Graduate Project
Information Retrieval
CS6200

Froggle

...for unbiased and ad free searches

Professor:

Rukmini Vijaykumar

Khoury College of Computer Sciences

Northeastern University

Graduate Students:

Ashutosh Ghildiyal 001236689

Krrish Mittal 00183326

Expected Contribution

Ashutosh Ghildiyal:

- Query Expansion using Stemming
- Query Expansion using Pseudo Relevance Model
- Query Expansion using Thesaurus, Synonyms
- Spell-Checker using Edit Distance
- Documentation

Krrish Mittal:

- Tf-idf scores
- Binary Independence Model
- BM25
- Lucene
- Implementing Stopping, Stemming
- Evaluation of Search Engine

Results obtained from different runs were discussed together. We analyzed the results obtained. We came up with the implications that we thought were correct. After a discussion, we put in those points in the report that we both agreed on.

Task1: Baseline Runs

TF/idf:

TFIDF or tf–idf , short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. We chose tf-idf as one of our retrieval models because tf-idf score is not affected by the presence of stop words, it gave us a better measure of the relevance of the words (and documents that contain those words). Since the main objective of our approach was to build a system that selects unique and relevant documents and present them in a ranked order to the user, tf-idf proved quite accurate since it is good with similarity measure between two documents. Eliminating documents with low relevance was both easy and accurate.

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

However, we want to verify our findings. In order to do so we used another model that relies on the “Independence” of the occurrence of words in a document, also known as Binary Independence Model. "Independence" signifies that terms in the document are considered independently from each other and no association between terms is modeled. This assumption is very limiting, but it has been shown that it gives good enough results for many situations. The result obtained from tf-idf were closer to the subject, for example the query about time sharing system fetched documents about system itself and the problems that uses this system. The result had data related to the query, however the first few results obtained from Binary Independence model were much more accurate. Documents fetched by BIM had information such as definition of time-sharing system, information about how to build such a system, key components of such a system and so on. Results obtained from BIM and tf-idf were somewhat similar int the top 50 results. This might be because both the models assume independence occurrence.

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

Also, If we assume pi constant, si approximated by entire collection, get idf-like weight.

Further we used BM25 and lucene ‘s default retrieval model. BM25 is a popular and effective ranking algorithm based on binary independence model. It adds document and query term weights.

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

Top 5 documents retrieved by different retrieval models for query 13.
This query helped us evaluate, whether the retrieval model is able to search documents when queries are not just key words or a phrase. Results generated from running this query are displayed in the table below.

Query-13: Query is about code optimization for space efficiency

tf-idf	lucene	BIM	BM25
2897	2897	1974	2897
2748	1947	2666	1947
1362	1795	2897	1886

Results fetched by BIM, tf-idf, Lucene and BM25 are all about code optimization. The documents fetched by tf-idf, Lucene and BM25 were more accurate. The results fetched by BM25 were more accurate than any other retrieval model. They were about “object code optimization”, “generation of optimal code to save space” and a “case study of new code generation technique”. Results for Lucene and BM25 were very similar however the documents fetched by BM25 were more accurate later on. BM25 results had data about both code optimization and space efficiency. 2897 can be considered as the most relevant document.

For the follow on tasks, we used BM25, since the results generated by BM25 were relevant, unique and accurate. However, we are submitting other runs with Lucene, tf-idf and BIM model as a part of our test collection.

Task-2: Query-Expansion

While expanding query, we make sure that the expanded query and the original query does not have much difference in their length (limited number of new words were added). Also, the scope of the query should not increase, it should rather be more specific than before. This helped in obtaining results that were more relevant a and high precision value.

Query-Time-Stemming: This is another technique for increasing the flexibility of the search engine. In conventional approach, words that are not stop words are taken from the query, stems for such words are generated and are then added to the query in order to expand. One can also evaluate proximity measure or co-occurrence parameter for the new words and then decide what words to add. The BM25 scores surely improved once we put in the stem words, lemmatized words in the query. The document scores almost doubled after the expansion. (Query : 13)
: BM25 Scores

Doc Number	Before Expansion	Expansion using stemming, lemmatization
2897	11.9324	23.864
1947	11.0497	22.099
1886	10.922	21.84
1795	10.724	21.44
2680	10.7112	21.44

Implementation:
We made a slight modification to this approach. We evaluated stems for the non-stop words in query, if the stem exist in the English dictionary and the stem is not already present in the query, we added it to the query. However, if the stem does not exist in the English dictionary, we added the lemmatize form of this word to the query provided that it does not exist in the query. The lemmatized form a word is its noun form. For example, “visiting” will be converted to “visit”. Further we added the synonyms of the words obtained earlier to the query. We limited the number of words added to the query to n =6 to n = 8.
We used wordnet package to obtain synonyms of the newly added words. No duplicate words were added to the queries.

