Information Retrieval

Northeastern University

Spring 2019

Prof. Rukmini Vijaykumar

Team:

Tejashwini Busnur

Vasavi Venkatesh

Introduction:

The goal of the project is to design and build our own information retrieval system and evaluate and compare their performance levels in terms of retrieval effectiveness.

We have implemented the project in the following phases:

Phase 1 – Indexing and Retrieval:

Task 1:

The following retrieval models were used along with our implementation of unigram indexer:

- Tf/idf
- Query Likelihood Model (JM Smoothed)
- BM25
- Lucene's default retrieval model

Task 2:

We have picked Lucene retrieval model's base run and implemented query enhancement using the below approaches:

- Pseudo relevance feedback using Rocchio's algorithm
- Query time stemming

Task 3:

We picked the base line runs of Lucene's retrieval model and BM25 and performed:

- Stopping – removed the common words (using common_words.txt) from the original set of queries (cacm.query.txt)
- Indexed the stemmed version of the corpus (cacm_stem.txt) and retrieved results for the queries in cacm_stem.query.

Phase 2 – Displaying Results:

We implement snippet generation and query term highlighting on Lucene's baseline results. We have picked the top 5 document results of each query and snippets for each document based on the sentences which have a significance factor. We highlight the terms in the snippets, which are also present in the query.

Phase 3 – Evaluation:

Used the MAP, Precision and Recall, P@K and MRR metrics on the runs.

Extra Credits:

Implemented a query interface that is tolerant of spelling errors and suggests possible 6 replacements for the spelling errors.

Member Contributions:

Tejashwini Busnur:

1. Tf-idf
2. Query Likelihood model (JM Smoothed)

3. Lucene
4. BM25
5. Evaluation
6. Extra Credit

Vasavi Venkatesh:

1. Query expansion using Pseudo relevance Feedback
2. Query expansion using query time stemming
3. Snippet Generation
4. Baseline runs after stopping
5. Baseline run after stemming
6. Evaluation

Literature and Resources:

Tf-idf:

Term frequency (tf) = Count of index term occurrences in a document/ Count of terms in document

Inverse document frequency (idf_k) = $\log(N/n_k)$

Where N is the total number of documents and n_k is the number of documents containing the term k.

The document term weight is calculated using tf * idf

Lucene:

PyLucene 6.5.0 with its default retrieval model and standard analyzer for indexing and document collection.

JM Smoothed Query Likelihood Model:

$$p(q_i|D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

λ is a coefficient controlling the probability assigned to unseen words

$f_{qi;D}$ the number of times a query word occurs in the document

$|D|$ is the total number of words in the document.

c_{qi} is the number of times a query word occurs in the collection of documents,
and $|C|$ is the total number of word occurrences in the collection.

Multiplying many small numbers together can lead to accuracy problems so we used logarithms to turn this score into a rank-equivalent

BM25:

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

Where N – number of documents in the collection, R and r are relevance information set to zero , f_i is the frequency of the term in the document where as qf_i is the frequency of the term in the query.

Dice coefficient:

$$\frac{2 \cdot n_{ab}}{n_a + n_b}$$

Where,

n_{a-} number of documents containing the term a

$n_{b\,-}$ number of documents containing the term b

$n_{ab\,-}$ number of documents containing both the terms a and b

Rocchio's algorithm

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

Where $q_{j\,-}$ initial weight of the query term j, Rel is the set of identified relevant documents, Nonrel is the identified set of non-relevant documents., dij is weight of the jth term in the document. α ,β,and γ values are 1, 0.75 and 0.15 respectively.

Stopping:

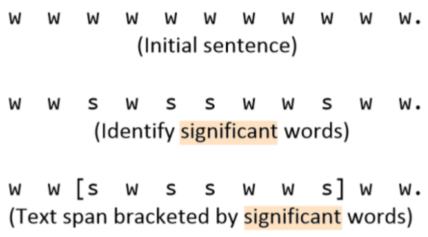
The common words from the file 'common_words' are removed from the query and the query is run for BM25 and Lucene retrieval models.

Stemming:

Created the parsed corpus from the file 'cacm_stem' and the stemmed queries from 'cacm_stem.query.txt' was run for BM25 and Lucene retrieval models.

Luhn's significance factor

The top 5 documents for each query were selected to generate snippets based on Luhn's significance factor.



A sentence is considered to be significant if it present in the query and is not a common word. A bracket is chosen in the sentence (starting and ending with a significant word) and the below formula is applied to calculate significance factor of the sentence:

Significance Factor = (Number of significant words in the bracket)^2/Total number of words in the bracket

Precision and Recall:

$$Recall = \frac{|A \cap B|}{|A|}$$
$$Precision = \frac{|A \cap B|}{|B|}$$

|A∩B| represents the set of documents that are both relevant and retrieved

|A| relevant set of documents for the query

|B| set of documents retrieved

MAP:

Mean average precision = Sum(Average precision of queries) / Number of queries

MRR:

Mean reciprocal rank = Sum (Reciprocal rank of each query) /Number of queries

Implementation and Discussion:

Retrieval Models:

The cleaned corpus (without HTML tags and punctuations) were indexed and run through each retrieval model as mentioned in the overview.

