075.txt CACM-0950.txt CACM-2266.txt CACM-1601.txt **TF-IDF** CACM-2973.txt CACM-1262.txt CACM-2714.txt CACM-0371.txt CACM-0141.txt
performance evaluation and modelling of computer systems (Query number = 25)	**BM25** CACM-2318.txt CACM-1938.txt CACM-2319.txt CACM-3089.txt CACM-3119.txt **Smoothed Query Likelihood** CACM-2318.txt CACM-2319.txt CACM-2268.txt CACM-1938.txt CACM-2812.txt

	TF-IDF CACM-2318.txt CACM-1653.txt CACM-2984.txt CACM-1938.txt CACM-1374.txt

In the table above, the results for the BM25 model are as expected because of the relevance information given in the file “cacm.rel.txt”.
For query 12, 19 and 25 the top 5 results include the documents mentioned in the “cacm.rel.txt” file.
For query 12 i.e., “portable operating systems” the top 5 documents include CACM-3127 and CACM-2246 which is expected given their relevance information in the “cacm.rel.txt” file for query 12.
For other two models i.e., Smoothed Query Likelihood and tf-idf the top 5 results have common documents indicating the correctness of the algorithm, the results are different from BM25 indicating lack of relevance information, which was provided to BM25.

Comparison of Results for Stemmed Queries:

Stemmed Queries (On Stemmed Corpus)	**Results**
portabl oper system (Query number = 1)	**BM25** CACM-3127.txt CACM-3068.txt CACM-2319.txt CACM-2379.txt CACM-1591.txt **Smoothed Query Likelihood** CACM-3127.txt CACM-2246.txt CACM-3196.txt CACM-1461.txt CACM-2593.txt **TF-IDF** CACM-3127.txt CACM-1461.txt CACM-2246.txt CACM-3068.txt CACM-2796.txt
parallel algorithm (Query number = 3)	**BM25** CACM-2664.txt CACM-2685.txt CACM-1262.txt CACM-2700.txt CACM-1828.txt **Smoothed Query Likelihood** CACM-2714.txt CACM-2973.txt CACM-2266.txt CACM-3075.txt CACM-3156.txt

	TF-IDF CACM-2664.txt CACM-0141.txt CACM-1302.txt CACM-0392.txt CACM-2685.txt
perform evalu and model of comput system (Query number = 6)	**BM25** CACM-2318.txt CACM-3070.txt CACM-3048.txt CACM-2988.txt CACM-3147.txt **Smoothed Query Likelihood** CACM-2318.txt CACM-3070.txt CACM-3048.txt CACM-2741.txt CACM-2319.txt **TF-IDF** CACM-2318.txt CACM-2984.txt CACM-3070.txt CACM-1653.txt CACM-3089.txt

For the stemmed corpus, stemmed queries 1,3 and 6 were run on the models BM25, Smoothed Query Likelihood and tf-idf.
The effect of the stemming was observed with varying degrees on the queries, for instance, in query 1, which is the stemmed version of original query 12, only one relevant document i.e. CACM-3127 is present.
There is similarity observed between the top 5 results of BM25 model (with relevance information) and other models, with the stemming because the relevance information effect has been decreased, since all the words belonging to same stem class were reduced to same stem. This behavior is opposite to the observation of the original corpus (with stopping) where relevance information evidently played a major role.

Phase 2 Evaluation

Implementation

For Evaluation, we used recall, precision, mean average precision(MAP), mean reciprocal rank(MRR) as parameters for evaluating our retrieval models. We used scores from every baseline runs from their respective score directories.
We then calculate precision, rank, reciprocal rank for each query ID and store the results for each of the baseline runs. The scores are calculated in file *Query-Evaluation.py*
After that finally, by using the above results, we calculate average precision for each query and then we averaged the results over all the queries given to us.
Finally, we also calculate P@K for each model. The given value of K is 5 and 20.

