Cover page

Han Jiang (net_id: hj279)
COSC 488: Introduction to Information Retrieval - Fall 2021
Project Part 2: Query Processing

Status of assignment: Complete

Time spent on project: 40 hours

Things you wish you had been told prior to being given the assignment:
Maybe an expected performance of our systems.

Design Document

Language: Python 3.7

Libraries: I only imported standard libraries except:
math: to calculate log and square root
csv: to read the index from csv files

Run the code:
(1) static: 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]: has to be one of the following: “cosine”, “bm25”, “lm”
- [index-type]: has to be one of the following: “single”, “stem”
- [results-file]: the directory to generate the output file - results.txt

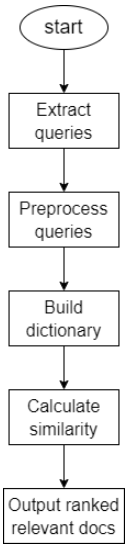
notice: please build the corresponding index before query processing
e.g.: python build.py ./BigSample/ stem ./result/ 0
or python build.py ./BigSample/ single ./result/ 0

Example: python ./query.py ./result/ ./queryfile.txt cosine single ./

(2) dynamic: query_dynamic.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
- [results-file]: the directory to generate the output file - results.txt

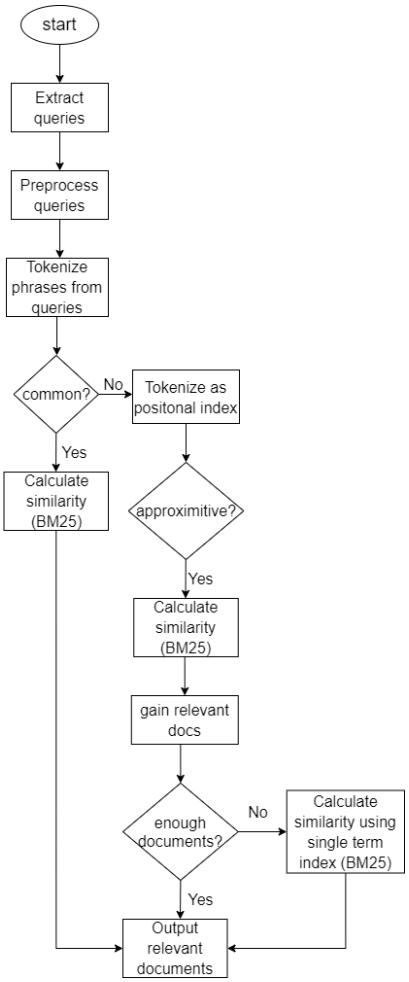
notice: please make sure to build **BOTH** positional index **AND** phrase index before query processing
e.g.: python build.py ./BigSample/ positional ./result/ 0
or python build.py ./BigSample/ phrase ./result/ 0

System design:
(1) Static query processing:
First read the querfile.txt line by line and extract the query numbers and the queries. Then preprocess the queries following the same rule as building the indexes. Then build the dictionary, calculate the similarity score using Cosine, bm25 or LM (query likelihood). Finally, sort the results and output the top 100 relevant documents.
*** Build dictionary: a dictionary shows that which words have appeared in each article:**
{doc1: {word1: tf, word2: tf, …,}, doc2: …, …}



(2) Dynamic query processing (BM25):

First, tokenize phrases from queries. Then, determine whether the phrases are common (standard: at least half of the phrases' document frequency larger than 10). If the phrases are common, then build the dictionary and calculate similarity, if not, then tokenize the queries as positional index. If the terms are within 5 positions, then the system considers them as a phrase and include the document to count similarity. If the relevant documents are not enough (100 docs), then calculate the similarity in single term index.



Report & Analysis

Report 1: Static query processing

My engine

Retrieval Model	MAP Single-term Index	Query Processing Time(sec) (Build dictionary + Calculate similarity)	MAP Stem Index	Query Processing Time(sec) (Build dictionary + Calculate similarity)
Cosine	0.19	118 + 54	0.13	346 + 44
BM25	0.41	130 + 5	0.27	346 + 4
LM	0.226	131 + 4	0.21	347 + 4

Elastic Search

Retrieval Model	MAP Single-term Index	Query Processing Time(sec) (Build dictionary + Calculate similarity)	MAP Stem Index	Query Processing Time(sec) (Build dictionary + Calculate similarity)
BM25	0.42	1	0.43	0.9
LM	0.40	0.9	0.39	0.9

Analysis:

- Building the dictionary takes too much time, the reason is that my posting lists store all of the documents that a term appears in one row of the csv file, when building the dictionary, the code walks through every document of each term. Building the dictionary of stem index takes more time is because the stemmed words appear in more documents.
- MAP results are impacted by the formula and the document list that the engine calculate the similarity.

Report 2: Dynamic query processing

Retrieval Model	MAP	Query Processing time (Building dictionary + Calculate SC)
BM25	0.36	346 + 2