Information Retrieval System

CS 6200: Information Retrieval

Northeastern University

Spring 2018, Prof. Nada Naji

TEAM MEMBERS

Shubham Rastogi

Abhidipto Sengupta

Priya Ranjan Panda

Table of Contents

1. Introduction:

1.1 Overview:

The goal of the project is to design and build our own Information Retrieval systems using the core concepts of Information Retrieval. We also need to evaluate and compare their performance in terms of retrieval effectiveness.

Four distinct retrieval models have been used to design our search engines which are as follows: -

- BM25 Retrieval Model
- TF-IDF Retrieval Model
- Smoothed Query Likelihood Model
- Lucene’s default Retrieval Model

We then reported top 100 retrieved ranked lists (one list per run per retrieval system) and use the ranked list generated by BM25 retrieval model and pseudo relevance feedback on top of it using modified version of Rocchio's algorithm to perform query enrichment.

We also performed stopping and stemming text transformation techniques on the given CASM corpus and then use the BM25, TF-IDF and Query Likelihood retrieval models to generate top 100 ranked documents.

Snippet generation and query term highlighting mechanism was also implemented in this project

Performance analysis in terms of retrieval effectiveness measures of various runs on retrieval models was also performed by computing the following measures:

- Mean Average Precision
- Mean Reciprocal Rank
- Precision @ rank K where K = 5 and K = 20
- Precision and Recall

1.2 Contribution of team members:

- Shubham Rastogi was responsible for implementing the mentioned retrieval models (BM25, TFIDF, Smoothed Query Likelihood Model) . He was also responsible for implementing various performance evaluation measures. He also implemented the synthetic spelling error generator. He also responsible for performing text transformation technique i.e. stopping.

- Priya Ranjan Panda was responsible for implementing query enrichment, Lucene retrieval model. He was also responsible for performing various baseline runs analysis reports. He also responsible for performing text transformation technique i.e. stemming.

- Abhidipto Sengupta was responsible for implementing snippet generation and query highlighting mechanism . He was also responsible for implementing soft -matching query handler and analysis report for the above implementation.

  All three were involved in documentation and generation of various analysis reports.

1. Literature and resources:

1.1 Overview:

<u>Phase 1 :: Indexing and Retrieval</u>

<u>Task 1 – Retrieval Models</u>

BM25 – It is a ranking function which is used by many search engines where it ranks many documents according to their relevance for the given query. Here for the purpose of the project we have used "cacm.rel" to find the relevance judgements. The constants used in the formula used by BM25 is according to TREC standards.

Lucene – It is a high-performance, full-featured text search engine library which has been used to index and rank documents. We have used a Lucene 4.7.2 version for the purpose of the project.

TF-IDF – Here we calculate ranking by computing the normalized value of term frequency multiplied by inverse document frequency.

Smoothed Query Likelihood Model - Here, we have used Jelinek-Mercer smoothing with a constant value of 0.35.

Indexer – It collects data. Parses and stores data for fast information retrieval. For the purpose of project, we have used Unigram Indexer which stores inverted index in the form "Term : [DocID : TF]" where Term is the word in the document, DocID is the name of the document and TF is the frequency of the term in the document.

<u>Task 2 – Query Enrichment</u>

We implemented a modified version of Rocchio's algorithm to implement the Pseudo Relevance Feedback for the query enrichment.

<u>Task 3 – Retrieval Runs on text transformed corpus (Stopping and Stemming)</u>

Stop words are functional words that occur frequently and do not provide any meaning to the context of the document. To improve the performance of a search engine, we filter the stop words . For the purpose of this task, we ignored the words mentioned in "common_words.txt" while indexing.

Stemming is the process of reducing derived words to their word stem. For the purpose of this task, we have used "cacm_stem.txt" as textual corpus and "cacm_stem.query" as a list of stemmed queries.

<u>Phase 2 :: Displaying Results</u>

For displaying the results, we implemented snippet generation and query term highlighting technique. For snippet generation technique, we followed Luhn's approach for selecting significant words for sentence selection for snippets. We implemented various methods for query term highlighting in snippets.

<u>Phase 3 :: Evaluation</u>

We used the following measures to assess the performance of our retrieval systems in terms of effectiveness.

