Search Engine Development

CS6200: Information Retrieval
Northeastern University, Fall 2017
Prof. Nada Naji

Gaurav N. Gandhi

Hardik R. Shah

Paarth J. Kotak

Contents

1. Introduction

1.1. Project Overview

The aim of the project is to develop, design and evaluate our retrieval systems in terms of relative effectiveness. Four distinct retrieval models were developed:

- BM-25 retrieval model
- Tf-Idf retrieval model
- Query Likelihood model
- Lucene (open-sourced)

Techniques used to run the retrieval models:

- Baseline run: BM-25, Tf-Idf, Query Likelihood model, Lucene
- stopping technique: BM-25, Tf-Idf, Query Likelihood model
- stemming technique: BM-25, Tf-Idf, Query Likelihood model
- pseudo relevance feedback run: BM-25

The BM-25 model, Tf-Idf model, Query Likelihood model were run with the purpose of generating top 100 ranked documents each.

Performance Assessment Techniques:

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

1.2. Member Contributions

Paarth Kotak: Paarth was responsible for implementing Tf-Idf algorithm. He also wrote the query file parsing logic. Additionally, he modified the Lucene code to make it compatible with the data structures used in the project. Moreover, he wrote the literature of the documentation and provided valuable inputs while performing query-by-query analysis and deducing results and conclusions.

Gaurav Gandhi: Gaurav was responsible for BM25 algorithm and played a vital role in maintaining the structure of the project by designing data structures wherever it was necessary and split common functions to reduce code redundancy. He also developed the evaluation measures like MAP, MRR, P@K, Precision & Recall. Additionally, he implemented excel integration into Java to output the results using external libraries.

Hardik Shah: Hardik has primarily worked on implementing Query Likelihood model. He also worked on snippet generation and incorporated various techniques to parse and display snippets using HTML tags from within the code. Additionally, Hardik has provided inputs on query-by-query analysis and contributed to the results and conclusions with his findings while implementing parts of the project.

2. Literature and resources:

2.1 Brief Terminology Explanation

- BM-25 Model: For BM25 model, a set of relevant documents was used. The values of 'K' = 0, 'k1' = 1.2 and 'k2'= 100 are chosen as per TREC standards.
- tf-idf measure: $\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} * \log \frac{N}{n_t}$

  where, $f_{t,d}$ = frequency of the term r in document d

  $\sum_{t' \in d} f_{t',d}$ = total number of documents that contain the term

  N = Total number of documents

  n_t = documents containing the term
- Lucene: The Lucene open sourced code with modifications was used to perform indexing and search operations.
- Stopping: A standard common stop word list is used to perform stopping. Any word occurring in the common stop word list would not be indexed.
- Stemming: The stemmed version of the corpus provided, needed pre-processing to remove tags and extra spaces and standardize each document.
- 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 = $\sum \frac{\text{Average Precision}}{\text{Number of Queries}}$
- 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: Snippet generation involved selecting sentences and ranking them using a significance factor. The words found the in the snipped and the query are bolded to reassure the user of the results produced.

2.2 External Libraries Used

- JSOUP: JSOUP is a standard java library to parse documents.
- Apache POI : It is a standard java library to write results to an excel file
- Lucene Libraries:
  - ➢ lucene-core-7.1.0.jar
  - ➢ lucene-queryparser-7.1.0.jar
  - ➢ lucene-analyzers-common-7.1.0.jar

3. Implementation and Discussion

3.1 Tf-Idf measure

$$\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \ * \ 1 + \log \frac{N}{n_t + 1}$$

where, $f_{t,d}$ = frequency of the term r in document d

$\sum_{t' \in d} f_{t',d}$ = total number of documents that contain the term

N = Total number of documents

n_t = documents containing the term

The reason we are adding 1 to the denominator is to avoid a divide by zero error incase a term does not appear in the document. Similarly, if the log of a number becomes zero then the entire tf-idf result will become zero, hence 1 is added to the idf part of the formula.

Procedure Implemented:

1) Given: Corpus and query list file.
2) Split the query into single terms
3) The implementation makes use of a structure called a **Posting** which has *docID* and *termFrequency.*
4) The posting of each term is selected and tf-idf is calculated for it.
5) Results are stored in descending order of the score and outputted in the format mentioned below: **QueryID 'Q0' DocID Rank tf-idf Score ' tf-idf Model'**

3.2 BM-25 Model

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

where,

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. The value of b is taken as 0.75 as per TREC standards.

Results are stored in descending order of the score and outputted in the format mentioned below:

QueryID 'Q0' DocID Rank BM25_Score ' BM25 Model'

3.3 Query Likelihood Model

Jelenik Mercer smoothing:

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

where, $\lambda = 0.35$

Procedure Implemented:

1) Generate the inverted index.
2) Generate the relevant document list.
3) Find the term frequency of each term in the query.
4) Using the Jelinek-Mercer Smoothing technique a cumulative likelihood score is calculated. Log is applied to solve accuracy issues that arise when dealing with small numbers.
5) Results are stored in descending order of the score and outputted in the format mentioned below: **QueryID 'Q0' DocID Rank QLM_Score ' QL Model'**

3.4 Snippet Generation

Procedure Implemented:

1) Each sentence in a document is ranked using a significance factor.
2) A window of size 'n' is selected and significant words within that window are identified. The significant words are first squared and then divided by 'n'.

$$f_{d,w} \geq \begin{cases} 1 - 0.1 \times (10 - s_d), & \text{if } s_d < 10 \\ 1, & \text{if } 10 \leq s_d \leq 25 \\ 1 + 0.1 \times (s_d - 25), & \text{otherwise} \end{cases}$$

3) Results are displayed by **bolding** the terms in the snippet which are present in the query.