Phase 4 Results

Task 1: Baseline_run_BM25_result	IR_Project\Task1\BM25\BM_25_RESULTS.xls

Task 1: Baseline_run_Lucene_result	IR_Project\Task 1\Lucene\ Lucene_Scores_RESULTS.xls
Task 1: Baseline_run_Smoothed_Query_ Likelihood _result	IR_Project\Task 1\ Smoothed Query Likelihood Model\ Jelinker_Scores_RESULTS.xls
Task 1: Baseline_run_ tf-idf _result	IR_Project\Task 1\ tf-idf\ TF-IDF_Scores_RESULTS.xls
Task 2: Psedo_relevance_Feedback_result	IR_Project\Task 2\BM25\ BM_25_PSEUDO_RESULTS.xls
Task 3: Stemming_BM25	IR_Project\Task 3\Three_baseline_runs_for_Stemming\BM25\ BM_25_STEMMING_RESULTS.xls
Task 3: Stemming_Smoothed_Query_Likelihood_Model	IR_Project\Task 3\Three_baseline_runs_for_Stemming\Smoothed Query Likelihood Model\ Jelinker_Scores_STEMMED_RESULTS.xls
Task 3: Stemming _tf-idf	IR_Project\Task 3\Three_baseline_runs_for_Stemming\tf-idf \TF-IDF_Scores_STEMMING_RESULTS.xls
Task 3: Stopping_BM25	IR_Project\Task 3\Three_baseline_runs_for_Stopping\BM25\ BM_25_STOPPING_RESULTS.xls
Task 3: Stopping _Smoothed_Query_Likelihood_Model	IR_Project\Task 3\Three_baseline_runs_for_Stopping\Smoothed Query Likelihood Model\ Jelinker_Scores_STOPPING_RESULTS.xls
Task 3: Stopping_tf-idf	IR_Project\Task 3\Three_baseline_runs_for_Stopping\tf-idf\ TF-IDF_Scores_STOPPING_RESULTS.xls
Phase 2: Snippet Generation and Query term highlighting	IR_Project\Phase-2\Snippets
Phase 3: MAP, MRR, P @ K, Precision and Recall	IR_Project\Evaluation
Extra Credits	IR_Project\Extra Credits\BM25 and IR_Project\Extra Credit\Evaluation

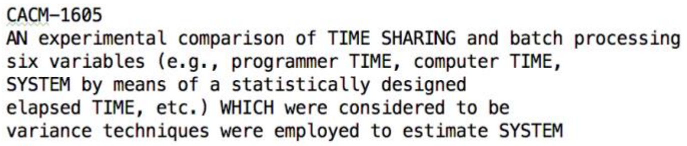
Reading the output for Runs:

Query_id   Q0 doc_id Rank score systemname

Reading output for Evaluation:

Query_id   Q0 doc_id Rank score systemname Relevant/Non-Relevant Precision_score Recall_score

Reading output for Snippet and Query Term Highlighting:



```
CACM-1605
AN experimental comparison of TIME SHARING and batch processing
six variables (e.g., programmer TIME, computer TIME,
SYSTEM by means of a statistically designed
elapsed TIME, etc.) WHICH were considered to be
variance techniques were employed to estimate SYSTEM
```

CACM-1605 => Document Name
Each line is a snippet
The Upper-case words like “TIME”, “SHARING”, “AN” is query term highlighting

Extra Credits

The extra credits deal with retrieval with proximity-enabled. We have used the techniques discussed in "Term Proximity Scoring for Keyword-Based Retrieval Systems" [4]. The BM25 retrieval is used. Process of calculating document score:

1. For each document, the BM25 score is calculated for a query
2. The terms in the query are taken in order and made into a set and a permutation of keywords are formed. For ex: The query "Information Retrieval is" would become [("Information", "Retrieval"), ("Information", "is"), ("Retrieval", "is")]
3. Term pair instance tpi(t_i, t_j) [4] is calculated for each term pair in the query is calculated such that the maximum distance between two terms can't be more than 3.
4. In a typical corpus, there would be instances of term pairs occurring several times in the document this is taken into account[4] as follows

$$w_d(t_i,t_j) = (k_1+1)\cdot\frac{\sum_{occ(t_i,t_j)} tpi(t_i,t_j)}{K+\sum_{occ(t_i,t_j)} tpi(t_i,t_j)}$$

5. For a given query the contribution of a term pair is calculated as follows, known as TPRSV[4]

$$TPRSV(d,q) = \sum_{(t_i,t_j)\in S} w_d(t_i,t_j)\cdot\min(qw_i,qw_j)$$

6. The new document score is calculated by adding the above calculated TPSRV value for that document with BM25 score and the Top 100 documents is shown to the user.

