Information Retrieval Systems

Project

(Submitted in partial fulfillment of the requirements of Northeastern University's CS6200 Fall 2017 course, taught by Prof. Nada Naji)

Preethi Anbunathan
Vaibhav Rangarajan
Srijit Ravishankar

CONTENTS

1. INTRODUCTION

GOAL:

To design and build information retrieval system**s,** evaluate and compare their performance levels in terms of retrieval effectiveness.

PHASES:

1. **INDEXING AND RETRIEVAL:**
   - ➢ BM25, tf-idf, Smoothed Query Likelihood Model and lucene are used as retrieval models combined with a word unigram indexer. The top 100 retrieved ranked lists are reported.
   - ➢ BM25 is chosen to perform Pseudo Relevance Feedback and the results are compared.
   - ➢ Stopping is performed without stemming on BM25, tf-idf and Smoothed Query Likelihood model.
   - ➢ Query by query analysis is performed for three queries.

2. **RESULT DISPLAY:**
   - ➢ A snippet generation technique and query time highlighting within the results in Lucene are implemented.

3. **EVALUATION:**
   - ➢ To assess the performance of the retrieval systems in terms of effectiveness, the following are performed:
     - MAP
     - MRR
     - P@K, K=5 and 20
     - Precision and Recall

TEAM MEMBER CONTRIBUTIONS:

1. **VAIBHAV RANGARAJAN:** Vaibhav was responsible for implementing the various evaluation measures like MAP, MRR, P@K, Precision & Recall for the seven distinct runs. Moreover, he made notable contributions in documenting the conclusions and outlooks from the findings, observations and analyses of the results.

2. **SRIJIT RAVISHANKAR:** Implemented the four baseline runs using tf-idf measure, BM25 Model, Smoothed query likelihood model and Lucene Systems. He was also responsible for documenting his design choices and developing the Snippet generation logic.

3. **PREETHI ANBUNATHAN:** Implemented the query expansion technique using Pseudo relevance feedback. Additionally, developed the search engines which incorporate the stemming and stopping techniques. Carried over the corresponding documentation and query-by-query analysis for stemmed and non-stemmed runs.

2. LITERATURE AND RESOURCES

OVERVIEW:

The overview of the techniques used in the implementation are:

- **BM25 Model:** The values of K, k1 and k2 are chosen as per TREC standards. Given relevant documents has been used for computing the scores.
- **tf-idf:** The product of term frequency of the document and the inverse document frequency are computed to obtain the scores.
- **Smoothed Query Likelihood Model:** Documents are ranked by the probability that the query could be generated by the document model. Jelinek-Mercer smoothing is performed to overcome the problem of zero probability occurrence of a missing word.
- **Lucene:** Lucene's default retrieval model is used for obtaining scores.
- **Pseudo Relevance Feedback:** Rocchio algorithm and pseudo relevance feedback technique is implemented for query expansion.
- **Stopping:** The given stop list is used and removed from the documents and they have not been indexed.
- **Stemming:** The given stemmed version of the corpus is used to perform stemming.
- **Snippet generation:** Snippets to display are obtained for every document and query term highlighting is done using Colorama for every snippet.
- **Mean Average Precision:** MAP is computed using the following formula,

$$MAP = \sum \text{Average Precision} / \text{Total number of queries}$$

- **Mean Reciprocal Rank:** The reciprocal rank of a query response is the multiplicative inverse of the rank of the first relevant document. MRR is the average of reciprocal ranks over a set of queries.
- **P@K:** Precision @ K=5 and K=20 and calculated using the following formula,

$$Precision = | Relevant \cap Retrieved | / | Retrieved |$$

- **Precision and Recall:** Precision and recall for every document is computed using the following formula,

$$Precision = | Relevant \cap Retrieved | / | Retrieved |$$
$$Recall = | Relevant \cap Retrieved | / | Relevant |$$

THIRD PARTY LIBRARIES:

The following third-party libraries are required for the implementation:

- **Lucene libraries:** The following are to be added:
  1. lucene-core-VERSION.jar

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

- **Beautiful Soup:** Documents and the queries are parsed using Beautiful soup and the pre-processing works are done.
- **Colorama:** For the purpose of query term highlighting in the snippets generated, colorama is used.

SCHOLARLY WORKS AND RESEARCH ARTICLES REFERENCES:

The following articles were referred for the implementation,

Smoothed Query Likelihood Model:

- http://projects.ict.usc.edu/nld/ir-class/sites/projects.ict.usc.edu.nld.ir-class/files/slides/09.pdf

BM25:

- https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model-1.html

tf-idf:

- https://lizrush.gitbooks.io/algorithms-for-webdevs-ebook/content/chapters/tf-idf.html

Snippet Generation:

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

Query term highlighting:

- https://pypi.python.org/pypi/colorama

3. IMPLEMENTATION AND DISCUSSION

The different phases of the implementation are detailed with the discussion of results for each implementation.

