CS 6200 Information Retrieval Final Project Report

Spring 2019 —Professor Rukmini Vijaykumar

Brian Desnoyers
Khoury College of Computer Sciences
Northeastern University
Boston, MA, U.S.A.
bdesnoy@ccs.neu.edu

John Goodacre
Khoury College of Computer Sciences
Northeastern University
Boston, MA, U.S.A.
goodacre.j@husky.neu.edu

Akshay Kulkarni
Khoury College of Computer Sciences
Northeastern University
Boston, MA, U.S.A.
kulkarni.akshay@husky.neu.edu

I. Introduction

This project explores the development and evaluation of several search engines for a provided test collection of scholarly journal abstracts. Baseline systems were improved via query enhancement techniques, such as word embedding query expansion and stemming. A snippet generation and spelling error correction model were also developed to improve the baseline retrieval systems.

Team member contributions (in addition to the homework retrieval models) are listed below:

- Brian Desnoyers
  - Baseline Query Likelihood Model (JM smoothed)
  - Baseline Query Likelihood Model (Dirichlet Smoothed)
  - Word Embedding-Based Query Expansion
  - Task 3- Stopping and Stemming
  - Snippet Generation and Query Term Highlighting
  - Phase 3- Evaluation
  - Relevant Sections of Readme Documentation
  - Relevant Sections of Final Report
- John Goodacre
  - Task 3- Stopping
  - Relevant Sections of Readme Documentation
  - Relevant Sections of Final Report
- Akshay Kulkarni
  - Vector-Space Term Frequency–Inverse Document Frequency model
  - Rocchio Algorithm for Relevance Feedback w/ Vector-Space Model
  - Relevant Sections of Readme Documentation
  - Relevant Sections of Final Report

II. Background

A. Dataset

This project utilizes a dataset consisting of 3024 abstracts from the Communications of the ACM (CACM) published between 1958 and 1979 [1]. In addition, the DBLP Computer Science Bibliography was utilized for generating models used for query expansion. This database is a collection citation information from computer science journal articles [2].

B. Baseline Models

This project involves the development of several information retrieval systems, with an evaluation and comparison of their performance in terms of retrieval effectiveness. This involves four baseline models.

The first baseline model is a Best Matching 25 (BM25) ranking [3], which expands the binary independence model to incorporate term weights for both queries and documents. This method attempts to estimate the probability that a particular document is relevant, assuming that terms are independent given document relevance [4], similar to a Naïve Bayes classifier. The BM25 algorithm can also be used in the absence of relevance information, however. The BM25 ranking algorithm is thus a probabilistic model.

The second and third baseline models are both query likelihood models which compute the probability of a query based on the language model for each document. These models use simple unigram language models for each document. Because it is unlikely that all relevant documents will contain the query terms, in order for their language models to generate those terms, smoothing must be used. One of these models uses Jelinek-Mercer (JM) smoothing, which uses a probability from a collection-wide language model to adjust the probability citecroftsearch. The weight of this term is adjusted by a parameter λ, which increases the smoothing effect. Thus small values, such as 0.1 tend to work well for shorter queries, while larger values, such as 0.7 tend to work better for longer queries [1]. The other model uses Dirichlet Smoothing /citecroftsearch. which instead uses a parameter μ, which increases the weighting of documents with more matching terms [1]. Dirichlet Smoothing typically performs better than JM smoothing on smaller queries, and typical

values for μ are between 1000 and 2000.

The fourth (an extra) baseline model is a term frequency–inverse document frequency (tf-idf) vector space model [5]. This model involves computing distances between document and query vectors, through metrics such as cosine distance, which was utilized for this project. These vectors are computed based on the tf-idf values for each term within the vocabulary, which is product of the term frequency within the document and the reciprocal document frequency of the term [5]. The Tf-Idf Vector Space model has been implemented by computing ranking scores based on dot product of cosine weights of tf-idf vectors computed from a document index, In order to reduce the impact of frequent terms, log transformation with laplace smoothing was applied to raw term frequencies. Additionally Rocchio relevance feedback method (Rocchio's algorithm) was incorporated into the Vector Space model as a query modification method using the relevance information provided, which transforms the query term weights in the query vector by adding a component based on the average of weights in the relevant documents and negating by a component based on the average weight in the non-relevant documents.

The final baseline model is Lucene, which is an open-source indexing and searching tool [6], and is used within common search engine implementations, such as Apache Solr [7] and Elasticsearch [8]. Lucene uses its own scoring function which is based on tf-idf [6].