It should be noted that by applying the above method, the BM25 scores won't have any significant changes on the documents retrieved i.e., the top 100 documents retrieved would still be the same only their ranks be different based on the number of query terms which are close to each other in these documents.

Comparison Result and Effectiveness for Proximity Enabled Searching

The MAP for BM25 with Proximity is 0.53 which is lesser than BM25 without proximity which has a MAP of 0.54, this decrease in MAP can be attributed to the fact that in relevant documents the number of term pairs in proximity might be lesser than non-relevant documents in top 100, so the ranks of these non-relevant documents increases which in turn decreases the overall precision. The MRR remains the same for the BM25 with and without proximity enabled.
Running the BM25 with proximity-enabled on corpus with Stop words removed increases the MAP to 0.55, thus two important terms in the query would be much more nearer to each other than they were before hence you can see the increase in MAP. The MRR remains the same.

Let's take an example of query 1 in BM25 with and without proximity enabled, if you see the top 10 documents from CACM-1605 to CACM-2621 both the results, most of the documents are ranked the same except that their scores are different, there is one difference document CACM-1523 appears in 8th rank in BM25 without proximity while it appears in 7th rank in proximity-enabled BM25. This is because of the fact that more query term appears in close proximity to each other in document CACM-1523.

V. Conclusion and Outlook

Conclusion

Throughout all the baseline runs, we found that BM25 works out to be the best retrieval model as compared to other retrieval methods like TF-IDF, Smoothed Query Likelihood model and Lucene. It is also evident by the fact that MAP or Mean Average Precision for BM25 is much higher than any of the models.

It was also found out that the MAP and MRR for BM25 is not significantly different than BM25 with query expansion using Rocchio's algorithm for Pseudo Relevance feedback to include topical relevance. This can be because of the fact the Pseudo Relevance takes in consideration two vector sets, first of which is the relevant documents and other being the non-relevant documents which may or may not benefit the rankings when applied to the given queries.
TF-IDF has the lowest scores for baseline runs, a MAP of 0.31, Smoothed query model is a slightly higher than that at 0.35. Lucene produces much better scores of MAP and MRR at 0.41 and 0.70, which is still less than BM25 baseline, but better than other retrieval algorithms.
It can be noted that Jelinek-Mercer (Smoothed Query Likelihood model) produces much better MAP and MRR at 0.40 and 0.66 when stop words are removed in the corpus.
The proximity-enabled BM25 (Extra Credits) which is run with and without Stopping, produces almost same results as baseline BM25, this is because only few of the document's ranking changes which doesn't produce a significant change in MAP and MRR

Outlook

Different machine learning based approaches can be applied to retrieval models like BM25 which can improve the effectiveness of the runs. We can also provide the user with a User Interface through which a better visual guideline can be presented and also it can be used to help him/her select the relevant and the non-relevant documents from the set.
We can also include a methodology to apply spell check to the user defined queries so that better results can be generated.

Bibliography

[1] Croft, W. Bruce, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*. Vol. 283. Reading: Addison-Wesley, 2010.

[2] Lucene Version 4.7.2 documentation http://lucene.apache.org/core/4_7_2/

[3] BeautifulSoup, https://www.crummy.com/software/BeautifulSoup/

[4] Rasolofo Y., Savoy J. (2003) Term Proximity Scoring for Keyword-Based Retrieval Systems. In: Sebastiani F. (eds) Advances in Information Retrieval. ECIR 2003. Lecture Notes in Computer Science, vol 2633. Springer, Berlin, Heidelberg. Link: Springer