Building Search Engine and Evaluation

CS 6200 Information Retrieval Project - Spring 2016
Northeastern University
Prof. Nada Naji

Ajay Subramanya , Kamlendra Kumar , Smitha Bangalore Naresh

Introduction

The goal of the project is to build various components of the search engine like Parser, Indexer and Retrieval models. Perform certain optimizations such as Stemming, Stopping, Query Expansion and finally evaluate the retrieval models using metrics such as precision, recall, MAP, MRR and precision at K.

We used CACM corpus which contained html documents along with queries and relevance judgement.
It also contained stemmed version of the documents and stem queries to be used during stemming task to understand how stemming improves the relevance of search results.

The report briefly explains how each of these are implemented along with the assumptions made and techniques used.

Effort
Smitha Bangalore Naresh: Brainstormed to design the project. Worked on initial setup (i.e. Task1) and implemented parsing and plugged in tokenizing, indexing and with BM25, Lucene and TF-IDF retrieval models and implemented pseudo relevance feedback(Task 2) and explored other query expansion techniques. Wrote the skeleton of the report and helping in bugfixing and testing

Kamlendra Kumar: Brainstormed with the team over task details. Implemented the evaluation module. Researched about query expansion techniques, available apis to implement it. Implemented Synonyms query expansion using Lucene Wordnet.

Ajay Subramanya: Worked on brainstorming ideas on how to implement the project. Discussed with the team the various evaluation metrics that needed to be used. Implemented Task 3 - Stopping and stemming. Performed query by query analysis for the stemmed corpus.

Literature and resources

Following query expansion methods are implemented :

1. ***Pseudo Relevance Feedback***: We took top 10 ranked documents from BM25 for each query and found the term frequency in the pseudo corpus (10 docs). After removing the stopwords and query words and get the top 20 terms and add it to query and run BM25.

2. ***Synonyms*** : Lucene wordnet synonyms list was extracted and used to get the synonyms. 3 synonyms are extracted for each query term that are not stop words and appended to the query. We experimented with different number of synonyms and synonyms selection techniques but could not observe any conclusive pattern.

References:

Textbook (Search Engines Information Retrieval in Practice)
https://lucene.apache.org/core/3_0_3/api/contrib-wordnet/org/apache/lucene/wordnet/package-summary.htm
lactice)
http://nlp.stanford.edu/IR-book/html/htmledition/relevance-feedback-and-query-expansion-1.html

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Implementation

Each phase of the retrieval process along with the formula and techniques used are explained below:

Parsing :

For each html document in the corpus following cleanup steps were performed :

1) Removed all noise such as html tags, script, img tags and any anchor tags with citations etc.
2) Removed all punctuation marks excluding the following -.:,
3) '-' , and last ending '.' is retained for word like New-York, A.(esp used in Names). '.' and ',' is retained for numbers and removed from alphabetical word.
4) The CACM corpus contains numbers at the end which has to be removed after encountering occurrence of :pm or :am followed by numbers.
5) Finally case folding to lower case is applied and each cleanup document is written to a text file(removing any punctuations) with same name as html file under parsed directory. CACM-1755.html -> CACM1755.txt

Tokenizing and Indexing:

Create a inverted index data structure before starting. Map<Term, Map<DocId, TF>>

Create a documentLen data structure to store the number of tokens in each document. Map<DocId, DF>

For each parsed document in the following steps are applied :

1) Read the document and tokenize by splitting on single space to get the list of words(Unigrams). Update the documentLen map with DF.
2) Iteratively update the indexed index for each term and adding DocId and TF. If the term is already present update else create a new entry.

At the end we have an inverted index for all the parsed documents and we write it to a file. Also methods for deserializing are provided.

Retrieval Models:

Three retrieval models are implemented and are as follows:

1) BM25 : Standard algorithm of BM25 with parameters k_1=1.2 b=0.75 k_2=100 and no relevance information.

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

2) Lucene : Using default implementation of their ranking function with SimpleAnaylzer.
3) TF-IDF: Using the formula below where f_{ik} is the TF of kth term in query i and zero if f_{ik} is zero, N is total number of documents and n_k is number of documents term k is occurring.

   Score per doc = $(\log(f_{ik}) + 1). \log(N/n_k)$

4) Pseudo Relevance feedback and BM25(Refer Query Expansion section below.
5) Synonyms and BM25(Refer Query Expansion section below).
6) Stopping with BM25(Refer Stopping and Stemming section).
7) Stemming with BM25(Refer Stopping and Stemming section.
8) TF-IDF with Synonyms and Stopping.

Ranking:

Queries are read from the provided cacm.query file. For each query all the 3 base retrieval models are run using inverted index built (BM25, TF-IDF and Lucene) and top 100 results are written to files (Q1_BM25, Q1_Lucene, Q1_TFIDF). Each file contains query results in standard TREC format. Ex: 1 Q0 CACM2319.txt 1 19.967466707595143 ASK_IR.

Query Expansion:

Following query expansion methods are implemented :

Pseudo Relevance Feedback: Take top 10 ranked documents from BM25 for each query and find the term frequency on the pseudo corpus(10 docs) and remove stop words and query words and get the top 20 terms and add it to query and run BM25.