3.5 Pseudo Relevance Feedback

Procedure Implemented:

1) The initial query is used for retrieval
2) A threshold value of 10 is chosen as the number of documents whose results are to be included in the relevant set of documents, remaining documents are added to the non-relevant set.
3) New Query is generated using Rocchio algorithm:
    a. A query vector with the term frequency scores is generated.
    b. Relevant document vector is generated which is basically the inverted index of the top 10 documents.
    c. Non-Relevant document vector is generated. This is done by inserting all the terms from the inverted index which are not present in the relevant document vector.

d. The query is modified by using the formula:

$$\alpha\, q_0 + \frac{\beta}{|D_r|} * \sum d_j - \frac{\gamma}{|D_{nr}|} * \sum d_j$$

where,

q0 = original query vector
Dr = set of relevant docs
Dnr = set of nonrelevant documents
α, β, and γ are weights attached to each term.

4) The top 20 most highly weighted terms which are not present in the query are appended to the original query.
5) The retrieval process is restarted using the modified query.

3.6 Query-by-Query Analysis

Query 1: "portabl oper system"

Tf.Idf vs BM25 vs Query Likelihood

The top document reported by Tf.Idf is not related to portable operating systems but is still ranked highly since the document contains high frequency of the word "system".

Top 2 results of both these models are matching. It's interesting the notice that BM25 has reported document 1680 at 3rd rank but query Likelihood model has ranked this document at 43rd position. Document 1680 is not more relevant than document 1033 which is reported at 3rd rank for Query Likelihood model.

It can be observed that the BM25 and Query Likelihood model are very similar in terms of precision, but Query Likelihood has produced the best results as compared to all the models under consideration and Tf.Idf has retrieved the worst results.

Query 2: "parallel algorithm"

Querylikelihood vs Bm25 vs Tf-idf.

The top documents given by querylikelihood occurs below 20 ranks in BM25 model.

The BM25 has ranked these documents based on number of term occurances whereas Querylikelihood ranks doucments based on relevance and term frequency and considers smoothing all over the collection.

The documents is higher ranked in BM25 because there are more ocuurances query terms in the doucments irrespective of the relevance of doucments.

In querylikelihood the number of documents are diffrent than BM25 because Querylikelihood considers smoothing all over the collection. So, eventhough some documents have higher number of terms, because of smoothing and relevance other documents are ranked higher in Querylikelihood.

Tf.idf has no documents common compared to BM25 and Querylikehood. This is because even though some terms might be higher in some documents but these terms might not occur in many documents and thus the overall rank of the document decreases.

Query 3 : "parallel processor in inform retriev"

Querylikelihood vs Bm25 vs Tf-idf.

The top documents given by querylikelihood appear scattered in BM25 model.

As explained for the previous query The BM25 has ranked these documents based on number of term occurances whereas Querylikelihood ranks doucments based on relevance and term frequency and considers smoothing all over the collection.

Tf.idf has a few documents common compared to BM25 in the top order but very few with Querylikehood. This is because even though some terms might be higher in some documents but these terms might not occur in many documents and thus the overall rank of the document decreases.

4. Results

- 4 Baseline Run Spreadsheets:
  BM25_Baseline Lucene_Baseline QueryLikelihood_Baseline TFIDF_Baseline
- Pseudo-Relevance Feedback: PseudoRelevanceFeedback
- Stopped Runs:BM25_Stopped QueryLikelihood_Stopped TFIDF_Stopped
- Stemmed Runs: BM25_Stemmed QueryLikelihood_Stemmed TFIDF_Stemmed
- Evaluations: Evaluations
- Snippet Generation: Snippets
  Note:- All the results, intermediate and final (segregated by space) are inside the Deliverables folder.

5. Conclusions and Outlook

5.1 Conclusions

We ran eight runs as specified in the problem statement and after running all eight runs and evaluation for all runs, we came to the following conclusion:

1. The best results were obtained from Query Refinement i.e pseudo relevance feedback. For Pseudo Relevance feedback, the baseline run was Query Likelihood model. The MAP was 0.3638 and MRR was 1 was Pseudo Relevance feedback. The MAP obtained was 0.033 better than query likelihood model. This change in result was because of the query expansion technique used by implementing Rocchio Algorithm.

The 2nd best model was Query Likelihood model.

2. The Lucene also performed quite well for the baseline runs. The MAP scores for Lucene was 0.05 and MRR score was 0.296. However, the BM25 and Tf.idf didn't perform up to the mark for the baseline runs.

3. The BM25 scores improves when used with stop words. However, Query Likelihood model scores decreased when used with stop words. The decrease was not that significant. Hence, Stop words can have positive and negative impact based on occurrence on relevance words.

4. Tf.idf has the worst results compared to all models. It has MAP of 0.05 and MRR of 0.29 which is not good compared to other techniques. Even for P@5 and P@20 indicates zero for many queries which indicates that zero relevant documents has been retrieved. Hence, Tf.idf is not a good model for retrieval.

5.2 Outlook

The project implements all the basic features for implementing the search engine.

However, there are some improvements that can be done to improve the results:

1. Page rank can be implemented and taken into consideration while scoring documents. page rank determines the popularity of the page by the incoming link count.

2. Various compression techniques like Elias-γ and Elias-δ and be implemented for larger indexes.

3. Query logs can also be taken into consideration as part of evaluation process. Dwell time, links clicked and user profiles are some techniques that can be used for improvement

6. Bibliography

6.1 Reference 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