Cover page

Han Jiang (net_id: hj279)
COSC 488: Introduction to Information Retrieval - Fall 2021
Project Part 3

Status of assignment: Complete

Time spent on project: 30 hours

Things you wish you had been told prior to being given the assignment:
The results are mostly worse than the baseline.

Design Document

Language: python 3.8

Libraries: I only imported standard libraries except:
nltk.stem: permitted in the project requirement file
shutil: to delete a path and all the files under that path (In order to improve debug efficiency)
csv: to write the results in csv files

Run the code:
build.py
Parameters (all the parameters are required):
-[trec_files_directory_path]: directory containing the raw documents
-[index_type]: can be one of the following: "single" , "stem" , "phrase", "positional"
-[output_dir]: the directory where the result index and lexicon files will be written
-[constraint_size]: memory constraint size **<u>(if unlimited memory, the parameter should be 0)</u>**
Example: <u>python3 build.py ./BigSample/ single ./result/ 0</u>

query.py
Parameters (all the parameters are required):
- [index-directory-path]: directory contains the lexicons and posting lists
- [query-file-path]: the directory of queryfile.txt
- [retrieval-model]: in this project, only "bm25" is optional
- [index-type]: has to be one of the following: "single", "stem"
- [results-file]: the directory to generate the output file - results.txt
- [mode]: three options - "expansion", "reduction" or "hybrid"
- [top-ranked-document]: example: 10, 20…
- [top-ranked-terms]: example: 5, 10…
- [reduction-threshold]: example: 0.5, 0.8… (could be any value in expansion mode)

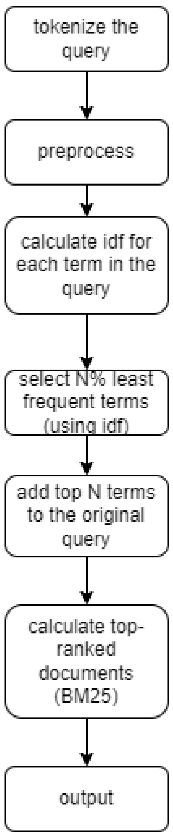
Example: python3 query.py ./result/ ./queryfile.txt bm25 single ./ expansion 10 20 0.5
Notice: the index type has to be the same as build.py
<u>Output:</u>
(1) The console will print the running time required in report 2
(2) The forward index will store in forward_index.json.
(3) If enable memory constraint, the temporary file will be generated under the path ./temp.

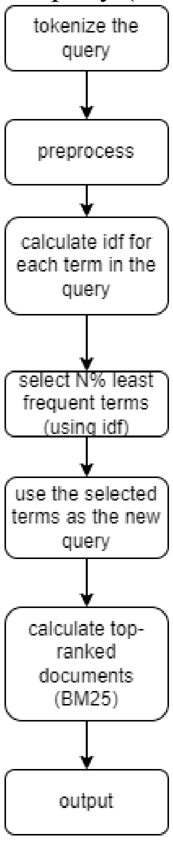
System design:
(1) Query Expansion: In this project, I implemented relevance feedback algorithm to achieve query expansion. To identify top-ranked documents, I choose BM25, and to select top feedback terms, I use n*idf as the sort criteria (n is number of

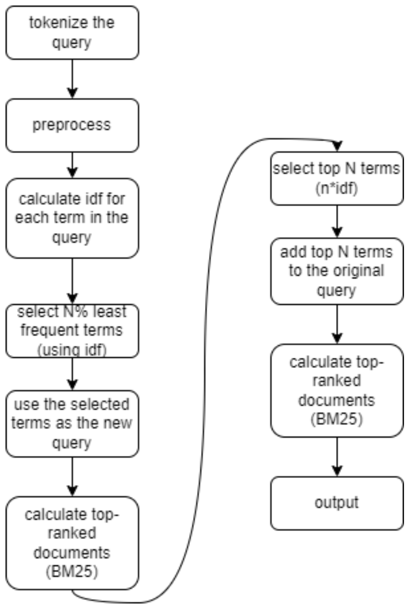
documents in relevant set having term t).

