CS6200 – Information Retrieval, Spring 2017

Project Report

Team Members:

Rakesh Krishna Radhakrishnan

Siddharth Talwekar

Vivek Nair

Instructor:

Nada Naji

Introduction

The goal of this project is to build information retrieval systems using the core concepts and processes of information retrieval learnt throughout the course in this semester. The project scope also comprises of performing optimization techniques such as stopping, stemming, query expansion as well as evaluation of different systems using evaluation metrics such as Precision, Recall, MAP, MRR and P@K.

Dataset

The CACM test-collection containing 3204 raw HTML documents has been used as corpus along with 64 unprocessed queries, stemmed version of raw documents, corresponding stemmed queries, relevance information for queries and a list of most frequent words in the corpus used as the stop list.

This report describes the implementation along with the design choices made during the project implementation.

Member's Contribution

Rakesh Krishna Radhakrishnan: Brainstorming, Parsing, Indexing, (Task1) and T-Tests implementation, Evaluation

Siddharth Talwekar: Brainstorming, Query Expansion and Pseudo Relevance Feedback, Evaluation, Snippet Generation, (Task2)

Vivek Nair: Brainstorming, Stopping and Stemming, Evaluation, Snippet Generation (Task3)

Literature and Resources

Search Engines: Information Retrieval in Practice by Bruce Croft, Donald Metzler and Trevor Strohman

Retrieval Models

TF.IDF (Term frequency – inverse document frequency)

It's a statistical measure used to evaluate importance of a word in a document. This value increases proportionally to the number of times a word appears in a document but is normalized by the frequency of the word in the collection to adjust for the unimportant words appearing frequently. The formula[1] for this is given by

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}} \qquad idf_k = \log \frac{N}{n_k}$$

Where *tf* is the term frequency calculated by dividing the frequency of the term *k* over the number of terms appearing in the document and *idf* is calculated by taking the log of number of documents over the number of documents containing the term *k*. We get the value of TF.IDF by multiplying above terms.

BM25

BM25 is a popular and effective retrieval model based on the binary independence model, the formula[1] for which is

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

- *k1*, *k2* are constants
- *qfi,* frequency of term *i* in the query
- *ri*, number of relevant documents containing term *i*
- *ni,* number of documents containing term *i*
- *R,* number of relevant documents for this query
- *N,* total number of documents in the collection
- *fi,* document frequency of the term
- *K,* given by the following formula

$$K = k_1((1 - b) + b \cdot \frac{dl}{avdl})$$

- *b,* parameter
- *dl,* is the length of the document
- *avdl,* is the average length of a document in the collection

Lucene

Lucene is a popular java based search engine used for a wide range of commercial applications. An open source information retrieval library[2], Lucene is widely used in

academic and commercial search engine applications. Solr and ElasticSearch are both based on Lucene libraries. By default, Lucene uses a TF.IDF similarity for scoring.

Query Expansion

Query expansion is a technique used to add additional terms to the query which are contextually relevant. There are various methods through which we can implement query expansion like using Thesauri, Pseudo Relevance Feedback, Inflectional and/or derivational variants. In this project, we have used Pseudo Relevance Feedback, where the most frequent words barring the stop words in the top N documents are assumed to be relevant to the query. In our design, the Top 20 frequent words from the top 50 documents are added to the query and results are then retrieved with the new expanded query.

Stopping

Stopping is a technique used to remove very high frequency words from corpus/query, which are generally not topically relevant and are function words. In this project, we have performed stopping on the corpus and the queries using the given list of common words.

Stemming

Stemming is a technique of using stems of words to expand the possibility of finding the multiple variances of a single word. We have used the stemmed corpus and queries provided for the project with TF.IDF and BM25 retrieval models.

