CS6200 PROJECT REPORT

SEMESTER: Spring 2019

PROFESSOR:

Prof: Rukmini Vijaykumar

PREPARED FOR

CS 6200, Information Retrieval

Northeastern University

PREPARED BY

1. Sahithi Vankayalapati 001425782
2. Karan Singh 001425775
3. Srinivasa Vijay Bhaskar Vemuri 001464009

1. Introduction:

The objective of this project is to build an Information Retrieval System by implementing the core concepts and techniques of Information retrieval that we learned in this course. And also, evaluate and compare the performance of different Retrieval Models in terms of Retrieval Effectiveness.

In this project we have designed our very own Information Retrieval Systems from scratch. We have designed this system in various phases of Indexing, Retrieval and Evaluation. We have built an indexer for the corpus, obtained ranked list of documents using various Retrieval Models and performed query expansion using 2 different techniques. We have also performed Evaluation of our Retrieval Models using various Effectiveness Metrics.

We also perform tests to find the differences in performance, implement snippet generation and query term highlighting on documents for a query.

1.2 Contributions

The contribution of each member to the project is as follows:

Sahithi Vankayalapati: Responsible for building the logic and code for Query Likelihood Model, Stopping, Cacm_Stemming Model, Code Integration and Semantic Query Expansion using Thesauri/ontologies.

Srinivasa Vijay Bhaskar Vemuri: Responsible for building the logic and code for Snippet Generation, Query Term Highlighting and implementing the query interface that is tolerant of spelling errors in the query terms.

Karan Singh: Responsible for building the logic and code for performing the Evaluation of our Retrieval Systems in terms of their effectiveness using 4 different evaluation measures.

All 3 of us together have then integrated and refined our old assignment's code as well as the new tasks that we executed for the project and we also made the system's design object oriented so that code redundancy is eliminated Documentation we divided the work so that each writes about things they implemented.

2. Literature and resources

2.1 Indexing

For our Information Retrieval System we have built a unigram indexer for the given corpus which creates an inverted index of the unigrams.

2.2 BM25 Retrieval System

BM stands for Best Matching and it is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. The algorithm extends the scoring function for the binary independence model to include document and query term weights.

2.3 tf-idf Retrieval System

Term frequency- inverse document frequency is numerical statistic that indicates the importance of a word to a document in the corpus. Term frequency is the frequency of a term in a document and idf is N/df where N number of documents and df is document frequency.

2.4 Lucene Retrieval System

We used Apache Lucene as one of our base run models. It is an open source information retrieval system software library which provide indexing and search capabilities.

2.5 Query Likelihood Model (Dirichlet Smoothing)

The query likelihood model is a language model used in information retrieval. It is possible to rank each document by the probability of specific documents given a query. This is interpreted as being the likelihood of a document being relevant given a query.

2.6 Pseudo Relevance Feedback

Pseudo relevance feedback , also known as blind relevance feedback , provides a method for automatic local analysis. The method is to do normal retrieval to find an initial set of most relevant documents, to then assume that the top k ranked documents are relevant, and finally to do relevance feedback as before under this assumption.

2.7 Semantic Query Expansion

Query expansion involves evaluating a user's input and expanding the search query to match additional documents. This includes adding semantically related words like synonyms to the query.

2.8 Snippet Generation and Query Term highlighting

Provides Query-dependent document summary using a significance factor for each sentence using the Luhn's algorithm. From the snippets, we have highlighted the query terms by performing a linear scan as the snippets are usually small.

3. Implementation and discussion

3.1 Cleaning

We implemented a **Cleaner** class, which has all the related methods.

For cleaning we used Beautifulsoup to get the text from the html files. We then removed redundant spaces and gave the option to remove punctuation and case folding. By default, both will be removed. To remove the numbers at the end, we have split the document and read it in reverse till we found a string that contains "AM" or "PM" or "am" or "pm", and then reversed and saved the content in the corpus folder.We have also cleaned the queries.

3.2 Indexing

As part of indexing we created **Indexer** class which has all the indexing related methods.

We have created unigrams to the corpus generated by cleaner. We wrote the inverted index and total number of words in each document. Instead of running the whole implementation every time, we have used the generated inverted index for further operations.

3.3. Rankings:

3.3.1: TF-IDF:

We implemented a **TF_IDF** class, which has all the related methods.

We implemented both log normalization form and the basic form of tf-idf and noticed that there is not much difference between the results of both, the reason being that cacm is a small and simple corpus and thus we used the basic form for our implementation.