(2) Query Reduction: I implement the method “processing least frequent terms first” in query reduction with idf as the sort criteria. First tokenize the query, then calculate the idf of each term in the query, then select the least frequent n% terms as the new query. Then, calculate the top-ranked documents using BM25.

(3) Hybrid: First, process query reduction on the original query, then, process query expansion on the new query (all using the same strategy as above).


tokenize the query
preprocess
calculate idf for each term in the query
select N% least frequent terms (using idf)
add top N terms to the original query
calculate top-ranked documents (BM25)
output
Query Expansion


tokenize the query
preprocess
calculate idf for each term in the query
select N% least frequent terms (using idf)
use the selected terms as the new query
calculate top-ranked documents (BM25)
output
Query Reduction


tokenize the query
preprocess
calculate idf for each term in the query
select N% least frequent terms (using idf)
use the selected terms as the new query
calculate top-ranked documents (BM25)
select top N terms (n*idf)
add top N terms to the original query
calculate top-ranked documents (BM25)
output
Hybrid

Report & Analysis

(1) Query Expansion:
(Using relevance feedback based on **single-term index** and BM25)
Baseline: 0.4138

		Number of Top-Ranked Documents			
Number of feedback terms		1	10	20	30
	1	0.4128	0.3144	0.3113	0.3097
	2	0.4148	0.2863	0.3036	0.2769
	5	0.3993	0.2732	0.1980	0.2040
	10		0.2253	0.1681	0.1558
	20		0.2481	0.1781	0.1607
	50		0.2297	0.1795	0.1861

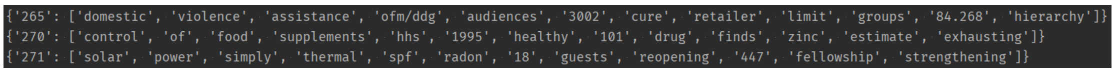
(Using relevance feedback based on **stem index** and BM25)

Baseline: 0.3819

	Number of Top-Ranked Documents			
Number of feedback terms		10	20	30
	5	0.2472	0.2594	0.2711
	10	0.2749	0.2376	0.2327
	20	0.2789	0.2090	0.1839

Analysis:

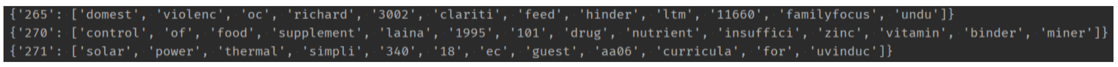
Among the experiments of query expansion, I find that most of the results are lower than the baseline. It is obvious that when having a short query, if we add too much terms into the query, there must be a lot noise. Take the figure 1 as an example.



```
{'265': ['domestic', 'violence', 'assistance', 'ofm/ddg', 'audiences', '3002', 'cure', 'retailer', 'limit', 'groups', '84.268', 'hierarchy']}
{'270': ['control', 'of', 'food', 'supplements', 'hhs', '1995', 'healthy', '101', 'drug', 'finds', 'zinc', 'estimate', 'exhausting']}
{'271': ['solar', 'power', 'simply', 'thermal', 'spf', 'radon', '18', 'guests', 'reopening', '447', 'fellowship', 'strengthening']}
```

Figure 1: expanded query (single-term index)

In the figure 1, which is the result of 20 top-ranked documents, 10 top terms expansion of single-term index, we can observe that there are a lot of meaningless terms that are added into the query. According to the results, generally, the more terms added, the worse the result is. Meanwhile, if we select the "good" words from too many top-ranked documents, basically more than 15, the precise score will decrease fast. Figure 2 is the result of 20 top-ranked documents, 10 top terms expansion of stem index.



```
{'265': ['domest', 'violenc', 'oc', 'richard', '3002', 'clariti', 'feed', 'hinder', 'ltm', '11660', 'familyfocus', 'undu']}
{'270': ['control', 'of', 'food', 'supplement', 'laina', '1995', '101', 'drug', 'nutrient', 'insuffici', 'zinc', 'vitamin', 'binder', 'miner']}
{'271': ['solar', 'power', 'thermal', 'simpli', '340', '18', 'ec', 'guest', 'aa06', 'curricula', 'for', 'uvinduc']}
```

Figure 2: expanded query (single-term index)

Comparing to Figure 1, the query added more meaningful stem like "clariti" and less meaningless terms like "ofm/ddg". Generally, the results of stem index are better than single-term index, the reason might be that the stems added to the query are more representative than the terms added from single-term index.