T-tests:

T-test is a statistical test used to evaluate the performance of 2 different search engines. We have used T-test to compare the effectiveness of search engine models built in this project. In T-tests, performance score of two systems is determined by the following formula[1][3]

$$t = \frac{\overline{B - A}}{\sigma_{B-A}} . \sqrt{N}$$

- A and B are the average precisions of 2 systems to be tested
- σ_{B-A} is the standard deviation of the differences.
- N is the number of queries.

Snippet Generation

A snippet is a brief summary of a document which is supposed to present a brief description of what the document contains and how it is relevant to the information seeker by highlighting the words which are significant or are topically or contextually related to the

query. We have used *Luhn's Approach*[1] for generating snippets for the top ranked documents for each query in our project.

Implementation and Discussion

We have implemented 9 variants of a search engine in total. Table 1 gives a brief summary on the different variants implemented.

Task	System	System Name	Retrieval Model	Stemming	Stopping	Query Expansion	Relevance information Used?
1	1	BM25-Baseline	BM25	No	No	No	Yes
	2	TFIDF-Baseline	TFIDF	No	No	No	NA
	3	Lucene-Baseline	Lucene	No	No	No	NA
2	4	BM25-QueryExpansion	BM25	No	Yes	Pseudo Relevance	NA
	5	TFIDF-QueryExpansion	TFIDF	No	Yes	Pseudo Relevance	No
3	6	BM25-Stopping	BM25	No	Yes	No	Yes
	7	TFIDF-Stopping	TFIDF	No	Yes	No	NA
	8	BM25-Stemming	BM25	Yes	No	No	Yes
	9	TFIDF-Stemming	TFIDF	Yes	No	No	NA

Table 1 System Description

Parsing

Parsing is the first step in which the raw HTML documents are parsed and tokenized to be used in the indexing stage. During this stage, we removed all punctuation from the text in the documents except the "-" and also preserved "." & "," if they are between numbers. Also, the tokenized text is then case-folded and saved to disk to be used in the indexing stage.

Indexing

During indexing, using the tokenized documents, we have built an inverted index which is a data structure like dictionary in python, in which a word is a key and its corresponding value is also a dictionary, in which the key is the document name in which the term appears and the value is the frequency of the word in the document. This index is then used in the retrieval models to find relevant documents for a query. This index is convenient to locate and find all documents for a term and the frequency of the term in those documents.

Retrieval Model

In this project, we have implemented three retrieval models, namely BM25, TF.IDF and Lucene. Below is the brief description of these models.

BM25

1. In BM25 implementation, we have used the inverted index created in the indexing stage in the score calculation for each query.
2. For each query, we calculate the BM25 scores for documents for each query term retrieved from the inverted index, and from the relevance information provided to us.
3. Then rank and store the documents in decreasing order of the BM25 scores and take the top 100 documents in the folder for each task, say for Task1, the results will be saved at:
   OUTPUT_FIILES/TASK1/ BM25_RESULTS
   And files will be named for each query as
   DOCUMENTS_RANKED_FOR_QUERY_<<query no>>.txt
   Where query no = is the query number.

Variations in BM25 implementations between different systems:

1. For System “BM25-Baseline”, we have not used stopping, stemming, query expansion.
2. For System “BM25-QueryExpansion”, we have used Pseudo Relevance Feedback system for query expansion. First, we do stopping at query time as well as at index creation. After performing step 2 mentioned above using the BM25-Baseline system, we perform query expansion with stopping. Then using the top ranked documents, we extract high frequency terms and add them to query. Thereafter, we again perform the BM25-Baseline system using the new expanded queries and get the results. We do not use relevance information in this system.
3. For System, BM25-Stopping, we do only stopping at index creation and at query time on BM25-Baseline system results and no query expansion, stemming.
4. For System BM25-Stemming, we do only stemming using stemmed data and stemmed queries and no stopping, query expansion.

TF.IDF

1. In TF.IDF implementation, we have used the inverted index created in the indexing stage in the TF.IDF calculation for each query.
2. For each query term, we get the details of the query term from the inverted index, then calculate TF.IDF value for each term and then rank the documents for the term in decreasing order of the TF.IDF value.