Pseudo-Relevance-Feedback: In this mode of query-expansion, the system simply *assumes* that the top-ranked documents are relevant. Words that occur frequently in these documents may then be used to expand the initial query. This method is based on the fact that the specific method for modifying the query depends on the underlying retrieval model. Before evaluating Pseudo relevance feedback, we tried finding the results for different values of k, from k = 10 to k = 70. The most promising or accurate results were obtained at k = 40. So, we finalized the number of relevant documents to be 40 for further calculations.
Implementation:
The documents were sorted as per their BM25 score and the top 40 documents were selected. High frequency words that are not in the stop list were extracted from this document. A co-occurrence (dice’s co-efficient) parameter was then calculated using the words from the query and the newly extracted high frequency words. If the new words co-occur with the query words in a certain number of documents, they were considered for expansion. Again, the number of words that were added to the query were limited from n = 6 to n = 8. The results obtained on searching the expanded query is discussed below. The score after expanding the query improved because the terms added to the query were taken from the top documents. This resulted in an increase of the doc score. (Query 13)

BM25 Score

Doc Number	Before Expansion	Expansion using Pseudo Relevance Feedback
2897	11.9324	42.9251
1947	11.0497	42.0591
1886	10.922	40.8579
1795	10.724	40.2561
2680	10.7112	40.2354

The above two results show that pseudo relevance feedback boosts up the doc score and the reason behind is -> the words taken for expansion are taken from the top documents using dice’s co-efficient.

Task-3: Stopping & Stemming

Stopping: The stopping component has the simple task of removing common words from the stream of tokens that become index terms. The most common words are typically *function* words that help form sentence structure but contribute little on their own to the description of the topics covered by the text. Examples are “the”, “of”, “to”, and “for”. Because they are so common, removing them can reduce the size of the indexes considerably.
The inverted index generated initially is free from stop words, we now removed the words from the query also.

Stemming:
Stemming is another word-level transformation. The task of the stemming component is to group words that are derived from a common stem. Grouping “fish”, “fishes”, and “fishing” is one example. By replacing each member of a group with one designated word (for example, the shortest, which in this case is “fish”), we increase the likelihood that words used in queries and documents will match.

Let’s analyze the result for the first three queries:

query 1 "portabl oper system": = portable operating system

3196	2246	1930	2593	3172

The documents are not that relevant to the query, The actual query is portable operating system, but due to stemming the search all words with oper : operations, operational, similarly portabl will be matched with any word with same stem.

Query2: code optim for space effici = code optimization for space efficiency

1947	2491	3033	2801	1807

For this particular scenario the stemming worked quite well as all the top fetched documents have words like efficient, efficiency, optimal, optimization and optimal. These words have somewhat similar meaning to the original query. Also, documents are relevant to the search

Query3: Parallel algorithm (same as original)

2973	0950	2266	3075	315

Stemming didn't work for this particularly since the stem is same as the original query. However it showed some good results since words like parallelism and paralleling were stemmed to parallel. Hence increasing the score of the document.

In my opinion stemming depends on your query, sometimes it has positive effects as seen in query 2 and quer3 but sometimes it is advisable to use (policy and police same stem).

Snippet Generation:

For snippet generation, we extracted the non- stop words from the query. These words are then searched in top 100 retrieved documents for that query. A sentence is extracted from the documents if the score for that sentence is above the threshold score. Score or weightage for a sentence is calculated using a simple formula:
Let say number of unique query words in a sentence for index i to index j is x.
Let say the total number of terms from t to j is y

Score = x^2 / y

*The snippets extracted here will later be used for the evaluation process. Document with high snippets score will be considered relevant. Let's see a snippet for
query 13: code optimization for space efficiency

Snippet:1
generation of optimal code for expressions via factorization

*given a set of expressions which are to be
compiled, methods are presented for increasing the
efficiency of the object code produced by first factoring
the expressions, i.e. finding a set of subexpressions*

The query had a total 100 hits, this snippet is taken from document 1886. So, the part of the page of extracted during snippet generation has information about the query. Similar runs on other queries revealed that queries with key words specific to documents has better snippets that long queries.