C. Query Enhancement

1) Query Expansion: Query expansion is a method used to improve retrieval system performance and relevance by adding additional terms to a query [9]. Several methods for query expansion exist, including pseudo-relevance feedback and word embedding-based query expansion. Pseudo-relevance feedback uses highly ranked documents to select new terms for query expansion and has been shown to be effective both with and without improvements, such as a term proximity heuristic for selecting expansion terms near query terms [10]. Another modern query expansion technique involves the use of word embeddings, which are commonly used in natural language processing. Word embeddings map co-occurance information to a lower dimensional space, often using a skip-gram or continuous bag of words model. Query expansion terms can be identified by finding nearest neighbors to query terms via these models [11].

2) Stopping: Stopping is a technique used to filter out specific words, known as stop words. Ideally these stop words will be common across many documents in the collection and thus have limited semantic meaning or impact on document relevance. Words might carry little meaning from a frequency (or information theoretic) point of view, or alternatively from a conceptual (or linguistic) point of view. Words that occur in many of the documents in the collection carry little meaning from a frequency point of view, because a search for documents that contain that word will retrieve many of the documents in the collection. By removing the very frequent words, the document rankings will not be affected that much. While these stop words can be removed at index time, but are often used only at query time as this allows for more flexibility during search. Stop word removal on the basis of frequency can be done easily by removing the words with the highest frequencies in the document collection. As a result of stopping the most 200-300 frequent words, indexes may be between 30% and 50% smaller [12].

3) Stemming: Stemming is the process of reducing all words with the same root (or, if prefixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes. There are various stemming strategies developed for different purposes [13]. Some stemming algorithms utilize a stem dictionary and others a suffix list. Many stemming algorithms, designed to improve IR performance and document relevance, do not use a stem dictionary, but an explicit list of suffixes, and the criteria for removing suffixes. Stemmers of the popular stemmer family, the Porter stemmers, have adopted this approach [14].

D. Snippet Generation and Highlighting

A classical approach for snippet generation is extractive summarization using an extension of Luhn's Algorithm [15]. This algorithm involves the identification of key phrases within sentences containing query terms. The query terms identified within these key phrases can be highlighted to the user via the search engine interface.

III. Implementation and Discussion

A. Baseline Model Implementation

The baseline models will be implemented in Python. The tf-idf vector space model was implemented by computing ranking scores based on cosine distance of tf-idf vectors computed based on a document index. This utilized a vocabulary generated from the index, with Laplace smoothing. Similarly, the query liklihood and BM25 baselines were implemented in Python. The final Lucene-based model was developed using PyLucene [16].

B. Query Enhancement

Query expansion was performed via pseudo-relevance feedback and Word2vec-trained [17] word embeddings. For pseudo-relevance feedback, 6 query terms were selected based on the 10 top ranked documents to find the expansion terms. Due to the limited size of the training set, a set of word embeddings from about three million article titles from the DBLP Computer Science Bibliography. This utilized a continuous bag of words model with a vector size of 100 and window size of 10 for use during training, as these are common parameters for these purposes [18]. The Gensim library was used to perform initial processing of these word embeddings

[19]. Stopping utilized the provided project stop list. Stemming utilized the provided query and document files.

C. Snippet Generation and Highlighting

Snippet generation was performed via extractive summarization using an extension of Luhn's Algorithm. Sentences were ranked based on a computed significance factor, which is the number of query terms within the key phrase divided by the total number of words in the key phrase. Initial query terms were extracted and highlighted within the terminal output (`\033[1m term \033[0m`).

D. Combining Approaches

To combine approaches, a final run was performed that combines stopping with the word embedding query expansion technique.

E. Extra Credit

A spelling error interface was implemented. This interface used a unigram language model trained via the dataset which was used in conjunction with a noisy channel model to provide up to 6 suggestions with the highest probability for each query word not in the index. As shown in Figure 1, this spelling corrector also found similar words such as singular/plural forms when one wasn't in the index.

F. Query-by-Query Analysis

Looking at, for example, the query "What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?," the top result for BM25 and the QLM discuss storage structures which is not relevant. The top document for the Lucene model discusses a computer sharing system. This result occurs even when stopping is applied to the BM25 model. When stopping is added to Lucene, however, the first result is about a time sharing system. In this single query-by-query example, the stopping seems to have a more positive impact on the Lucene model. This is discussed and analyzed in more detail across the dataset using evaluation metrics as described in the Results section of this document.

Query-by-query analysis for the stemming results is included in the Results section of this document.

IV. Results

The summary of results across all runs is shown in Table 2. This includes mean average precision (MAP), mean reciprocal rank (MRR), precision at k for $k = 5$ and $k = 20$ (P@5, P@20), overall precision (for all 100 documents), and overall recall (for all 100 documents). Complete query-by-query results are included with this submission as described in `Readme.txt` (files named `results_*.txt`). Similarly, full tables for all evaluation metrics are included with this submission as described in `Readme.txt` (files named `eval_*.txt`). For the stopping there was 77.4% overlap for the BM25 results and 84.9% overlap for the Lucene results.

