CS6200 Information Retrieval Final Project

- Course Name: Information Retrieval
- Semester: Spring 2015
- Team Members: Wei Xu, Yu Xie, Yue Liu
- Intructor: Prof. Naji

Introduction

Desciption

This project uses a combination of Java and Python. All retrieval models themselves are written in Java, and everything else including query expansion, evaluation, and other text processing is written in Python.

Contributions

- Wei Xu:
  - Builds BM25 search engine
  - Builds Tf-Idf search engine
  - Documentation
- Yu Xie:
  - Implements query expansions
  - Implements evaluation scripts
  - Documentation
- Yue Liu:
  - Builds Lucene search engine

    - Implements stopping parser
    - Documentation

Literature and Resources

Retrieval Models

We used 3 retrieval models:

1. BM-25
2. Tf-Idf
3. Lucene (default setting)

BM 25 is the one selected to apply advanced searching options, including stopping, stemming, and query expansion.

Stopping

Stopping is a technique that exclude words that are too common to identify targeted documents when evaluating scores of documents for given queries. The provided Common Words List is used. When applying stopping, these words are both excluded from documents and queries.

Query Expansion

Words API is used to retrieve information about synonyms and derivants of English words. It has 2500 free hits each day, which is enough for our project.

Even for search engines that do not apply stopping, we elimitate common words when expanding queries. These words would remain in queries, but we do not expand them. For example, it doesn't make any sense to expand query "what is cacm" into something like "what which whats is are am be cacm".

We use the same term weigts for original query terms and expanded terms, and two expansion techniques are used as follows:

1. Synonym

Expand queries with words that have similar meanings. For example. the word "code" is expanded as 6 other words: "encipher", "cypher", "encrypt" "codification", "inscribe", and "cipher".

2. Derivants

Expand queries with derivants of the original term. For example, the word "code" is expanded as 3 other words: "codify", "coding", and "coder".

External Libraries Required

1. unirest: The HTTP request library;
2. Natural Language Toolkit: The NLP library;

Implementation and Discussion

Other than the relevance information for BM25 model, we did not use any corpus-specific or query-sensitive optimization.

Query Expansion

First, we tokenize the original 64 queries using parseQueries.py.

Then we use Words API to retreive (getWordInfo.py) synonyms and derivants of each term that apper in tokenized queries. After that, we got two files derivative and similar. The first word of each row of these files comes from original queries, and remaining ones are expanded query terms. When expanding, we exclude common words (CommonWords.py).

Finally, use getQueries.py to appy the query expansion. At the same time, we also generate queries with stop words excluded. A combination of stopping and expansion (using derivants) is also applied. Together we got five files in total:

1. Original Queries
2. Expanded Queries Using Derivants
3. Expanded Queries Using Synonyms
4. Queries without Common Words (Stopping)
5. Expanded Queries Using Derivants without Common Words (Stopping)

We use the five queries files above as following: In task 1, we use 1; In task 2, we combine 1 with 2 and 3, respectively; In task 3A, we use 4; In Phase 2 (7th run), we use the combination of 4 and 5.

Retrieval Models

1. Lucene: We are using standard analyzer.

2. Tf-Idf: For TFIDF, we are using the formula tf * idf, wher tf, idf are

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

$$idf_k = \log \frac{N}{n_k}$$

3. BM25: For Parameters, we set k1 = 1.2, k2 = 100.0, b = 0.75; And the formula we are using is

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

   In this search engine, we are utilizing the relevance informations in BM25. More specifically speaking, we retrived all the relevance infomation in the program and calculate N, R, n, r to apply them in our formular above. It's worth mentioned that the filenames in cacm.rel has some mistakes: when the file number is less than four digits, the name will omit the '0' at the frnt digits. To fix this, we mannually add the missing '0's to the relevance document.

Query-By-Query Analysis for Stemming

For query 1 "portabl oper system", I think the original meaning of this query is search the "portable operating system", and as we can see from the results, the top scored 5 documents are "2246, 1930, 3196, 3127, 2593". For 2246, 1930, 3196 and 2593, I think they are less relevant to the query, it's partially because of the stemming it self. For instance in 2246, there is not "operating" or "System", but through stemming, the word "operation" in 2246 is stemmed as "oper", and thus got a high score.

For query 2 "code optim for space effici", I think the original meaning of this query should be "code optimization for space efficiency". For this query, the top 5 docs are "2491, 1947, 3033, 2801, 1807". And in my opinion, the stemming strategy for this query works pretty good. As we can see from the top documents, many words have similar meanings but are not the same format are retrived through stemming method. For example: optim, optimal, optimize, optimization - they are all retrived correctly! And for effici, efficient, efficiency, they are retirved correctly too!

For query 3 "parallel algorithm". We can see that stemming has no effect on the query words for this query. However stemming shows positive effects in the documents. "Parallelism", "Paralleling" are stemmed as "parallel", and this increases the score for these relevant documents.

In all, the effectiveness for stemming really depends on the specific words, query and documents. Sometimes it may works pretty well, but when it comes to some general words like "police", "political", "polite" etc, it may have some side effects on our results.

Evaluation

All evaluation values are obtained through running evaluation.py.

There are 12 documents that missing relevance information (34, 35, 41, 46, 47, 50, 51, 52, 53, 54, 55, 56). Thus any search engine can never find any relevant documents of these queries, so we exclude them in evaluations. It means all the averaged results are base on the remaining 52 valid queries.

Results

	Mean Average Precision	Mean Reciprocal Rank
Lucene	0.414	0.683

Tf-Idf	0.308	0.521
BM25	0.480	0.732
BM25 (Query Expansion Using Synonyms)	0.465	0.708
BM25 (Query Expansion Using Derivants)	0.468	0.728
BM25 (Stopping)	0.449	0.725
BM25 (Stopping and Derivants)	0.443	0.718

As listed in the table, the performance of BM25 is much better than Lucene and tf-idf models. Beyond the difference among retrieval models, a significant reason is that it takes advantage of the relevance information.

We used two approaches of query expansion. Both synonyms and derivants result in decrease of performance. Synonyms has a worse impact on the results than Derivants. By observing expanded queries, we can see that the synonym API includes too many similar terms, some of which may be irrelevant. For example, it expands "selection" into 7 terms including "option", "survival", "pick", and "choice". Besides spicies evolution related topic, "selection" and "survival" are not even synonyms. The derivants is less "aggresive" when expanse queries since it only include derivants of terms. Also, the decrese is nonsignificant, so it is difficult to tell which method is better.

In our case, stopping also decreases the performance in both precision and reciprocal rank.

Other evaluation reports including precision at rank 5 and 20, precision and recall of every query at every rank, can be found in directory "evaluation".

Conclusions and Outlook

Relevance information can improve performance of search engines a lot. However these data are usually not available, so we also need to focus on other improvements for search engines.

For query expansion, it is likely that we could over expanse terms. Usually some irrelavant terms are included so that precision decreses. When using query expansion, we need to be very careful and selective for the expanded terms.

Other options like stopping can also have a negative impact.

Bibliography

1. Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. "Introduction to information retrieval/Christopher D." (2008).
2. Croft, W. Bruce, Donald Metzler, and Trevor Strohman. Search engines: Information retrieval in practice. Vol. 283. Reading: Addison-Wesley, 2010.
3. Course Notes and Slides of CS 6200 Spring 2016, Northeastern University.