ReadMe and Report

To run this application you need to have these libraries, files and directories:

Task_1:
Libraries: os, nltk, collections, json, numpy

Directory: scrap – corpus data

Task_2:
Libraries: os, nltk, collections, itertools, numpy

Directory: scrap – corpus data

Task_3:
Libraries: os

Directory: inverted_index

Files: index_1, index_2, index_3 inverted index files generated in task 1- a.

The code is in python 3.7.2. Make sure you have scrap named directory and provided corpus data in it.

To run each task run python Task_X.py where X is the number of task. E.g. python Task_1.py for task 1.

From design choice point of view I chose python inbuilt default dictionary as indexing data structure because of O(1) access. It has terms as keys and list as values and again within that list it has dictionary objects with document id: term position or term frequency based on the inverted index. For inverted index in task 1 it was

{"term" : [{"document1": "term_frequency"}, "document2": "term_frequency"]…….} and in

delta encoding it was

{"term" : [{"document1": "first_position, , d gap, , d gap"}, "document2": "first_position, d gap, d gap"]…….}

Output:

Task1 has output files for inverted index from 1-a named as **index_1.txt**, **index_2.txt**, **index_3.txt** for unigram, bigram and trigram respectively in *inverted_index* directory.
It has [documentId, term] table in **document_terms_task1_b.txt** and last document for task 1-d the delta encoded inverted index as **inverted_index_delta_encoded.txt**

Task2 has output files are in directory *Proximity_query* and named following:
Space , mission: k= 6 File name: **documents_with_space_mission_6.txt**
Space , mission: k= 12 File name: **documents_with_space_mission_12.txt**
earth , orbit: k= 5 File name: **documents_with_earth_orbit_5.txt**
earth , orbit: k= 10 File name: **documents_with_earth_orbit_10.txt**

It is just a list of documents.

For design choice I used casefolding for each input.

Task3 has output files in directory stats and are named following:
Document frequency: **document_frequency_1.txt, document_frequency_2.txt, document_frequency_3.txt** for unigram, bigram and trigram respectively.
Document frequency is sorted by terms and is in format:

Term Dcoument1, Document2, Document3 : Frequency

Term Frequency: **term_frequency_1.txt, term_frequency_2.txt, term_frequency_3.txt** for unigram, bigram and trigram respectively.

Term frequency is sorted by most to least and is in format :

Term: Frequency

Stop List: **stop_list_1.txt, stop_list_2.txt, stop_list_3.txt** for unigram, bigram and trigram respectively.

Stop list is in format:

Term: Frequency

Stop list explanation:

To generate a stop we can use various ways like tf-idf but after experimenting with few methods document frequency table gives good result. So, to choose a word as stop word in list we have to decide a cut off value in document frequency. After bit of experimentation I found that 800 is good value. It does not affect the context of the document and it mostly contain words which has only grammatical function or particular to the Wikipedia. E.g in trigram we have words like “from Wikipedia the”, “by using this” and “statement mobile

view”. We can see that this doesn’t have anything in common with the context of the document and its mostly for navigational purposes. Number of terms with increase in n decreases the list. E.g unigram has 160 stop words and trigram has 113 stop words.