Except using n*idf as the sort criteria (idf in the whole collection of documents), I also tried using idf in the N relevant documents. The results are as follow:

Number of top-ranked documents	Number of feedback terms	Precise
1	1	0.3786
10	5	0.2834
10	10	0.2444
20	10	0.2536
30	10	0.2168

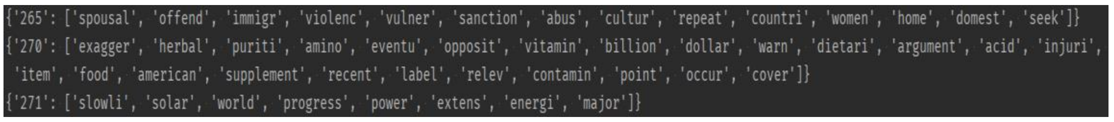
Comparing to the original sort criteria, this method only has better results when more relevant documents are selected to generate feedback terms, and the reason is obvious that if the documents selected are not enough, then idf is meaningless.

(2) Query Reduction:

Query frequency threshold	Single term index	Stem index
25%	0.1953	0.2665
50%	0.3724	0.3020
60%	0.3979	0.3153
70%	0.3460	0.3225
80%	0.3558	0.3240
100% (baseline)	0.3380	0.2733

Analysis:

During the process of query preprocessing, I removed the duplicated terms, so I use idf as the sort criteria. Figure 3 is the result of stem index query reduction using top 70% terms.



```
{'265': ['spousal', 'offend', 'immigr', 'violenc', 'vulner', 'sanction', 'abus', 'cultur', 'repeat', 'countri', 'women', 'home', 'domest', 'seek']}
{'270': ['exagger', 'herbal', 'puriti', 'amino', 'eventu', 'opposit', 'vitamin', 'billion', 'dollar', 'warn', 'dietari', 'argument', 'acid', 'injuri',
 'item', 'food', 'american', 'supplement', 'recent', 'label', 'relev', 'contamin', 'point', 'occur', 'cover']}
{'271': ['slowli', 'solar', 'world', 'progress', 'power', 'extens', 'energi', 'major']}
```

Comparing to the short queries and expanded queries, the most relevant terms are having lower order. This might because of the sort criteria, adjusting the criteria will be the future work. According to the results, query reduction using single-term index gets better precise.

(3) Hybrid (combined query expansion and query reduction based on single-term index):

		Query frequency threshold		
Number of feedback terms	Number of Top-Ranked Documents	25%	50%	80%
5	10	0.1491	0.2820	0.3408
	20	0.1263	0.3515	0.3914
10	10	0.1613	0.2755	0.3040
	20	0.1459	0.2783	0.3269
20	10	0.1581	0.2880	0.2947
	20	0.1663	0.2715	0.3165

Analysis:

According to the results, the precise increases when the threshold grows or the number of feedback terms and top-ranked documents are selected. Besides, query expansion and query reduction can affect the precise in different degree. The final result mostly dependent on the precise of query reduction. The reason is that query reduction is processed before the first time calculating BM25, namely, finding relevant documents. In that case, the relevant documents that are found totally based on what terms the query reduction left in the query. Besides, I did some extra experiments:

Top-ranked documents	Top-ranked good terms	Reduction threshold	result
3	5	0.6	0.3589
20	5	0.8	0.2666
20	20	0.6	0.2646
30	10	0.6	0.2646
30	10	0.8	0.2666

Comparing to purely query reduction, the results of the hybrid method are less precise. Which impacts this most is the sort criteria of query expansion, because the sort criteria implemented in query reduction is already relatively reasonable, in that case, I could assume that the terms left in the query in the end can represent the meaning of the query rather correctly. The only unstable factor in this system is the result of query expansion.