3.3.2 Query Likelihood Model (Dirichlet Smoothed) :

We implemented a **DMSmoothing** class, which has all the related methods. This is our second Retrieval Model, we have implemented this using mu value as 2000, |C| is the frequency of corpus, computed using the counter dictionary that has total words in each document. Cqi is the query frequency in whole corpus, and f is the frequency of term in document. |D| is document length, which is addition to JM smoothing and more relevant. We have experimented with different values of mu and thought that 2000 was more relevant. We noticed that small values of mu gives more importance to relative weighting of words and large values favour the number of matching terms.

3.3.3 BM25:

We implemented a **BM25** class, which has all the related methods. For this model, we have used N as 3204, the total size of cacm corpus. The k1 as 1.2 , k2 as 100 and b as 0.75. We have reused the code and wrote the files in descending order based on the document score. As the query frequency is also taken into consideration here with a linear increase of term frequency in document, this has yielded us a good MAP and MRR value.

3.3.4 Lucene (In Java):

As we already had the lucene model inbuilt, we indexed our cacm corpus and added all the queries in a single file, which our code will convert to a map and then run and compute score using inbuilt TopScoreDocCollector.

3.4 Query Enhancement:

3.4.1 Pseudo relevance feedback:

We implemented a **PseudoRelevance** class, which has all the related methods. We chose to work on bm25 model as it yielded better scores of MAP.

For pseudo relevance feedback, before query expansion we have removed all the stop words, so that we won't expand query on common words, we also made sure not to add the same words as in query.

We also performed Query Expansion on our QLM model but we saw that query expansion on BM25 yielded better results than QLM.

3.4.1 Semantic Query Expansion:

We implemented **SemanticQueryExpansion** class, which has all the related methods. We have used Wordnet of nltk library to get the semantics(synonyms and antonyms) of a word. We have removed common words to get any synonyms. And appended the results to the query. We have implemented this on Dirichlet Smoothing and BM25 to compare the models as well.

3.5 Stopping

Stopper class has all the related methods. We will perform stopping using common_words .txt, we added all these words to a list and then rebuilt a new inverted index by not including stop words in inverted index, we also didn't add then to counts dictionary which had document and total number of words in the document. We performed this on BM25 and it had yielded an improved MAP and MRR scores.

3.6 Indexing Stemmed Version

We performed this method using the inbuilt cacm.stem corpus and queries, we cleaned the corpus using the Cleaner class and all the implementation. We worked with top three stemmed queries as they seemed to be complex and average in length. We used Indexer class from before to create inverted index. We reused all the code for performing this stemmed corpus and queries.

3.7 Snippet Generation and Query term highlighting

We have ranked each sentence in a document using a significance factor(using Luhn's law) and selected the sentences with the highest significance factor for the summary. Significance factor for a sentence is calculated based on the occurrence of significant words. Text is bracketed by significant words and significance factor for bracketed text spans is computed by dividing the square of the number of significant words in the span by the total number of words.

Within the snippets, we have highlighted the words which are query terms.

3.8 EVALUATION

The metrics used to measure effectiveness is as follows:

3.8.1 Precision and Recall

Precision is the proportion of the retrieved documents that are relevant and Recall is the proportion of relevant documents that are retrieved. The implementation is as follows:

We stored the relevant documents and retrieved documents in 2 dictionaries, we then calculate precision using retrieved_document_count and relevant_document_count. Once this done we then calculate recall using the maintained dictionaries.

3.8.2 MAP

To summarize the effectiveness of rankings from multiple queries, the average of the precision values from each ranking is obtained as an effectiveness measure which is the Mean Average Precision and is calculated as :

$$\mathbf{MAP} = \frac{\sum_{q=1}^{Q} \mathbf{AveP(q)}}{Q}$$

Where Q is the number of queries, AveP(q) is the average precision score for the qth query.

3.8.3 MRR

It is computed as the average of reciprocal ranks over a set of queries and is calculated using the formula:

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}.$$

Where Q is the number of queries, rank_i refers to the rank position of the first relevant document for the i-th query.

3.9 Query-by-query Analysis:

We will be doing a query by query analysis for QLM(Dirichlet Smoothing) and BM25 Retrieval Models, the following are the queries:

Stemmed queries/ Normal Queries:

1. parallel algorithm / parallel algorithms
2. appli stochast process / applied stochastic processes
3. distribut comput structur and algorithm / distributed computing structures and algorithms

For BM25 Retrieval Model(Stemmed vs Non-Stemmed Version):

Query 1:

This query has 16 common in the top 20 Documents for the above models. The high similarity is because we can see that one of the terms i.e parallel is same for both the versions of the query.