Results are interpreted within the Conclusion section of this document.

A. Stemming Results

While the stemming results could not be directly compared, they yielded highly similar results that appeared to be relevant qualitatively. For example, for the first query, "portabl oper system," the top results from both systems were extremely similar, which was not the case for results from the previous runs. This is likely not only due to the stemming, but also because of the fact that the queries used for the stemming sources were much more succinct. This demonstrates that many of the models used, specifically the BM25 and standard Lucene models, may work better for concise keyword queries, rather than prose. Similar results occurred for the second query "code optim for space effici", and the top hits referring to space efficient code, such as the top result about indirect threaded code (`CACM-2748.html`).

V. Conclusion

From these result tables, it is clear that the TF-IDF model using Rocchio relevance feedback performed best, however, it should be noted that it had ground-truth relevance information available. This therefore likely artificially inflated the results, as seen by the mean reciprocal rank value of 1.0. While the standard TF-IDF vector space model performed very well in comparison to other systems (e.g. its MAP was similar to the BM25 algorithm after stopping), its efficiency made its use significantly less desirable. One area for future work is to improve the tf-idf vector space model as it had limited performance during this assignment, taking almost an hour to run through all the queries.

These tables also show that the impacts of enhancements, such as query expansion, may have been limited by the non-succinctness of the queries used. While results during expansion, especially for the word embedding-based query expansion, were able to find similar terms (e.g. system to systems), they also found synonyms for non-stopwords that should be removed. As a result, these measures appeared to only moderately improve performance.

Despite the traditional improvement of Dirichlet smoothing over JM smoothing for a query likelihood model [1], the JM smoothed model appeared to significantly outperform the other model, likely due to the length of the queries [1].

Similarly the BM25 model seemed to slightly perform the Lucene-based model based on its MAP, MRR, and P@5 values.

References

[1] B. Croft, D. Metzler, and T. Strohman, "Search engines: Information retrieval in practice, 2008."

[2] M. Ley, "The dblp computer science bibliography: Evolution, research issues, perspectives," in *International symposium on string processing and information retrieval*. Springer, 2002, pp. 1–10.

[3] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford *et al.*, "Okapi at trec-3," *Nist Special Publication Sp*, vol. 109, p. 109, 1995.