3. Take the top 100 documents and store them in the folder for each task, say for Task1, the results will be saved at:
   OUTPUT_FIILES/TASK1/ TFIDF_RESULTS
   And files will be named for each query as
   DOCUMENTS_RANKED_FOR_QUERY_<<query no>>.txt
   Where query no = is the query number.

Variations in TF.IDF implementations between different systems:

1. For System "TFIDF-Baseline", we have not used stopping, stemming, query expansion.
2. For System "TFIDF-QueryExpansion", we have used Pseudo Relevance Feedback system for query expansion. First, we do stopping at query time as well as at index creation. After performing step 2 mentioned above using the TFIDF-Baseline system, we perform query expansion with stopping. Then using the top ranked documents, we extract high frequency terms and add them to query. Thereafter, we again perform the TFIDF-Baseline system using the new expanded queries and get the results.
3. For System, TFIDF-Stopping, we do only stopping at index creation and at query time on TFIDF-Baseline system results and no query expansion, stemming.
4. For System TFIDF-Stemming, we do only stemming using stemmed data and stemmed queries and no stopping, query expansion.

Lucene

For implementation using Lucene,

1. First we have created the indexer class and used the SimpleAnalyzer for the indexer.

2. Thereafter we ran the given queries using the Lucene Queryparser and collected the top 100 documents using the TopScoreCollector class.

3. Thereafter we generated the top 100 documents for each query ranked by their scores.

4. The scores are calculated using the default ranking function of Lucene.

Query by Query Analysis

Since we found BM25-Stemming to be the most promising system when compared to TF.IDF based on the evaluation results, we have given the query by query analysis for the first three stemmed queries using this system.

1. **Query:** portabl oper system
   Top 5 documents retrieved by system BM25-Stemming are as follows:
   12 Q0 CACM-3127 1 17.978680326 PC

12 Q0 CACM-2246 2 11.5446217705 PC
12 Q0 CACM-3196 3 9.71542422701 PC
12 Q0 CACM-3068 4 9.30855939095 PC
12 Q0 CACM-2319 5 9.11047024451 PC

The top document "CACM-3127" contains information about "Thoth, a Portable Real Time Operating System" which looks topically relevant to the query. It contains the maximum number of words in the query and therefore it is at the top. It has a word count for *portabl* as 2, *oper* as 2 and *system* as 5. The top three ranked documents also appear in the given relevance information for this query. Also, stemming might result into contextually nonrelevant words getting matched to query terms, thereby resulting in somewhat inappropriate results as in the last two documents. The last two documents talk about the stems *oper* and *system*, but not about *portabl,* because of which even though they are topically less relevant, they are coming up in the top 5 ranks.

2. **Query:** code optim for space effici
   Top 5 documents retrieved by system BM25-Stemming are as follows:
   13 Q0 CACM-2897 1 12.4647667399 PC
   13 Q0 CACM-1947 2 11.9067022298 PC
   13 Q0 CACM-1795 3 11.7504286797 PC
   13 Q0 CACM-2495 4 11.5866193379 PC
   13 Q0 CACM-2491 5 10.3691928489 PC

   For this query, the top ranked document, "CACM-2897" contains the maximum number of words in the query and therefore it is ranked at the top. It has a word count for *code* as 11, *optim* as 10 and *for* as 3. Also, all the five documents are there in the given relevance information for this query. Hence the retrieval for this query seems very effective.