1. MAP
2. MRR
3. Precision
4. Recall
5. P@K

<u>Extra Credit:</u>

<u>Task 1 – Synthetic Spelling Error Generator:</u>

For implementing a synthetic spelling error generator, we sorted the terms of the query based on their lengths and then filtered the top 40 % of the terms to induce error by shuffling the non-boundary characters. A baseline run of retrieval model (BM25) was performed on the error induced queries and based on the top 100 ranked retrieved documents, the various performance measures were computed to measure the effectiveness of the model. The performance result was compared with the previous obtained performance results i.e. the result obtained from original.

<u>Task 2 –Soft Query-Matching Handler:</u>

For implementing a soft query matching handler, we used the error model and language model to obtain various correct suggestions for each erroneous query terms, from which the best suggestions were selected to replace the erroneous terms in the queries. A baseline run of retrieval model (BM25) was performed on the corrected queries and based on the top 100 ranked retrieved documents, the various performance measures were computed to measure the effectiveness of the model. The performance result was compared with the previous obtained performance results i.e. the result obtained from original and error induced queries.

1.2 Third Party Tools:

The following third-party tools were used in implementing this project:

<u>Lucene</u>: We used the following libraries for Lucene:

- lucene-core-VERSION.jar
- lucene-queryparser-VERSION.jar
- lucene-analyzers-common-VERSION.jar

Various python libraries were used, those are as follows:

<u>Beautiful SOUP</u>: It is a Python package which has been used for parsing the documents.

LXML – It is a Python library used for parsing xml documents.

1.3 Research Article References

http://www.cs.ucr.edu/~vagelis/classes/CS172/publications/jasistSalton1990.pdf
https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

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 BM-25 Model – BM 25 is one of the most the most effective and popular ranking algorithms which is based on binary independence model. It extends the scoring function for the binary independence model to include document and query term weights which is based on probabilistic arguments and experimental validation.

It uses the following formula

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

Where,

r_i – the number of relevant documents containing the term i
n_i – the number of documents containing the term i
N – the total number of documents in the corpus
R – the total number of relevant documents for the query
f_i – frequency of term i in the document
qf_i – frequency of term i in the query
$k1$ – the value is 1.2
$k2$ – the value is 100
K - $k1((1 - b) + b * \frac{dl}{avdl})$. It normalizes the tf component by document length where b is a parameter, dl is the length of a document and $avdl$ is the average length of a document in the collection.
b – the value is 0.75

We sort the documents in non-increasing order of their BM25 scores. The top 100 documents are printed in the file "Top_100_Query_Result_BM25.txt" in the following pattern -
$\boldsymbol{query\_id\ \ Q0\ \ doc\_id\ \ rank\ BM25\_score\ \ BM25}$

3.1.2 Query Likelihood Model

In Query likelihood retrieval model, we rank documents by the probability that the query text could be generated by the document language model. In simple words, we calculate the probability that we could pull the query words out of the bucket of words representing the document. The below mentioned formula calculates it.

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

Where
λ – 0.35 as per TREC standard
D – number of words the document
C – number of words in the corpus
f_{q_i} – number of times word q occurs in document i
c_{q_i} – number of times word q occurs in corpus i

We sort the documents in non-increasing order of their query likelihood retrieval scores. The top 100 documents are printed in the file “Top_100_Query_Result_QueryLikelihoodModel.txt” in the following pattern -
query_id Q0 doc_id rank Query_Likelihood_score Smoothed Query Likelihood Model

3.1.3 TF-IDF Model –

TF-IDF is one of the most popular term weighting schemes which are used by search engines as a tool scoring and ranking a document’s relevance given a user query. We multiplied normalized tf with idf to get the tf-idf measure.

$$tf \ - \frac{\text{number of term occurrences}}{\text{total number of terms in that document}}$$

idf – calculated by adding 1 to the log of total number of documents divided by 1 + number of documents in which the term appears. To prevent it to become 0,1 is added.

We sort the documents in non-increasing order of their tf-idf scores. The top 100 documents are printed in the file “Top_100_Query_Result_TF-IDF.txt” in the following pattern -
query_id Q0 doc_id rank TF − IDF_score TF − IDF

3.1.4 Lucene –

Lucene scores the documents by using a combination of Vector space model and the Boolean model to predict the relevance of the given query to the given document.

