Search Engine Design

CS6200: Information Retrieval

Northeastern University

Fall 2017, Prof. Nada Naji

Team Members:

Mansi Kulkarni

Sougata Dafader

Contents

1. Introduction:

1.1 Overview:

The goal of the project is to design and build our own information retrieval systems, evaluate and compare their performance levels in terms of retrieval effectiveness. This project designs search engines using four distinct retrieval models as mentioned below:

- BM25 retrieval model
- tf-idf retrieval model
- Query-Likelihood language model
- Lucene Systems

Along with these four baseline runs, we have chosen BM25 model for :

- *Pseudo relevance feedback* query expansion technique
- *BM25, tf-idf, Query-likelihood* retrieval model along with *Stopping* technique
- *BM25, tf-idf, Query-likelihood* retrieval model on the stemmed version of the corpus *'cacm_stem.txt'* using *'cacm_stem.query'*.

Now we assess the performance of these eight distinct runs in terms of the following retrieval effectiveness measures:

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

1.2 Contribution of team members:

The detailed contribution of each individual member is as follows:

- Sougata Dafader: Sougata was responsible for implementing the four baseline runs using tf-idf measure, BM25 Model, Query-Likelihood Model and Lucene Systems. He was also responsible for documenting his design choices and developing the Snippet generation logic and query-by-query analysis for stemmed and non-stemmed runs.
- Mansi Kulkarni: Mansi was responsible for implementing query expansion technique using *Pseudo relevance feedback*. Additionally, she calculated the various evaluation measures like MAP, MRR, P@K, Precision & Recall for the seven distinct runs. Moreover, she made contributions in documenting the conclusions and outlooks from the findings, observations and analyses of the results.

2. Literature and resources:

2.1 Overview:

The overview of the techniques used for implementing various aspects of the project are outlined below:

- **BM25 Model**:
  - For BM25 model, we have used '*cacm.rel*' file as the set of relevant documents. The values of '*K*', '*k1*' and *'k2'* are chosen as per TREC standards.
- **tf-idf measure**:
  - For calculating tf-idf for each document, we are using the normalized value of term frequency for the document multiplied by its inverse document frequency.
- **Query Likelihood model**:
  - For calculating Query-Likelihood, we have used frequency of query terms in each document and document length for each document.
- **Lucene**:
  - We have used the standard Lucene open source library with minor modifications to perform indexing and search operations.
- **Query Expansion**:
  - The query expansion for our project has been done using *Pseudo Relevance Feedback* technique and *Rocchio* algorithm.
- **Stopping**:
  - The standard stop list – '*common_words.txt*' has been used to perform stopping. Any word appearing in the above-mentioned stop list has not been indexed.
- **Stemming**:
  - We have performed stemming using the BM25 model with the help of the stemmed version of the corpus *'cacm_stem.txt'* and query *'cacm_stem.query.txt'*.
- **Precision & Recall**:
  - The formulae used for calculations of precision and recall are:
    Precision = |Relevant ∩ Retrieved| / |Retrieved|
    Recall = |Relevant ∩ Retrieved| / |Relevant|
- **MAP**:
  - The formula used for calculation of Mean Average Precision is:
    MAP = Σ Average Precision / Number of Queries
- **MRR**:
  - *Reciprocal rank* is reciprocal of the rank at which the first relevant document is retrieved. *Mean Reciprocal Rank* is the average of the reciprocal ranks over a set of queries.
- **P@K**:
  - *P@K* is calculated as the number of relevant documents in the *top K* retrieved documents. We have calculated for K = 5 and K = 20.

- **Snippet Generation**: We used *Query biased document snippets methodology* to generate the snippets. In this method individual scores of each documents are used for a search model and it's unigram/bigram/trigram is created. Thereafter the result is then used to get context in the snippet from the given document. Snippets with highlighted terms are shown in the console. The background of the highlights is white and the foreground is black.

