Cover Page:

a. Complete
b. 20 hours
c. More detailed requirements about using proximity index. I waste lots of time trying to incorporate phrase frequency into score calculation which turns out to be unnecessary.

Design:

Programming language: python3

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

Command for building index: python3 build.py [raw-documents-path] [index-type] [output-dir]

Command for static query processing: python3 query_static.py [index-path] [query-file] [retrieval-model] [index-type] [result-file]

Command for dynamic query processing: python3 query_dynamic.py [index-path] [query-file] [result-file]

query_static

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. Read the query file and filter out query number and query for retrieval.
4. 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.
5. Select all the relevant documents. The relevant documents will have at least one term matching the query.
6. Implement cosine similarity, bm25 similarity, language model similarity. Assign a score for each relevant document.
7. Rank 100 highest score relevant documents and output to the file.

query_dynamic

1. Use argparse to parse arguments.
2. Load stop words for query preprocessing; single term index / stem index files for building inverted index table; phrase index and proximity index files for finding relevant documents
3. Read the query file and filter out query number and query for retrieval.
4. 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.
5. Select all the relevant documents using phrase index. If the number of relevant documents is not enough, select more relevant documents using proximity index. If the

number is still not enough, select more relevant documents having at least one term matching the query.

6. Implement language model similarity. Assign a score for each relevant document.
7. Rank 25 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
lm	0.4452	0.4310	1.07	0.39	0.4379	0.4701	0.95	0.51

Retrieval Model	MAP	Query Time (sec)
lm + single	0.4287	3.02

1. When read from index files to build inverted index tables, build a document index table to help calculating similarity scores (dividend of cosine similarity, document length). When reading, calculate total document length for bm25 and lm to prevent going through all the documents again. It significantly reduces query time.
2. My search time includes reading from index files and writing output to a file, so the time is longer than elastic search time.
3. My search engine's bm25 single MAP is larger than elastic search's bm25 single MAP. I change elastic search bm25 k1 = 1.2 and b = 0.75 which are the same as my program. Elastic search doesn't support adding k2. My MAP is higher can be the result of using a slightly different version of bm25 than elastic search. I use the bm25 formula provided in the slide.
4. My search engine's lm single MAP is larger than elastic search's lm single MAP. I change elastic search lm mu = 416 (calculated average document length in my program). I am not sure why my MAP is higher.
5. I am not sure why most of my search engine's MAP is smaller than elastic search's MAP. Ideally, the MAP value should be very close. Because I don't have the output file of elastic search result, I can't do any analysis on the specific query.
6. "cosine" MAP is smaller than "bm25" and "lm" MAP. "bm25" MAP is similar to "lm" MAP.
7. My search engine's "cosine" use much more time than "bm25" and "lm". To reduce query time, I can calculate the divisor value for every document in the "cosine" formula and store the value before calculating cosine similarity. Then, I only need to pull out that value for calculation.
8. My dynamic search engine is slow because it utilizes proximity index to find matching phrases between queries and documents.