Roger Wang, rw794@georgetown.edu

Cover Page:

a. Complete clustering
b. 10 hours
c. More detailed requirements about clustering. I am not sure how to use clustered query result to help ranking.

Design:

Programming language: python3

Used libraries: re, html, nltk, os, argparse, time, w3lib, math, collections, copy

Command for query processing: python3 query.py [index-path] [query-file] [retrieval-model] [index-type] [result-file] [optional: -c1 / -c2 n]

Example: python3 query.py ./index_tables queryfile.txt bm25 single -c1 2

index-path: The directory storing index files.

query-file: The path of query file.

retrieval-model: cosine / bm25 / lm

index-type: single / stem

result-file: The path for output file.

c1: Cluster document collections.

c2: Cluster query results.

n: Number of clusters.

1. Use argparse to parse arguments.
2. Load stop words for query preprocessing and single term index / stem index files for building inverted index table.
3. If there is clustering, calculate all terms' tf * idf for all documents and store them in "weight_table". "weight_table" is a dictionary. Its key is the document number. Its value is a "Counter" which is a special dictionary. "Counter" key is the term in the document. "Counter" value is the calculated tf * idf. I decide to use "Counter" because it is a powerful dictionary. Its method "most_common(n)" can be used to get *n* largest values which is useful for ranking and reduce calculations. Its method "subtract()" can be used to calculate the distance between a cluster's center and a document. I can also do plus operation between

"Counter" to sum up the value of multiple "Counter", which is useful for recalculate the center of clusters.

4. Clustering implementation details:
    a. Check if the specified number of clusters *n* is in the correct range [1, total number of document].
    b. Calculate each document each term's tf * idf and store the values in dictionary "weight_table".
    c. Assign *n* centers to *n* different documents.
    d. Create *n* groups of clusters corresponding to *n* centers. Each cluster is a "set" consist of document numbers.
    e. Unsupervised learning:
        i. I decide to iterate 10 times because it will be too slow to iterate too many times. If I have enough computation power, I would iterate lots of times. The centers will become more stable with more iterations. I would stop the iteration when the centers move below a certain threshold.
        ii. In each iteration, for each document, I calculate the distances between the document and all centers. Then, I assign the document to the cluster. This cluster has the shortest distance between the document and the cluster's center out of all centers.
        iii. After assigning all documents to the clusters, calculate a new center for each cluster. The calculation is the average of all document terms' tf * idf. To reduce calculation and speed up the program, I only keep 1000 largest tf * idf values for the new center.
        iv. After ranking all the documents, for clustering document collections, select the documents in the query's best matching cluster; for clustering query results, add an additional ranking value to the documents in the query's best matching cluster so that these documents will be in the top rank.
5. Read the query file and extract query number and query for retrieval purpose.
6. Preprocess query the same way as preprocess documents in project 1. If a term in the query is not in the index table, remove it from the query.
7. Select all the relevant documents. The relevant documents will have at least one term matching the query.
8. Implement cosine similarity, bm25 similarity, language model similarity. Assign a score for each relevant document.
9. Rank 100 highest score relevant documents and output to the file.

Analysis:

Retrieval Model	MAP single index		Query Time (sec)		MAP stem index		Query Time (sec)	
	my search	elastic search	my search	elastic search	my search	elastic search	my search	elastic search
cosine	0.2822	0.3326	3.02	0.38	0.3333	0.3225	2.91	0.53
bm25	0.4342	0.4300	1.09	0.54	0.4453	0.4795	0.95	0.57

Roger Wang, rw794@georgetown.edu

lm	0.4452	0.4310	1.07	0.39	0.4379	0.4701	0.95	0.51
cosine + 2 doc clusters	0.2822		25.55		0.3283		18.27	
cosine + 3 doc clusters	0.2543		25.95		0.3283		21.88	
cosine + 4 doc clusters	0.2506		30.06		0.3260		25.97	
cosine + 5 doc clusters	0.2504		33.37		0.3229		30.02	
cosine + 6 doc clusters	0.2358		34.91		0.3230		34.75	
cosine + 7 doc clusters	0.2358		37.13		0.3230		37.43	
cosine + 8 doc clusters	0.2358		40.6		0.3230		41.75	
cosine + 2 result cluster	0.2822		24.92					
cosine + 3 result cluster	0.2606		24.27					
cosine + 4 result cluster	0.2575		28.69					
cosine + 5 result cluster	0.2597		32.08					
cosine + 6 result cluster	0.2453		35.08					
cosine + 7 result cluster	0.2453		37.6					
cosine + 8 result cluster	0.2453		39.5					
bm25 + 2 doc clusters	0.4342		23.47					
bm25 + 3 doc clusters	0.3482		24.05					
bm25 + 2 result cluster	0.4342		23.68					
bm25 + 3 result cluster	0.3584		24.12					

1. My clustering has little improvement based on MAP. The running time is incredibly long. Because clustering can be preprocessed, it can be run before reading the query. Thus, the query time can be significantly reduced.
2. When the number of cluster is larger than 2, the performance decreases. Maybe the query originally doesn't find enough documents, so it doesn't make sense to divide documents into many categories.

3. There are lots of queries having the same MAP values. It can be because all the related documents are in the same category.
4. There can be some flaws about improving time performance by limiting the center to 1000 largest terms. Maybe all terms are important. A center with only 1000 terms doesn't have enough dimensions to measure the distance to a document. Lots of documents are too short to have terms in the center. As a result, these documents can be in the same cluster.