2.2 Third Party Tools:

The following third party tools have been used in moderation as and when required:

- Beautiful SOUP: A python tool Beautiful SOUP has been used for parsing purpose to process both the documents in the corpus and the queries.
- Lucene Libraries: The following files are required for Lucene implementation:
  - lucene-core-VERSION.jar
  - lucene-queryparser-VERSION.jar
  - lucene-analyzers-common-VERSION.

- Colorama: A python tool Colorama was used for query highlighting during snippet generation. The tool provides a better highlighting technique than just using of quotes or other techniques.

2.3 Research Article References:

The following article was consulted for Query Expansion Technique using Pseudo Relevance Feedback:

http://nlp.stanford.edu/IR-book/pdf/09expand.pdf

The following article was consulted for designing the snippet generation:

http://dl.acm.org/citation.cfm?id=1376651

The following article was consulted for designing the four baseline runs:

http://nlp.stanford.edu/IR-book/essir2011/pdf/11prob.pdf

The following articles were consulted for the various evaluation measures:

- http://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf
- https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf

3.Implementation and Discussion:

The intricacies of the implementation techniques are detailed in this section. It also contains the query-by-query analysis comparing the results of stemmed and non-stemmed runs.

3.1 Baseline Runs:

3.1.1 tf-idf measure:

- For tf-idf measure, the formulae used are given below:

  tf = no. of times a term occurs in a document/ document length
  idf = 1 + log (Total no. of documents) / (number of documents in which term appears) + 1

- For inverse document frequency calculation, add 1 to the denominator to prevent it turn to 0 when the term does not appear in any document. We are also adding 1 to the log values to prevent entire idf value from becoming 0.

Steps to calculate tf-idf:

A. For each query in '*query.txt*' file, perform the following steps:
   a. For each term in the query, calculate tf*idf scores for all the documents in the corpus containing the term.
   b. Calculate total tf*idf scores of documents for all the terms in the query.
   c. Sort documents as per their scores in decreasing order.
   d. Print top 100 documents in '*TFIDF_doc_score.txt*' file in the below format:
      QueryID 'Q0' DocID Rank tf_idf_score ' tf_idf_Model'

3.1.2 BM25 Model:

In our BM25 Model, to calculate the scores of documents use the following scoring function:

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

- The summation is done over all the query terms.
- r_i -> number of relevant documents containing the term i
- n_i -> number of documents containing the term i
- N -> total number of documents in the collection
- R -> number of relevant documents for the query
- f_i -> frequency of the term i in the document
- qf_i -> frequency of the term i in the query
- k_1 -> the value is taken as 1.2 as per TREC standards
- k_2 -> the value is taken as 100 as per TREC standards
- 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 taken as 0.75 as per TREC standards.

- Sort documents as per their scores in decreasing order.
- Print top 100 documents in '*BM25_doc_score.txt*' file in the below format:
  QueryID 'Q0' DocID Rank BM25_score ' BM25_Model'

3.1.3 Query-Likelihood model:

- In the *query likelihood* retrieval model, we rank documents by the probability that the query text could be generated by the document language model. The retrieval model specifies ranking to be documented by $P(Q|D)$, which we calculate using the unigram language model for the document

$$P(Q|D) = \Pi\ i=1 \text{ to } n\ P\ (qi\ |\ D)$$

where *qi* is a query word, and there are *n* words in the query. To calculate this score, we need to have estimates for the language model probabilities $P\ (qi\ |\ D)$.

$$P(qi\ |\ D) = f\ qi, D\ |/|D|$$

where *fqi,D* is the number of times word *qi* occurs in document *D*, and $|D|$ is the number of words in D. The major problem with this estimate is that any of the query words are missing from the document, the score given by query likelihood for P(Q|D) will be zero. *Smoothing* is a technique for avoiding this estimation problem by using the below formula:

$$(1 - \alpha_D)P(q_i / D) + \alpha_D\ P(q_i / C)$$

collection. If $P(q_i / C)$ is the probability for query word *i* in *collection language model* for the document the collection C then the estimate we use for an unseen word in a document is $\alpha_D P(q_i/C)$, where α_D is a coefficient controlling the probability assigned to unseen words.[9] In general, α_D can depend on the document.