1. IMPLEMENTATION

1.1 BM25

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document. The following formula is used for computing scores,

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

- 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 - k1 ((1 – b) + b * dl /avdl) where dl is document length, avdl is average length of document in the collection.
- b - 0.75 as per TREC standards.

The documents are ranked in the decreasing order of their score and are displayed in the following format,

QueryID 'Q0' DocID Rank tf_idf_score ' BM25_Model'

1.2 tf-idf

The tf-idf scores of the documents are computed using the following formula,

- The log term frequency of a term t in d is defined as

$$1+\log(1+tf_{t,d})1+\log(1+tf_{t,d})$$

- The log inverse document frequency which measures the informativeness of a term is defined as:

$$idf_t = \log_{10} N / df_t$$

where N is the total number of documents in the collection

- Combining the two, the tf-idf score is given by

$$w_{t,d} = (1+\log(1+tf_{t,d})).\log_{10} N/df_t$$

- The tf.idf score increases with number of occurrences within a document
- The tf.idf score increases with rarity of terms in the collection
- The documents are ranked in the decreasing order of their score and are displayed in the following format,

QueryID 'Q0' DocID Rank tf_idf_score ' tf-idf_Model'

1.3 Smoothed Query Likelihood Model

- The documents are ranked by the probability that the query could be generated by the document model.
- Given a query, compute the score by performing Jelinek-Mercer smoothing.
- The following formula is used for computing the score,

$$P(Q|D) = \prod^{n}_{i=1} ((1-\lambda)\ f_{qi,D}/\ |D| + \lambda\ c_{qi}/\ |C|)$$

- $\lambda = 0.35$ as given in the specification
- $f_{qi,D}$ represents the frequency of query term in the document
- $|D|$ is the total number of words in the document
- c_{qi} represents the frequency of query term in the collection
- $|C|$ is the total number of words in the collection
- Logarithm is taken for this value to overcome the accuracy problems that occur in multiplying small numbers.
- The modified formula is as follows,

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

1.4 Lucene

The default retrieval model implemented in Lucene is used for calculating the scores. Query parser and Lucene analyzers are the default packages used.

1.5 Pseudo Relevance Feedback

For performing pseudo relevance feedback using Rocchio Algorithm, the following steps are performed,

- Normal retrieval is performed with the initial query
- The top k documents (k =10 chosen) from the results are included in the relevant set of documents and the rest of the documents are non-relevant set.
- Rocchio algorithm is used to generate the new query.
  i. Initial query vector with the tf scores and aligned with the inverted index terms is generated.
  ii. Relevant set of document vector is generated.
  iii. Non-relevant set of documents vector is generated.
  iv. Then modified query is generated by following the formula.
  v. 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 nonrelevant documents respectively, and α, β, and γ are weights attached to each term.
  For our project, we have chosen $\alpha = 1.0$, $\beta = 0.75$, and $\gamma = 0.15$
- The top 20 terms with the highest weight and which are not present in the query are appended to the original query.
- Normal retrieval is performed with the new query and the search results are generated.

1.6 Query-by-query analysis

The three queries chosen for the query-by-query analysis are the following,

- Before stemming- portable operating systems
  After stemming - portabl oper system
- Before stemming- code optimization for space efficiency
  After stemming - code optim for space effici

- Before stemming- parallel algorithms
  After stemming – parallel algorithm

A. *portable operating systems/ portabl oper system*

All the terms in the stemmed and non-stemmed version are different. The terms are stemmed as follows, portable -> portabl, operating -> oper, systems -> system.

When stemming is performed the terms 'portable', 'portables' are stemmed to 'portabl'. Similarly, the terms 'systems', 'systematic', 'systematical' are stemmed to 'system' and 'operating', 'opera', 'operation' are stemmed to 'oper'.

BM25:

The top 20 document ranks for the stemmed BM25 rank list and the non-stemmed BM25 ranked list have 5 documents in common, which are CACM-3127, CACM-3068, CACM-1750, CACM-1680 and CACM-2740. This is because the stemming has a considerable impact on the query terms.

Lucene:

The top 20 document ranks for the stemmed Lucene rank list and the non-stemmed Lucene ranked list have 8 documents in common, which are CACM-3127, CACM-1461, CACM-3068, CACM-2246, CACM-2319, CACM-2740, CACM-2597 and CACM-2629. This is because the stemming has a considerable impact on the query terms.

tf-idf:

The top 20 document ranks for the stemmed tf-idf rank list and the non-stemmed tf-idf ranked list have 6 documents in common, which are CACM-3127, CACM-2897, CACM-2495, CACM-2246, CACM-2319 and CACM-2740. This is because the stemming has a considerable impact on the query terms.