Query Expansion:

- Query time stemming:
    1. We created stem classes from the corpus using NLTK's porter stemmer.
    2. For the input query terms, we created a map of query term and the stem classes based on the above collection.
    3. Since it is not feasible to add all the stem classes of the query term to the original query, we created an association metric for each pair of words in the stem classes based on the Dice co-efficient.
    4. We remove the stem classes which have zero association metric and keep the remaining stem classes in the enhanced query.
- Pesudo Relevance Feedback:

1. We have considered the top 10 results of each query from BM25's base run as the relevant set of documents.
2. After multiple runs, we found the top 20 query terms ordered by their rocchio score were optimal suggestions and we add this to the original query and run BM25 (no relevance) model again.

Snippet Generation:

We have taken the top 5 documents for each query result for snippet generation and query term highlighting.

a. We divide the document into sentences separated by '.'
b. To find the window for calculating significant factors we find block of sentence starting and ending with a significant word.
c. A word is said to be significant if it is present in the query and is NOT a common word.
d. Based on the significant factor sorted, we display the top three significant sentences.
e. In the displayed snippets, we highlight the terms which are present in the query (ie. The significant words)
f. In case there are no significant words, we display only the title of the document.a.

Task 3 (Query by Query analysis):

1) portabl oper system:
    a. The first result retrieved by BM25 model (without relevance), 3127 is very relevant to the query since the document is exclusively about portable operating system. There are not a lot of stem classes for portable in the corpus and hence the results were relevant. The first 10 results gave mixed results. Due to the presence of the term 'oper' which is a stem word for many classes the results produced were not highly relavant
2) perform evalu and model of comput system:
    a. Similar to the above query, the first query talks about model of computer systems which is very relevant to the topic. The first three results are very relevant but due to the presence of the term 'model' 'comput' and 'system' there were some mixed results. However, overall the query seems to be having relatively higher relevance of query results.
3) Parallel algorithm:
    a. This query seems to give a mix of results which point to parallel algorithm, or a number of results related to parallel processors. However, since, not a lot of words have the stem for algorithm the query results seem relatively relevant. We noticed that when the stem has less variants the query results seem more accurate. As observed in the previous queries when the query terms is longer in a stemmed corpus the results become less relevant, Going by that logic, this query has better and relevant results.

Extra Credit (Query Interface that is tolerant of spelling errors)

A spelling error tolerant query interface has been implemented based on a 3rd party library Spell checker by Peter Novig.

Step 1: Erroneous Query Generation:

- Misspelled queries are generated using the same library and written into a file and placed under data folder.
  Erroneous queries filename : cacm.query.error.txt
- The library uses swapping of letters with edit distance for generating the misspelled words in queries.

Step 2: Compute and generate correct queries

The erroneous queries file is processed and further it generates up to 6 suggestions for each misspelled query and writes into file under results folder

Suggested queries folder : results\extra_credits\

- Inverted index is fed as dictionary to the spellchecker library.
- We have manually generated correctly spelled english words which are not part of corpus into the unindexed_words.txt. Which is also fed into the spellchecker.
- Before generating the query the suggested words are sorted based on the term frequency. Such that it helps in building the top 6 query suggestions.

Result Files:

Result	Path
Task 1 (Rank lists of retrieval model base runs)	submission\phase_1\task_1
Task 2 (Enhanced Queries and their results of Query expansion)	submission\phase_1\task_2
Task 3 (a – Stopping and b – Stemmed corpus)	submission\phase_1\task_3
Snippet Generation	submission\phase_2
Evaluation	submission\phase_3
• MAP_MRR.txt	In each folder of the retrieval model results
• Query precision recall tables	
• P@K=5	
• P@K=20	

Conclusion:

Retrieval Model	MAP	MRR
BM 25 (No relevance)	0.34269624155	0.541023892196
Lucene (No relevance)	0.308719822679	0.514672567016
TF-IDF	0.208387942326	0.375238361546
JM Smoothed	0.156864456072	0.259842468074
Pseudo Relevance Feedback (BM25)	0.0235344350799	0.0305870028109

Query time Stemming (BM25)	0.295159646624	0.481619952357
BM 25 (Stopping)	0.311295362834	0.513756894001
Lucene (Stopping)	0.305284588275	0.503407392665
TF IDF (Stopping)	0.194776927976	0.340602526325

1. We have noticed that the results for BM25 gives the best results for the given corpus.
2. We expected the query expansion results to be better but on the contrary they did not give optimal results. Based on some trial and error we noticed that the more queries we add to the original query they deviate from the relevance.
3. The results with stopping did not yield optimal results since we did not alter the corpus index that is we removed stop words only from the query but still indexed stop words in the corpus. We believe the results would be significantly better if we remove stop words from the corpus too.

Bibilography

[1]Course Notes and slides of CS 6200 Spring 2019, Northeastern University.

[2] W.Bruce Croft, Donald Metzler, Trevor Strohman. Search Engines:Information Retrieval in Practice

[3] https://lucene.apache.org

[4] https://docs.python.org

[5]https://lucene.python.org