- Use formula for smoothed Query likelihood model

  $\log P(Q/D) = \Sigma_{ni=1} \log ((1 - \lambda)\ f_{qi;D} / /D/ + \lambda\ c_{qi} / /C/)$ where value of $\lambda=0.35$.

- Print top 100 documents in 'SQL_*doc_score.txt*' file in the below format:

  QueryID 'Q0' DocID Rank SQL_score 'Smoothed_Query_Likelihood_Model'

3.2 Pseudo Relevance Feedback

For performing pseudo relevance feedback using *Rocchio* Algorithm, perform the following steps:

A. Normal retrieval is performed with the initial query
B. Top k documents (in our case k =10) from the results are included in the relevant set of documents and the rest of the documents are non-relevant set.
C. Rocchio algorithm is used to generate the new query.
   a. Initial query vector with the term frequency scores and aligned with the inverted index terms is generated.
   b. Relevant set of document vector is generated.
   c. Non-relevant set of documents vector is generated.
   d. Then modified query is generated by following the formula:
   e. The algorithm proposes using the modified query qm:
      $qm = \alpha q0 + \beta\ /|Dr|\ \sum dj - \gamma\ /|Dnr|\ \sum dj$
      Where q0 is the original query vector, Dr and Dnr are the set of known relevant and non-relevant documents respectively, and α, β, and γ are weights attached to each term.
      $\alpha = 1.0$, $\beta = 0.75$, and $\gamma = 0.15$.
D. Top 20 terms with the highest weight and which are not present in the query are appended to the original query.
E. Normal retrieval is performed with the new query and the search results are generated.

3.3 Query-by-query analysis:

The two queries that we have chosen are

- Parallel algorithms / parallel algorithm
- Performance evaluation and modelling of computer systems / perform evalu and model of comput system
- Portable Operating Systems / portabl oper system

For the first query, we can see that the stemmed version and non-stemmed version are almost same except for *'algorithms'* getting stemmed to *'algorithm'*. However, we can see that for the second query, five terms:
('*Performance'-> 'perform'; 'evaluation' -> 'evalu'; 'modelling'-> 'model'; 'computer*' -> '*comput*'; *'systems' -> 'system'*) have changed after stemming and for the third query we can see that all the three terms, (portable ->portabl ,Operating->oper, Systems-> system).

In case of stemmed result, all terms belonging to the same stem class are getting replaced by the stem. For example, the terms '*Performance'*, '*performing'*, '*performer'*, '*performed'* etc. will be replaced by the stem '*perform'*. Moreover, the terms '*computer*', '*computing*', '*compute*', '*computed*' etc. will be replaced by '*comput*' and 'operating', 'operable', 'operated', 'operator' will stem to 'oper'. So, the documents having any of the terms that will get transformed to the same stem by the stemming algorithm will have high scores.

On the other hand, in case of the non-stemmed run, each distinct term is considered. For example, the term '*Performance'* is different from '*perform*', '*computer*' is different from '*compute*' etc. As both the queries and documents are not stemmed, each word (even though they might belong to the same stem class) will have different weightage.

In case of the first query, if we consider the top 20 documents retrieved by the BM25 model during the stemmed and non-stemmed runs, we will find that there are eleven common documents *('CACM-0950', 'CACM-1262', 'CACM-2714', 'CACM-2700', 'CACM-2266', 'CACM- 2685*' etc.). This happen because *stemming* has a very little impact on the query terms. In case of the second query, if we *consider* the top documents retrieved by them during the stemmed and non-stemmed runs, we will find *that* there are only six common documents (*'CACM- 3048', 'CACM- 3070', 'CACM-2988'* etc.). This happen *because* stemming has considerable impact on the query terms. In the third query, ('CACM-3127','CACM-2372','CACM-3068' , 'CACM-1750','CACM-2629, 'CACM-2640','CACM-2379','CACM-2950' etc.)are common which are a considerable number of common documents more than the second query but lesser than the first one due to moderate stemming of words belonging to a stem class.