B. *code optimization for space efficiency/ code optim for space effici*

The terms are stemmed as follows optimization ->optim, efficiency-> effici. Hence here the stemmed and non-stemmed version are different.

When stemming is performed the terms 'optimization', 'optimize', 'optimal', 'optimum' are stemmed to 'optim'. Similarly, 'efficiency' is stemmed to 'effici'.

BM25:

The top 20 document ranks for the stemmed BM25 rank list and the non-stemmed BM25 ranked list have 4 documents in common, which are CACM-1795, CACM-2897, CACM-2495 and CACM-2491. This is because the stemming has a considerable impact on the query terms.

Lucene:

The top 20 document ranks for the stemmed Lucene rank list and the non-stemmed Lucene ranked list have 9 documents in common, which are CACM-3100, CACM-1211, CACM-3068, CACM-2246, CACM-2319, CACM-1740, CACM-2597, CACM-1111 and CACM-2629. This is because the stemming has a considerable impact on the query terms.

tf-idf:

The top 20 document ranks for the stemmed tf-idf rank list and the non-stemmed tf-idf ranked list have 6 documents in common, which are CACM-1461, CACM-3068, CACM-2246, CACM-2319, CACM-2740, CACM-2597. This is because the stemming has a considerable impact on the query terms.

C. parallel algorithms/ parallel algorithm

The stemmed version and non-stemmed version are almost same except for ‘algorithms’ getting stemmed to ‘algorithm’.

BM25:

The top 20 document ranks for the stemmed BM25 rank list and the non-stemmed BM25 ranked list have 11 documents in common, which are CACM-0950, CACM-1262, CACM-2714, CACM-2700, CACM-2266, CACM- 2685 etc. This happens because stemming has very less impact on the query terms.

Lucene:

The top 20 document ranks for the stemmed Lucene rank list and the non-stemmed Lucene ranked list have 10 documents in common, which are CACM-3127, CACM-1461, CACM-3068, CACM-2246, CACM-2319, CACM-2740, CACM-2597, CACM-2629 etc. This is because the stemming has very less impact on the query terms.

tf-idf:

The top 20 document ranks for the stemmed tf-idf rank list and the non-stemmed tf-idf ranked list have 11 documents in common, which are CACM-3100, CACM-1211, CACM-3068, CACM-2246, CACM-2319, CACM-1740, CACM-2597, CACM-1111 etc. This is because the stemming has a considerable impact on the query terms.

4. RESULTS

The following documents contain the results of the implementation and comparison for the various baseline runs are performed.

- **PHASE 1 - TASK 1**

  BM25 - BM25_SCORE_LIST.txt
  Lucene - LUCENE_SCORE_LIST.txt
  Smoothed query likelihood - SMOOTHED_MODEL_SCORE_LIST.txt
  tf-idf - TF_IDF_SCORE_LIST.txt

  PHASE 1 – TASK 2

  Query expansion - PSEUDO_RELEVANCE_BM25_SCORE_LIST.txt

  PHASE 1 – TASK 3

  Stopping BM25 – BM25_SCORE_STOPPED_LIST.txt
  Stopping Lucene – LUCENE_SCORE_STOPPED_LIST.txt
  Stopping tf-idf – TF_IDF_SCORE_STOPPED_LIST.txt
  Stemming BM25 - BM25_SCORE_STEMMED_LIST.txt
  Stemming Lucene - LUCENE_SCORE_STEMMED_LIST.txt
  Stemming tf-idf - TF_IDF_SCORE_STEMMED_LIST.txt

- **PHASE 2**

  Snippet Generation – Once run the snippets are generated with query terms highlighted.

- **PHASE 3**

  PRECISION, RECALL

  BM25_SCORE_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
  LUCENE_SCORE_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
  PSEUDO_RELEVANCE_BM25_SCORE_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
  SMOOTHED_MODEL_SCORE_LIST_FOR_EVALUATION_PRECISION-RECALL.txt

TF_IDF_SCORE_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
BM25_SCORE_STOPPED_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
LUCENE_SCORE_STOPPED_LIST_FOR_EVALUATION_PRECISION-RECALL.txt
TF_IDF_SCORE_STOPPED_LIST_FOR_EVALUATION_PRECISION-RECALL.txt