The idea behind the Vector Space model is that the more number of time the query appear in a document relative to the number of times the query term appears in all the documents in the collection, the more relevant is the document for that particular query.

The idea behind the Boolean model is to first narrow down the documents that needs to be scored based on the use of Boolean logic in the Query specification.

Scoring is dependent on the way the documents are indexed.

We sort the documents in non-increasing order of their scores. The top 100 documents are printed in the file “Top_100_Query_Result_Lucene.txt” in the following pattern -
query_id Q0 doc_id rank Lucene_score Lucene.

3.2 Query Enrichment Using Pseudo Relevance Feedback

We implemented the Pseudo Relevance Feedback by using Rocchio's algorithm. This algorithm modifies the initial weight of the query vector to produce an expanded query by maximizing the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents.

The formula used is as follows –

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

q_j – initial weight of the query term j
Rel – set of identified relevant documents
$Nonrel$ – set of non-relevant documents
|.| - gives the size of the set
d_{ij} – is the weight of the j^{th} term in document i
α, β, γ – are the parameters that control the effect of each document

For our implementation, we have chosen the value of
$\alpha -$ 1.0
$\beta -$ 0.75
$\gamma -$ 0.15

To increase the contribution by the relevant documents , the parameter β was chosen to be higher and the parameter γ was chosen to be smallest of three in order to reduce the contribution made by non-relevant documents.

We took this values on the basis of experiments done and described in the article "*Improving Retrieval Performance by Relevance Feedback." Gerard Salton; Chris Buckley. Journal of the American Society for Information Science (1986-1998); Jun 1990.*"

Following are the steps of implementation –

- BM25 model was used to retrieve the results from the initial query.
- We took top "m" documents where m was chosen to be 10 to be included in the relevant set of documents.
- We then applied the Rocchio's algorithm and retrieved the top 20 terms with highest weights which was then appended to the query if they were not already present.
- The BM25 model then retrieved new documents with the modified query

3.3 Snippet Generation and Query Term Highlighting

We used Luhn's approach for snippet generation and made some modification in the process of choosing significant words. To improve the process of Snippet Generation, stop words were filtered out. Also, the term was selected if it satisfied Luhn's approach or was a part of the original query. The following approach was used.

We calculated the significant words by using the following frequency-based criterion

- If $f_{d,w}$ is the frequency of the word w in document d, then w is a significant word if it is not a stop word, and

$$f_{d,w} \geq \begin{cases} 7 - 0.1 \times (25 - s_d), & \text{if } s_d < 25 \\ 7, & \text{if } 25 \leq s_d \leq 40 \\ 7 + 0.1 \times (s_d - 40), & \text{otherwise,} \end{cases}$$

  Where s_d is the number of sentences in the document d.

- The significant words were selected if it satisfied the above equation or was a part of the query.
- We generated a span of words from the first occurrence of a significant word to the last occurrence of the significant word in a sentence and the sentence significance score was calculated by squaring the number of significant words in the span divided by the length of the span.
- We then sorted the sentences in non-increasing order of their scores and displayed the top 4 sentences per document and also highlight the query terms present in the snippet which were not stop-words.

3.4 Evaluation

Evaluation of corpus is essential as it assess the performance of our retrieval systems in terms of effectiveness. We used measures such as Precision, Recall, Mean Average Precision(MAP), Mean Reciprocal Rank(MRR), Precision @ rank K where K = 5 and K = 20 to measure the effectiveness of the retrieval models that we implemented.

Precision – It is the ratio of number of relevant documents that were retrieved for a given query to the number of retrieved documents.

$$\text{Precision} = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

Recall - It is the ratio of number of relevant documents that were retrieved for a given query to the number of relevant documents for the given query.

$$\text{Recall} = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

Mean Average Precision (MAP) – It provides a brief summary of the effectiveness of the ranking algorithm over many queries. It is defined as the mean of all the average precisions for all the queries. Average Precision of a query is the sum of precision values when relevant document is found to the total number of relevant documents for that query.

Mean Reciprocal Rank (MRR) – It is defined as the mean of all the reciprocal ranks for the queries. Reciprocal rank is defined as the rank at which first relevant document is found.