The result also verifies the list of documents that we obtained in the first run. BM25 had document 1886 in its top three search results. Now we are getting a snippet from that document. Similar snippets were obtained for document 1947 and 2897.

Evaluation:

For obtaining the ninth run we combined tf-idf with stemming (query expansion technique).

Results obtained are as follows: tf-idf scores:

Doc Number	Before	After Combining
2897	.3657	.7312
2748	.3534	.7069
1362	.3374	.67499
2464	.3289	.6579

From the results above, one can infer that combining tf-idf with squery expansion using stemming boosted the scores of documents. The score almost doubled for most of the cases.

Recall and Precision are a good metric to evaluate the results obtained from the search engine. The definition of these measures assumes that, for a given query, there is a set of documents that is retrieved and a set that is not retrieved (the rest of the documents). This obviously applies to the results of a Boolean search, but the same definition can also be used with a ranked search. We used MAP and MRR that averages over the value of precision and recall.

MAP:
Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

where Q is the number of queries.

MRR:

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\mathrm{rank}_i}.$$

The mean reciprocal rank is a statistic measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by probability of correctness. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer: 1 for first place, ½ for second place, ⅓ for third place and so on. The mean reciprocal rank is the average of the reciprocal ranks of results for a sample of queries Q.

Results obtained from the above evaluation metric is displayed in the table below:

Retrieval Model	Mean Reciprocal Ranking	Mean Average Precision
Lucene	0.653	0.404
BM25	.7928	0.4931
BIM	0.515	0.304
Tf-idf	0.71221	0.4178
BM25 (Query Expansion with stemming)	0.7281	0.4631
BM25 (Query Expansion with Pseudo Relevance Feedback)	0.7381	0.4781
Tf-idf (Query Expansion with stemming)	0.70121	0.4018

The performance of BM25 is much better than Lucene and tf-idf models. We noticed this difference at the very beginning of our report. The reason behind this difference in the performance is the relevance information that BM25 has.

We used two approaches of query expansion. The first one was pseudo relevant feedback and second one was expansion using synonyms of stem (or lemmatized) words. Synonyms has bad impact on the performance and the reason behind this is the that they can have varied meaning in varied context. On the other hand, pseudo relevant feedback has less drastic impact and the reason behind this is that the words used in the expansion were taken from the document.

Query expansion techniques didn’t make any significant improvement in the original results.

Conclusions & Results:

- BM25 is better than tf-idf, Lucene, BIM.
- Though using expansion techniques increases BM25 score of documents, they were not that significant in the final result.
- Expansion using pseudo relevance feedback proved to be better than expansion using stemming and synonyms.

Extra Credits:

For extra credits we implemented a spelling suggestion system using the concept of edit distance.

In computational linguistics and computer science, edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. Edit distances find applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question.

We used an approach similar to the one mentioned above. However, we used our index instead of the dictionary. If the word in the query is not a stop word and is unrecognized, the system calculates the edit distance for the above word with every word present in the index. The first six terms with edit distance less than or equal to 1 or 2 are then suggested to the user. The results that we obtained were further tested against the result obtained from the hunspell.
Hunspell.spell(wrongly_spelt_word) => suggestions.
The approach proved pretty accurate if the wrongly spelt term was actually present in the index.
Here are some examples:
Wrongly_spelt_word = rig
Suggestions expected = rigid, right, rigged (from hunspell)
Suggestions actual = right, rigid, triga, original, rodriguez (from the implementation)

Initial Query: portabe comptrs
Suggestions:
Portable | compares, compass, compare, computes, computers

Initial Query: preervd desriptoy

Suggestions:

preserve, preserved | descriptor

Bibliography:

- Search Engines, Information Retrieval in Practice by W. Bruce Croft , Donald Metzler, Trevor Strohman

- Modern Information Retrieval" by Ricardo Baeza-Yates.

- "Learning to Rank for Information Retrieval and Natural Language Processing" by Hang Li

- Course Material from CS 6200 Spring 2019

- www.wikipedia.org