P@K, K=5 and K=20

BM25_SCORE_LIST_FOR_EVALUATION_P5_SCORE.txt
BM25_SCORE_LIST_FOR_EVALUATION_P20_SCORE.txt
BM25_SCORE_STOPPED_LIST_FOR_EVALUATION_P5_SCORE.txt
BM25_SCORE_ STOPPED_LIST_FOR_EVALUATION_P20_SCORE.txt
LUCENE_SCORE_LIST_FOR_EVALUATION_P5_SCORE.txt
LUCENE_SCORE_LIST_FOR_EVALUATION_P20_SCORE.txt
LUCENE_SCORE_ STOPPED_LIST_FOR_EVALUATION_P5_SCORE.txt
LUCENE_SCORE_ STOPPED_LIST_FOR_EVALUATION_P20_SCORE.txt
PSEUDO_RELEVANCE_BM25_SCORE_LIST_FOR_EVALUATION_P5_SCORE.txt
PSEUDO_RELEVANCE_BM25_SCORE_LIST_FOR_EVALUATION_P20_SCORE.txt
SMOOTHED_MODEL_SCORE_LIST_FOR_EVALUATION_P5_SCORE.txt
SMOOTHED_MODEL_SCORE_LIST_FOR_EVALUATION_P20_SCORE.txt
TF_IDF_SCORE_LIST_FOR_EVALUATION_P5_SCORE.txt
TF_IDF_SCORE_LIST_FOR_EVALUATION_P20_SCORE.txt
TF_IDF_SCORE_ STOPPED_LIST_FOR_EVALUATION_P5_SCORE.txt
TF_IDF_SCORE_ STOPPED_LIST_FOR_EVALUATION_P20_SCORE.txt

MAP

MAP_SCORES.txt

MRR

MRR_SCORES.txt

5. CONCLUSION AND OUTLOOKS

CONCLUSION

- Our overall findings and conclusions from the eight assigned runs and after applying various evaluation measures on their result set are outlined.
- The best results were obtained from the BM25 run which incorporated Query Expansion technique (Pseudo Relevance feedback).
- As per its MAP scores (0.5126525699923674) and MRR scores (0.6753968253968254), this run was undisputedly better than all other retrieval models.
- Even the base run of BM25 yielded good results in terms of MAP, MRR and was more effective than Lucene, smoothed query likelihood model and tf-idf measures.
- However, BM25 along with stop lists caused a slight decrease in the values of effectiveness measures.
- This observation 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.
- 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 out of 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 smoothed query likelihood model. This observation might be attributed to the fact that it incorporates both Boolean model along with Vector Space Model.

OUTLOOK

- Our project incorporates all the basic features of an information retrieval system. However, certain features and functionalities as mentioned below can be included in future to hone the performance of the search engines:

- Alongside considering topical features of a document, we can take into consideration their quality features like ‘incoming 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.
- As the corpus grows, it will be prudent to use different compression techniques like ‘Elias-γ encoding’, ‘v-byte encoding’ to store the inverted index.
- The primary motivation of our IR system was to achieve retrieval effectiveness.
- Going forward, we can also take retrieval efficiency into consideration.

6. BIBLIOGRAPHY

1. Iman, Hazra, and Aditi Sharan. "Selecting Effective Expansion Terms for Better Information Retrieval." International Journal of Computer Science and Applications 7.2 (2010): 52-64. Web. 4 Dec. 2016. <http://www.tmrfindia.org/ijcsa/v7i24.pdf>.

2. Feng, Zheyun. "Query Expansion by Pseudo Relevance Feedback." N.p., 27 Aug. 2012. <http://fengzheyun.github.io/download/projects/before2015/document/DocRetri.pdf>

3. Pal, Dipasree, Mandar Mitra, and Kalyankumar Datta. "Query Expansion Using Term Distribution and Term Association." N.p., 4 Mar. 2013. Web. 5 Dec. 2016. <https://arxiv.org/abs/1303.0667>.

4. Xu, Jinxi, and W. Bruce Croft. "4. Query Expansion Using Local and Global Document Analysis." N.p.,02 Oct. 2010. Web. 05 Dec. 2016. <http://www.eng.utah.edu/~cs7961/papers/XuCroft-SIGIR96.pdf>.

5. Joachims, Thorsten. "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization." N.p., 1997. Web. 4 Dec. 2016. <https://www.cs.cornell.edu/people/tj/publications/joachims_97a.pdf>.

6. Verstak, Alexandre A., and Anurag Acharya. Generation of Document Snippets Based on Queries and Search Results. Patent US 8145617 B1. 27 Mar. 2012. Print.

7. Carr, Ian, and Lucy Vasserman. "SnipIt - A Real Time Dynamic Programing Approach to Snippet Generation for HTML Search Engines." (n.d.): n. pag. Web. <http://www.cs.pomona.edu/~dkauchak/ir_project/whitepapers/Snippet-IL.pdf>.