Precision @ rank K where K = 5 and K = 20 – Precision @5 and Precision @20 assess the performance of a search engine. It indicates the number of relevant documents that have been retrieved at higher ranks. The Information Retrieval System is said to be highly efficient if it shows high precision at higher ranks which is proportional to the number of relevant documents retrieved.

3.5 Query by Query Analysis

Queries that have been selected for this analysis are

- appli stochast process(Stemmed) / Applied stochastic processes(Unstemmed)
- distribut comput structur and algorithm(Stemmed) / Distributed computing structures and algorithms(Unstemmed)
- code optim for space effici (Stemmed) / code optimization for space efficiency (Unstemmed)
- portabl oper system(Stemmed) / portable operating systems(Unstemmed)

The detailed report is mentioned in the attached excel sheet.

QueryByQueryAnaly
sis(StemmedVsUnste

3.6 Extra Credit

Task 1:

For implementing a synthetic spelling error generator, we sorted the terms of the query based on their lengths and then filtered the top 40 % of the terms to induce error by shuffling the non-boundary characters. A baseline run of retrieval model (BM25) was performed on the error induced queries and based on the top 100 ranked retrieved documents, the various performance measures were computed to measure the effectiveness of the model. The performance result was compared with the previous obtained performance results i.e. the result obtained from original.

Task 2:

For implementing a soft query matching handler, we used the error model and language model to obtain various correct suggestions for each erroneous query terms, from which the best suggestions were selected to replace the erroneous terms in the queries. A baseline run of retrieval model (BM25) was performed on the corrected queries and based on the top 100 ranked retrieved documents, the various performance measures were computed to measure the effectiveness of the model. The performance result was compared with the previous obtained performance results i.e. the result obtained from original and error induced queries.

pdf

ExtraCredit_Report(T
ask1&2).pdf

The detailed report for task 1 and task 2 is mentioned in the attached report.
Note : The detailed is also present in the deliverables.

4 Results

All the baseline runs along with stopping, stemming, pseudo relevance feedback and retrieval effectiveness measures results are saved in the below attached file named " Evaluation.docx"

Evaluation.xlsx

5 Conclusions and Outlook

5.1 Conclusions:

1. Among all the 4 runs, SQLM performed most poorly when compared to other models based on MAP and MRR. This was performed on a corpus with no text transformation.

2. Based on MAP and MRR, BM25 gave the best results. This was performed on a corpus with no text transformation

3. Lucene performs as well as BM25 on a corpus with no text transformation.

4.The order of performance in case of stopped queries are similar to the above conclusions.

5.The document fetched with BM25 with enhanced queries (enhancement done by Pseudo Relevance Feedback) reduced the performance in terms of MAP and MRR. This might be possible because of incorrect consideration relevant documents.

5.2 Outlook

- Retrieval Efficiency can be considered to build our search engine along with retrieval effectiveness.
- We can use different compression techniques such as $v - byte\ encoding, Elias - \gamma\ encoding.$
- Proper user interface will make the search engine more user friendly and easy to use.
- Tags inside semantic HTML pages can be taken into account for better snippet generation by taking into account the texts inside various relevant and important tags.
- System can be improved to incorporate longer queries.
- The search engine's results could be improved if query logs and relevance feedback can be collected form the user.

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://www.cs.ucr.edu/~vagelis/classes/CS172/publications/jasistSalton1990.pdf
https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

6.3 Websites

https://en.wikipedia.org/wiki/Stemming
https://en.wikipedia.org/wiki/Search_engine_indexing
https://en.wikipedia.org/wiki/Apache_Lucene
https://en.wikipedia.org/wiki/Okapi_BM25
https://en.wikipedia.org/wiki/Information_retrieval#Mean_average_precision
https://en.wikipedia.org/?title=Mean_average_precision&redirect=no
https://en.wikipedia.org/wiki/Tf%E2%80%93idf
https://en.wikipedia.org/wiki/Beautiful_Soup_(HTML_parser)
https://en.wikipedia.org/wiki/BM25
https://en.wikipedia.org/wiki/Mean_reciprocal_rank
https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model-1.html
http://www.pythonforbeginners.com/beautifulsoup/
https://dl.acm.org/citation.cfm?id=1165776
https://lucene.apache.org/core/
https://lucene.apache.org/core/3_5_0/scoring.html
https://www.crummy.com/software/BeautifulSoup/bs3/documentation.html
https://www.python.org/downloads/