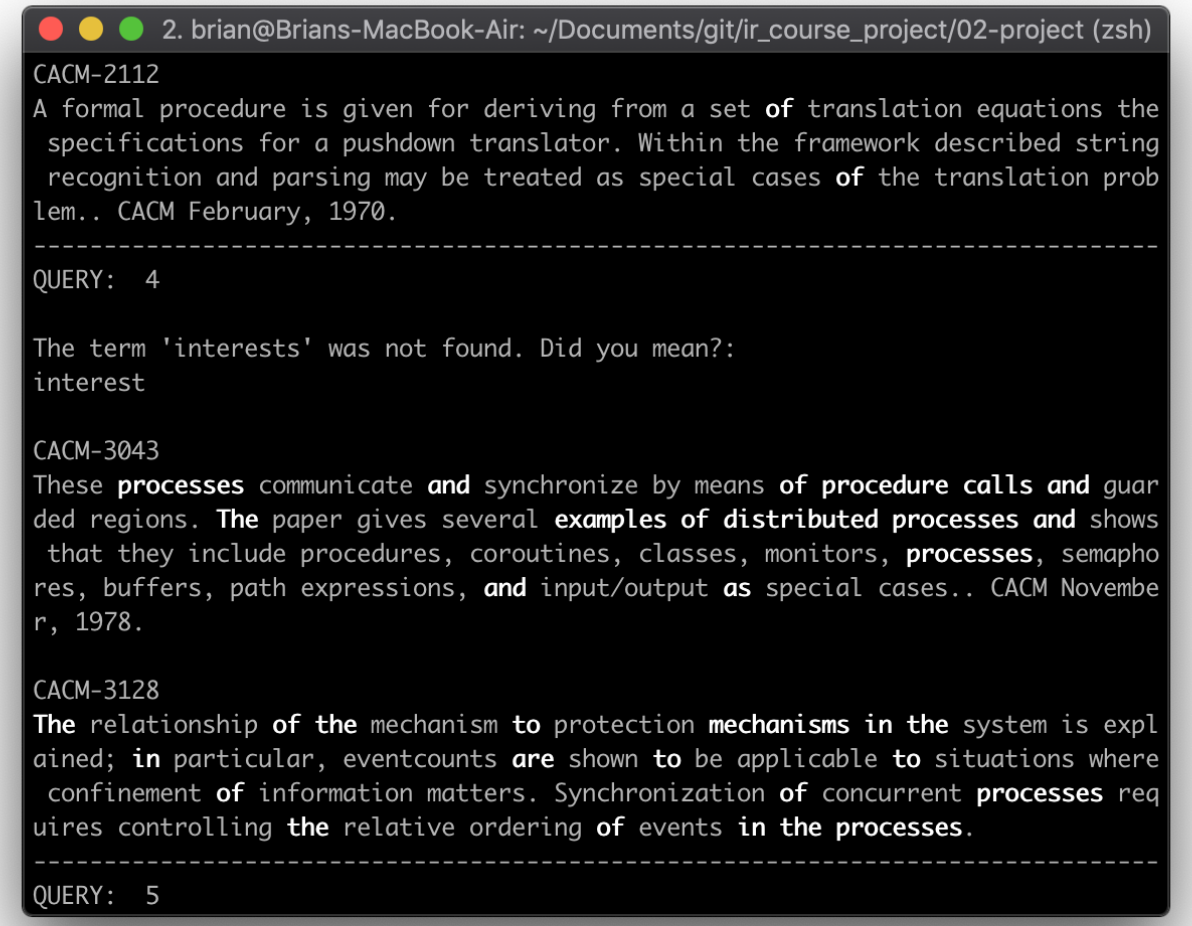

Fig. 1. An example suggestion from the spelling corrector. In this example, since query expansion was not used, a suggestion was made to change a plural form to the singlular.

[4] C. T. Yu and G. Salton, "Precision weighting-an effective automatic indexing method," Cornell University, Tech. Rep., 1975.

[5] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[6] A. Białecki, R. Muir, G. Ingersoll, and L. Imagination, "Apache lucene 4," in *SIGIR 2012 wWrkshop on Open Source Information Retrieval*, 2012, p. 17.

[7] T. Grainger and T. Potter, *Solr in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2014.

[8] M. S. Divya and S. K. Goyal, "Elasticsearch: An advanced and quick search technique to handle voluminous data," *Compusoft*, vol. 2, no. 6, p. 171, 2013.

[9] E. N. Efthimiadis, "Query expansion." *Annual review of information science and technology (ARIST)*, vol. 31, pp. 121–87, 1996.

[10] Y. Lv and C. Zhai, "Positional relevance model for pseudo-relevance feedback," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '10. New York, NY, USA: ACM, 2010, pp. 579–586. [Online]. Available: http://doi.acm.org.libproxy.mit.edu/10.1145/1835449.1835546

[11] F. Diaz, B. Mitra, and N. Craswell, "Query expansion with locally-trained word embeddings," *arXiv preprint arXiv:1605.07891*, 2016.

[12] D. Hiemstra and F. de Jong, "Statistical language models and information retrieval: natural language processing really meets retrieval." 2001.

[13] J. B. Lovins, "Development of a stemming algorithm," *Mech. Translat. & Comp. Linguistics*, vol. 11, pp. 22–31, 1968.

[14] E. Airio, "Word normalization and decompounding in mono-and bilingual ir," *Information Retrieval*, vol. 9, no. 3, pp. 249–271, 2006.

[15] H. P. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, no. 2, pp. 159–165, 1958.

[16] A. Vajda, "Pulling java lucene into python: Pylucene," *Retrieved March*, vol. 23, p. 2008, 2005.

[17] T. Mikolov, K. Chen, G. S. Corrado, and J. A. Dean, "Computing numeric representations of words in a high-dimensional space," May 19 2015, uS Patent 9,037,464.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[19] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on*

	MAP	MRR	P@5	P@20	Precision	Recall
Baseline BM25	0.1596	0.7214	0.3808	0.2106	0.0781	0.6260
Baseline Lucene	0.1565	0.7010	0.3731	0.2087	0.0792	0.6323
Baseline Query Likelihood Model (JM smoothed)	0.1459	0.6557	0.3346	0.1952	0.0719	0.5889
Baseline Query Likelihood Model (Dirichlet Smoothed)	0.0452	0.1811	0.0846	0.0587	0.0306	0.2367
Pseudo Relevance Feedback Lucene	0.1458	0.5757	0.2962	0.1981	0.0773	0.6083
Word Embedding Query Expansion Lucene	0.1417	0.5741	0.3231	0.1923	0.0719	0.5818
Stopping BM25	0.1679	0.7396	0.3731	0.2259	0.0804	0.6368
Stopping Lucene	0.1611	0.7156	0.3769	0.2154	0.0795	0.6364
Stopping Word Embedding Query Expansion Lucene	0.1452	0.5766	0.3192	0.1990	0.0746	0.5994
TF-IDF Vector Space Model	0.1679	0.6708	0.3884	0.2288	0.0859	0.6605
TF-IDF Vector Space Model w/ Rocchio Algorithm	0.3441	1.0000	0.8731	0.5135	0.1369	0.9322

Fig. 2. A summary of the result table across all main runs.

New Challenges for NLP Frameworks. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.