3. **Query:** parallel processor in inform retriev
   Top 5 documents retrieved by system BM25-Stemming are as follows:
   19 Q0 CACM-2714 1 8.91706013839 PC
   19 Q0 CACM-2973 2 8.67294417455 PC
   19 Q0 CACM-0950 3 8.16168328042 PC
   19 Q0 CACM-2266 4 8.15776803415 PC
   19 Q0 CACM-2433 5 8.10577700748 PC

   For this query, we have observed, that the top ranked document, "CACM-2714" contains the maximum number of words in the query and therefore it is ranked at the top. It has a word count for *parallel* as 07, *processor* as 3 and *in* as 3. Also, the top 4 documents are there in the given relevance information for this query. For the fifth document however, the stem *parallel* occurs 5 times, due to which it is at this rank but it is not topically relevant to the query.

T-Tests

Using ***scipy*** library available for python, we have calculated T-Tests values for all the systems. During the calculation, we compare the average precision of two systems and derive the t-values and p-values and compare their effectiveness.

All the result files generated are saved at:

<<SOURCE_CODE_FOLDER>>/ OUTPUT/ EVALUATION_RESULTS

And the file for the results is named as follows:

ALL_SYSTEMS_T_TESTS_RESULTS.csv

Snippet Generation

For snippet generation, we are using the BM25-Baseline system. For each query, we have used the Luhn's approach to calculate the significance factor for each word in a sentence and then calculate the total score of the sentence using the scores of all the words in the sentence. Thereafter, we collect the scores of all the sentences and pick the sentence with the maximum score which is then used in query terms highlighting for that document. The snippets are collected for each query and then saved in a file named as

SNIPPETS_FOR_QUERY_<<query no>>.html

Where query no = is the query number.

These files containing the snippets are saved at:

OUTPUT/TASK1/ SNIPPETS_GENERATION_RESULTS_FOR_BM25

Results

1. **MAP and MRR Values for all systems**

The values of the evaluation results (MAP and MRR) for all the 7systems are presented in Table 2:

System	MAP	MRR	P@5	P@20
BM25-Baseline	0.518256	0.546959	0.488462	0.254808
TFIDF-Baseline	0.317758	0.260989	0.253846	0.186538
Lucene-Baseline	0.416707	0.398548	0.369231	0.203846
BM25-QueryExpansion	0.341704	0.267172	0.265385	0.181731
TFIDF-QueryExpansion	0.257439	0.194014	0.180769	0.125

BM25-Stopping	0.519695	0.5936	0.476923	0.274038
TFIDF-Stopping	0.361747	0.316347	0.269231	0.2

Table 2 Mean Values of MAP and MRR

2. **Precision and Recall values for all runs of all queries**

   The results for precision, recall, P@5 and P@20 are presented in the below files:

   For precision, recall values, files are named as:

   PRECISION_RECALL_VALUES_OF__<<SYSTEM_NAME>>

   Where SYSTEM_NAME can be any one of the systems above.

   Example: PRECISION_RECALL_VALUES_OF__BM25-Baseline.csv

3. **P@5 and P@20 values for all runs of all queries**

   For P@5 and P@20 values, files are named as:

   P@K_VALUES_OF__<<SYSTEM_NAME>>

   Where SYSTEM_NAME can be any one of the systems above.

   Example: P@K_VALUES_OF__BM25-Baseline.csv

Conclusions and Outlook

After observing all the systems and their results and performance metrics, we have concluded that, in our project, BM25-Baseline system is the best performing system among the baseline runs because it uses relevance information for rank calculation and outperforms TF.IDF and Lucene base models. When we do query expansion in BM25-QueryExpansion system, since we do not use relevance information, the MAP and MRR values are low as compared to the values for BM25-Baseline system. Then when we perform BM25-Stopping and relevance information it gives the best information as we know BM25 gives the best results with stopping.

When we perform query expansion in TF.IDF models, the performance drops in terms of MAP and MRR values due to irrelevant words coming up in expanded query. In TF.IDF stopping, however, the performance increases since we use stopping which helps in removing irrelevant words and thereby increasing the performance of the system.

Bibliography

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

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

[3] http://iaingallagher.tumblr.com/post/50980987285/t-tests-in-python