In case of tf-idf and Query-likelihood model as well the stemmed and non-stemmed results obtained indicate there are more Similar documents i.e. greater document overlap for first query, moderate for third query and least for second query due to stemming of words into stem classes.

4. Results:

Given below are the results obtained for different runs:

- Initial Runs:


BM25_doc_score.txt


TFIDF_doc_score.txt


SQL_doc_score.txt


Lucene_doc_score.txt

- Query Expansion Run:


BM25_rel_feed_doc_ score.txt

- Stopping Technique Run:


BM25_stopped_doc score.txt


TFIDF_stopped_doc_score.txt


SQL_doc_score.txt

- Stemming Technique Run:


BM25_stemmed_ doc _score.txt


Stemmed_TFIDF_doc_score.txt


stemmed_QL_doc_score.txt

- Comparison of MRR(s) and MAP(s):


Comparison_ MRR.txt


Comparison_ MAP.txt

5.Conclusions:

5.1 Conclusions:

Our overall findings and conclusions from the eleven assigned runs and after applying various evaluation measures on their result set are outlined below:

- The best System was found to be BM25 from its results which were obtained from the *BM25* document scores incorporating Query *Expansion* technique. As per its *MAP* scores (*0.479730025988*) and *MRR* scores (*0.699869791667*), this run was better than all other retrieval models.
- Even the initial BM*25 results* yielded good in terms of MAP, MRR and was more effective than Lucene, Smoothed Query Likelihood and tf-idf measures.
- However, *BM25* along with stop lists caused a decrease in the values of effectiveness measures of MAP and MRR. This observation was true in case of tf-idf and Smoothed Query-Likelihood model as well and indicates that the list of stop words might be including certain relevant high frequency words which is having a negative impact on the overall search effectiveness.
- BM25 along with stemmed query and stemmed corpus also produces less retrieval effectiveness in terms of BM25 document scores as the words are now stemmed to their stem classes which will result in more overlap for documents for all the words contained in those documents.
- tf-idf measures have produced the least values for all the evaluation techniques. Even the p@5 and p@20 values for many queries has been 0 indicating that zero relevant documents were retrieved in top 5/20 ranked documents. So, ideally tf-idf does not serve as a good measure for retrieval model.
- Lucene provides slightly better MAP and MRR values than Query-likelihood measures. This observation might be attributed to the fact that it incorporates both Boolean model along with Vector Space Model.

5.2 Outlook

Our project takes into account all the basic features of an information retrieval system. However, certain features and functionalities can be added to make their performance even better in terms of future improvements and effectiveness as follows:

- We can consider topical features of a document, taking into consideration their quality features like '*incoming links'*, 'outgoing links ', '*update count'* etc. as a part of the final ranking. Using '*PageRank'* algorithm to calculate the popularity of a page is also a viable method which can be incorporated in our search engines.
- We can also incorporate HITS algorithm including good hubs and good authority scores to improve the retrieval effectiveness of relevant documents.
- As the corpus grows, it will be good to use different compression and encoding techniques like '*Elias-γ encoding*', '*v-byte encoding*' to store the inverted index.
- Meta data can be incorporated to further improve retrieval and relevance effectiveness.

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.* Cambridge England: Cambridge University Press 2009.

6.2 Scholarly articles:

- http://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf
- https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf
- http://nlp.stanford.edu/IR-book/pdf/09expand.pdf
- http://nlp.stanford.edu/IR-book/essir2011/pdf/11prob.pdf
- http://dl.acm.org/citation.cfm?id=1376651