Assignment 2 Report

Steps:

I. File Parser:

- First, we create a file parser 'data_parser.py' which created a 'parsed_data.json' file for corpus. We'll use this file for all further steps.

II. Preprocessing data:

- After creating json file, we use parsed data file along with the queries to preprocess the data. Below are the steps included in preprocessing the data:
- We do preprocessing of the data in two steps:
    1. We stem the parsed data file along with the queries and remove stop words from them.
    2. During the class creation we load the stop words and remove the stopwords from parsed data and queries.
    3. Then we convert the text to lower case and split the text using regex to break any text other than continuous characters or separated by single period. ([^A-Za-z0-9.]+|\.{2,})

Output files from preprocessing:

- Stemmed:
    1. Data corpus file: data_stemmed.json
    2. Queries: stemmed_queries.json

- Unstemmed:
    1. Data corpus file: data_unstemmed.json
    2. Queries: unstemmed _queries.json

III. Tokenizer:

- For tokenizing I have created two separate python scripts, one for stemmed data and the other one for unstemmed data to compare both the results.

Also, the class created for tokenizing will create tokens both for queries and data corpus.

- After splitting text, we get the tokens.
- Any trailing or leading periods in the word are removed as they might be the last word of sentence.

Output files:

Stemmed:

- Stemmed_tokens.json
- Stemmed_words_id_dict.json
- Stemmed_doc_len_stat.json

Unstemmed:

- Unstemmed_tokens.json
- Unstemmed_words_id_dict.json
- Unstemmed_doc_len_stat.json

IV. Indexing

- We have created two separate python scripts for stemmed and unstemmed documents.
- We use the tokens that we have created from previous steps and word dictionaries
- We have created 84 separate csv files(for both stemmed and unstemmed) for creating index of terms for the corpus where each index file stores the indexes of 1000 documents from the corpus.

  Output Files:

  - indexfiles_stemmed/index/.csv
  - indexfiles_unstemmed/index/.csv

- Likewise, we have created 84 separate csv files (for both Stemmed and Unstemmed) documents from the corpus where each catalog file is made for 1000 documents. Each catalog file stores the term, offset , size for each term in

the document where offset refers to current position-initial position for each term.

Output Files:

- indexfiles_stemmed/catalog/.csv
- indexfiles_stemmed/catalog/.csv

- Then we have merged all the 84 catalog files and 84 indices file for both stemmed and unstemmed documents.
- After merging (process explained later section), we have three files as Index. (Inverted list, catalog and document map)

Output Files:

For Stemmed:

- indexfiles_stemmed/catalog.csv
- indexfiles_stemmed/catalog.csv.gz
- indexfiles_stemmed/index.csv
- indexfiles_stemmed/index.csv.gz
- indexfiles_stemmed/doc_id_map.json

For Unstemmed:

- indexfiles_unstemmed/catalog.csv
- indexfiles_unstemmed/catalog.csv.gz
- indexfiles_unstemmed/index.csv
- indexfiles_unstemmed/index.csv.gz
- indexfiles_unstemmed/doc_id_map.json

V. Merging

- For merging I load up catalog and use the term positions in catalogue and read each line of it to identify the position in inverted index and then use that information to find all details regarding that term and merge them using a dictionary. Once we get all details regarding one term then save it to disk, reset and continue.

- According to my implementation, while merging as the term info I would find all positions information of term in a document and if the document is again seen with same word in the list at other position then it implies that the document is continuing so the positions are merged.

  Output Files:

  For Stemmed:

  - Indexfiles_stemmed/catalog.csv
  - Indexfiles_stemmed/catalog.csv.gz

  For Unstemmed:

  -indexfiles_unstemmed/catalog.csv
  -indexfiles_unstemmed/catalog.csv.gz

VI. Index Compression

- For compression, I used gzip library.
- While retrieval, I used gzip to directly open the file reader and used it as normal implementation.

  Output Files:

  For Stemmed:
  - indexfiles_stemmed/index.csv
  - indexfiles_stemmed/index.csv.gz

  For Unstemmed:
  - indexfiles_unstemmed/index.csv
  - indexfiles_unstemmed/index.csv.gz

VII. Performance Requirements:

- I am only maintaining information of 1000 documents in memory to solve the first requirement.

- The space constraints are as mentioned and are described below.

Index Sizes:

Uncompressed stemmed(indexfiles_stemmed/index.csv)	197.9 MB
Compressed stemmed(indexfiles_unstemmed/index.csv.gz)	86.9 MB

Model	**Compressed stemmed**	**Uncompressed stemmed**	**Uncompressed not stemmed**
Okapi	0.2368	0.2368	0.1600
Tfidf	0.2990	0.2990	0.2015
Okapi BM25	0.2957	0.2957	0.2028
Unigram Laplace	0.2281	0.2281	0.1543
Unigram JM	0.2635	0.2635	0.1787

- The expectation is compressed stemmed to perform same as uncompressed stemmed as its same information, which is observed in our results.
- The expectation is stemmed index to perform better than unstemmed as we might lose information about query term which is in the relevant document in other form. We observe this in our results.

VIII. Searching

- I modified my programs to use the inverted index created by me and the results are as like the ones achieved in Assignment 1.
    - For getting terms information from inverted index, I have created 2 python scripts for stemmed and unstemmed data-retrieval_model_stemmed and retrieval_model_unstemmed.
    - After this we have implemented all retrieval models for stemmed documents to compare the results between Assignment 1 and Assignment 2.

Stemmed files for comparison:

Model	Assignment 2	Assignment 1
Okapi	0.2021	0.2167
Tfidf	0.2637	0.2820
Okapi BM25	0.2674	0.2833
Unigram Laplace	0.1842	0.1955
Unigram JM	0.2306	0.2489

Stemmed:

Model	Mean Precision	Precision at 10 documents	Precision at 30 documents
Okapi	0.2021	0.3520	0.2827
Tfidf	0.2637	0.4240	0.3427
Okapi BM25	0.2674	0.4120	0.3307
Unigram Laplace	0.1842	0.3720	0.2893
Unigram JM	0.2306	0.3640	0.3280

Unstemmed:

Model	Mean Precision	Precision at 10 documents	Precision at 30 documents
Okapi	0.1600	0.2320	0.1747
Tfidf	0.2015	0.2640	0.2387
Okapi BM25	0.2028	0.2640	0.2240
Unigram Laplace	0.1543	0.2440	0.2013
Unigram JM	0.1787	0.2600	0.2200

IX. Proximity Search

- I have used skip-gram/minimum span to calculate proximity on query terms in the document.
- Ran BM25 model to check the difference between the Assignment 1 and Assignment 2 results.
    a. Unstemmed index with unmodified queries: >= 0.18 for BM25
    b. Stemmed index with unmodified queries: >= 0.23 For BM25

Model	**Mean Precision**	**Precision at 10 documents**	**Precision at 30 documents**
Proximity Model (Unstemmed)	0.1811	0.2680	0.2227
Proximity Model (Stemmed)	0.2364	0.4000	0.3253