Query 2:

This query has 9 common in the top 20 Documents for the above models. The less similarity is because we can see that all these words have been stemmed to their root stem word, applied-->appli, stochastic -->stochast and processes-->process in the stemmed version of the query.

Query 3:

This query has 4 documents in common in the top 20 documents for the above models. The least similarity is because the query is long and all the terms except “and” have been stemmed to their root stem word in the stemmed version of the query.

It can be concluded that in all the three queries, stemming has the most amount of impact on this query as it is long and all the terms are also stemmed which causes most of the documents in the top 20 ranked documents to be different.

For QLM Retrieval Modell(Stemmed vs Non-Stemmed Version):

Query 1:

This query has 17 documents in common in the top 20 documents for the above models. The high similarity is because the query is short and has 1 term in common in both the versions of the query.

Query 2:

This query has 12 documents in common in the top 20 documents for the above models. We can say that this models reacts better to stemming as compared to the BM25 Model.

Query 3:

This query has 8 documents in common in the top 20 documents for the above models. The less similarity is because the query is long and all terms have been stemmed to their root stem word in the stemmed version of the query. We can also say that this model reacts better to stemming as compared to the BM25 Model.

We will now be doing analysis between 2 Retrieval Models for the following non stemmed queries:

For Tf_idf vs BM25 Base Runs:

Query Analyzed 1: What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?

This query has 7 common documents in the top 20 ranked documents. We can see that there are less documents in common because tf-idf is a very naive model whereas BM25 takes query relevance as well into consideration.

Query Analyzed 2: parallel algorithms

This query has 10 common documents in the top 20 ranked documents. We can see that there are more documents in common as compared to the previous query because this query is a very small query and thus the relevance of the query does not matter as much as it matters for longer queries.

Query Analyzed 3: Articles about the sensitivity of the eigenvalue decomposition of real matrices, in particular, zero-one matrices. I'm especially interested in the separation of eigenspaces corresponding to distinct eigenvalues. Articles on the subject: C. Davis and W. Kahn, "The rotation of eigenvectors by a permutation:, SIAM J. Numerical Analysis, vol. 7, no. 1 (1970); G. Stewart, "Error bounds for approximate invariant subspaces of closed linear operators", SIAM J. Numerical Analysis., Vol. 8, no. 4 (1971).

This query has 6 common documents in the top 20 ranked documents. We can see that this query has the least common documents as compared to the other queries because this is the longest query and tf-idf is naive and doesn’t take query relevance into consideration whereas BM25 does.

For PRF BM25 and SQE BM25

Query 1:

This query has 12 common documents in the top 20 documents. We can see that this is a mid size query and for both the runs we have performed query expansion just with different techniques and hence there are so many documents in common. We have some not common documents as well because this is a mid size query and PRF is a naive way of query expansion whereas SQE is an efficient way.

Query2:

This query has 15 common documents in the top 20 documents. We can see that this is a very small query and thus there are so many documents in common because we have performed query expansion for both the runs, it’s just with different techniques, this query has more documents in common as compared to the previous query because this is a small query.

Query 3:

This query has 18 common documents in the top 20 documents. We can see that this is a huge query and on the top of that we are expanding the query and hence our models do not perform efficiently for such long queries and thus giving too many documents in common because it is a very small corpus.

For BM25 Stopped and tf-idf stopped and expansion

Query 1:

This query has only 6 common documents in the top 20 documents because we are now comparing the most effective model with the least effective model and hence there are very less documents in common which is justified.

Query 2:

This query has only 7 common documents in the top 20 documents because of the same reason as stated above. We see 1 more document in common because this is a small query and the scope of documents for a small query is smaller and thus more documents in common as compared to the above query.

Query 3:

This query has 12 common documents in the top 20 ranked documents. The reason behind this is the same as the comparison for this query in the above models. The query is too huge and our models do not give effective results for such huge queries since the corpus size is also very small.