Synonyms : Lucene wordnet synonyms list was extracted and used to get the synonyms. 3 synonyms are extracted for query words that are not stop words and appended to the query. We experimented with different number of synonyms and synonyms selection techniques but could not observe any conclusive pattern.

Stopping and Stemming:

Stopping : First the stop list is read and then the corpus is filtered and stopwords are removed after tokenizing. Once the tokens are free of stop words they are indexed and fed to BM25 with the queries which also go through the previously mentioned steps for tokenizing. After which BM25 returns the most relevant documents in the corpus.

Stemming: Stemming was a more involved process. First the stemmed corpus is tokenized and built into an Inverted Index, this needed some text processing as all the documents in the corpus were in a single file and also there was some noise which needed to be removed. Once the noise is removed , all the documents are read into a HashMap and indexed. After this, the seven queries are taken and fed along with the index to our implementation of BM25. This returns results which should and in most cases are better than the corpus that was not stemmed.

Below are the query by query analysis of three stemmed queries,

portabl oper system

For this query we compared using BM25 for the stemmed and unstemmed corpus.

From the results we can conclude that there is a 40% overlap . Also looking at the results we noticed that stemming would increase the term frequency, which influences BM25 score and hence increase recall. In this specific case we observed that the top two docs were the exact same, but looking at the third document, it seemed like the stemmed result was more relevant to the query.

Also with stemming, we match results that have just system, which is a good thing.

parallel algorithm

For this particular query, in addition to the above generic comments, there is one important observation - in the queries that are not stemmed this query is “parallel algorithms” , what this does is ignores the documents in the corpus that have algorithm which is also a widely used word for algorithms.

distribut comput structur and algorithm

For this query, stemmed version outperforms without stemmed version as there are a lot of word that would give better results when stemmed. For instance computers, computing would match comput and similarly algorithm matches algorithm and algorithms. Also structures match structure and structures and probably many others.

Evaluation:

To evaluate performance of our search engines (runs) in terms of retrieval effectiveness the following measures are used. Relevance judgements for each query is provided in cacm.rel file. The results for each are written to a file.

1) MAP(Mean Average Precision)
2) MMR(Mean Reciprocal Rank)
3) Precision@K,K=5 and 20 (Average Precision of the search engine at K)
4) Precision & Recall

Results

Precision and recall table for each query and retrieval model can be found
https://drive.google.com/open?id=0B7Lmot5X3cxbZ2tHZks0X2hqSEE

	Retrieval Model used	**Results**
Table 1	BM25	MAP : 0.4442351769896199 MRR : 0.7330032814407814 P @ K5 : 0.3807692307692307 P @ K20 : 0.21153846153846156 Precision & Recall Table in above link for each query:<QNUM>_BM25
Table 2	Lucene	MAP : 0.42444283856318005 MRR : 0.7004631942131941 P @ K5 : 0.3730769230769229 P @ K20 : 0.20096153846153852 Precision & Recall Table in above link for each query: <QNUM>_Lucene
Table 3	tf-idf	MAP : 0.35900455133276643 MRR : 0.6040054450380536 P @ K5 : 0.30769230769230765 P @ K20 : 0.17307692307692307 Precision & Recall Table in above link for each query:<QNUM>_TFIDF
Table 4	BM25 with pseudo relevance	MAP : 0.3729343165842298 MRR : 0.5899624689482009 P @ K5 : 0.30769230769230776 P @ K20 : 0.2105769230769231 Precision & Recall Table in above link for each query:<QNUM>_BM25PseudoRel
Table 5	BM25 with synonym	MAP : 0.4288395637015576 MRR : 0.7029656241194703 P @ K5 : 0.3461538461538461 P @ K20 : 0.20961538461538462 Precision & Recall Table in above link for each query:<QNUM>_BM25SynRel
Table 6	BM25 with stopping	MAP : 0.45155160827099305 MRR : 0.738279082029082 P @ K5 : 0.38461538461538464 P @ K20 : 0.22211538461538466 Precision & Recall Table in above link for each query:<QNUM>_BM25Stopping
Table 7	tf-idf with synonym and stopping	MAP : 0.3809076124075063 MRR : 0.6125904010519396 P @ K5 : 0.3307692307692307 P @ K20 : 0.19423076923076923

		Precision & Recall Table in above link for each query:<QNUM>_T7

Conclusion and Outlook

After evaluating our data we found that BM25 with stopping gave us the best results. BM25 with query expansion gave similar results as BM25 with stopping just that more relevant documents were ranked higher than lesser relevant ones. Also, we noticed that stemming improved recall because there were more matches of documents in the corpus.

We also noticed that MRR, MAP increased when using tf-idf with stopping and synonyms as opposed to vanilla tf-idf.

In general we can notice that query expansion provides better results.

Improvements that we envision in what we have is , during query expansion we could use corpus stats and context information to get better terms. Additionally, we could use better heuristics to select the number of terms to be added to each query. We could potentially use n-gram models in our search engine. Inflectional and derivational variance of query expansion would provide query time stemming of existing query words which potentially could provide better relevance results.