4. Results

Task	Model	MAP	MRR
1 - Base runs			
1.1	TF-IDF	0.286	0.501
1.2	Query Likelihood Model -Dirichlet Smoothing	0.406	0.689
1.3	BM25	0.439	0.721
1.4	Lucene	0.398	0.670
2 - Pseudo Relevance, Semantic Query Expansion, Stopping.			
2.1	Pseudo Relevance Feedback - BM25	0.399	0.634
2.2	Pseudo Relevance Feedback - Dirichlet Smoothing	0.280	0.442
2.3	Semantic Query Expansion - BM25	0.446	0.725
2.4	Semantic Query Expansion - Dirichlet Smoothing	0.409	0.689
2.5	Stopping - BM25	0.464	0.724
2.6	Stopping - Dirichlet Smoothing	0.402	0.690
3 - Stopping and Query Expansion			
3.1	Stopped and Pseudo Relevance on TF-IDF model	0.236	0.405
3.2	BM 25 Model Stopped and semantic query expansion	0.421	0.644

5. Conclusions

In the Results table above we can see that the MAP values are in the following order for the base runs: BM25 > QLM > Lucene > tf-idf

The MRR values are in the following order for the base runs: BM25 > QLM > Lucene > tf-idf

We can clearly see that for the Base Runs the BM25 performs the best among all the models and the tf-idf performs the least. We can also see that there is a very little difference between the BM25 and the QLM models and we feel that sometimes the QLM model can also outperform the BM25 if the parameters are tuned in a more efficient manner with respect to the queries and the corpus.

For Query Expansion, Stopping and Stemming the MAP values are in the following order:

For Psuedo Relevance Feedback: We can see that BM25 is greater than QLM for just PRF but BM25 with PRF is less than BM25 base run which is because the PRF query expansion method is a blind technique which simply takes the top n terms from the top k documents to expand the query and thus the expansion queries might not always be relevant. Similarly we can see that QLM with PRF is less than the base run QLM for the very same reason. Although we can improve the MAP value of PRF by using Dice's coefficient instead of the naive approach and this can also be one of the extensions to our Project.

For Semantic Query Expansion: We can see that BM25 is greater than QLM and they are also greater than the base run values for BM25 and QLM which tells us that Semantic query expansion works better with our models and hence we can also say that the expansion terms are relevant to the queries when we perform Semantic Query Expansion. One of the interesting insights that we saw while performing Semantic Query Expansion was that initially when we used Python's Thesaurus library for generating synonyms, the MAP values were lesser than that of the base run but as soon as we used Word Net and implemented a couple of advancements with the synonyms generated via Word Net we saw that it yielded better MAP Values as compared to the Base Run.

For Stopping: We can see that BM25 is still higher than QLM and also higher than the base run which tells us that stopping had a positive impact on the retrieval model and resulted in increasing the MAP Values.

For Stopping and query expansion: We performed both stopping and query expansion, in one run we coupled stopping with PRF query expansion on our tf-idf Retrieval Model and in the other run we coupled stopping with Semantic query expansion on our BM25 Model because it was the best performing model. The MAP values that we got for this run was a little less than the base run as well as the Semantic Query Expansion Run, one of the possible reasons for this could be that since we had some of the stop words in the queries as well and we ended up removing those words from our corpus. As a result of which the matching of the query terms with the terms in the corpus relatively reduced as compared to before resulting in a score lesser than the Base run and then Semantic Query Expansion run but it still seems to be very close to those and thus appropriate.

Finally we can say that for the given queries and the corpus, the BM25 Retrieval Model was the best overall in terms of MAP values as well as the MRR values and the best run was when we applied Stopping on BM25 and Semantic Query Expansion on BM25.

5.1 Outlook

Our Project incorporates 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:

1. Multithreading: The response time of the System could be drastically improved using multiple threads.
2. Query Logs: Query logs can improve the performance of the Retrieval System as we have the record of all the historical queries.
3. Compression techniques: As the corpus grows, it the use of different compression techniques like 'v-byte encoding' and 'Elias-Y encoding' to store the inverted index enhances space efficiency.

6. Extra Credit: Spell Checker

We have designed a query interface that is tolerant of spelling errors in the query terms. The model is based on a combination of the noisy channel approach and the notion that all errors within the same edit distance are equiprobable.

In our implementation, we have broken the query into its respective terms and if the given word is not in the corpus, it is a potential error term. We found the possible correct terms for each error term and framed sentences using the corrected words. We have ranked these sentences based on the sum of the probability of each word in each of the corrected sentence and suggested the top 6 out of them.

As discussed in section 5.1, using the Query Logs, we can give higher preference to the queries which were selected by the user in the history and add some weight to such sentences while ranking the suggestions.

7. Bibliography

[1] "Search Engines: Information Retrieval in Practice," by W. Bruce Croft, Donald Metzler, and Trevor Strohman.

[2] Introduction to Information Retrieval by Christopher D. Manning, Hinrich Schütze, and Prabhakar Raghavan

[3] Simple Python Spell-Checker by Peter Norvig https://norvig.com/spell-correct.html