Vartan Benohanian
ID: 27492049
October 22, 2018

COMP 479 – Project 1

SPIMI Indexer

The codebase for my assignment is quite organized and easy to wrap one's head around. I try to stay efficient in my techniques by cutting down as much as I can the amount of operations needed to do something.

The meat of the application is in the *main.py* file. I also make use of a *definitions.py* file, the only prupose of which is to store the project's root directory in a variable, which was used to make sure generated files appear in the intended directory. Here is a detailed step-by-step demonstration of what's happening in the *main.py* file:

1. Initialize a Reuters object, which will parse the Reuters files. (Note: if the files aren't downloaded, it downloads the corpus for you.) The Reuters object's main responsibility is to tokenize the documents in the Reuters corpus. A document is defined as a *<REUTERS>* tag containing a *<TEXT>* tag within it. The latter contains the document content, such as title and body. The Reuters object is used to obtain a list of lists of tokens. Tokens are tuples of terms, and their corresponding document ID. The document ID is the value of the *NEWID* attribute in a *<REUTERS>* tag. Each list in this list is used to create one single block. The Reuters object takes in a few parameters:
    a. *number_of_files*: There are 22 files in the Reuters corpus, each containing roughly 1000 documents. It can take a long time to parse them all (about 90 seconds). For the purpose of accelerating the assignment, I kept this as a parameter, which I usually set to 1, so that I wouldn't have to wait a long time each time I was testing the script. The default value is 22.
    b. *docs_per_block*: Once everything is tokenized, the informaion will be split into a number of block files, which will then be merged (more on that later). This parameter specifies the number of documents each block will contain. The default value is 500.

c. *remove_stopwords*: Whether or not we will remove stopwords from the terms. Stopwords include words such as 'a', 'and', 'or', etc. The default value is false, meaning we shall not remove them.

d. *stem*: Whether or not we will stem the terms, i.e. reducing them to their minimal form. The default value is false.

e. *case_folding*: Whether or not we will convert all terms to lowercase. The default value is false.

f. *remove_numbers*: Whether or not we will remove terms that are just numbers. For this assignment, if terms have their commas and/or periods deleted, and are therefore a string of digits, they are considered a number. The default value is false.

2. Initialize a SPIMI object, which will parse the tokens returned by the Reuters object described above. The SPIMI object takes in a few parameters:

a. *reuters*: A Reuters object. We pass in the one we initialized previously in the program.

b. *output_directory*: The name of the output directory in which we will store block files and the inverted index. The default value is "DISK".

c. *block_prefix*: The prefix of block files which will be generated in the output directory defined in *b)*. The default value is "BLOCK". The block files will be numbered. The numberings will appear immediately after the prefix.

d. *output_index*: The name of the file containing the inverted index, which will belong in the output directory defined in *b)*. The default value is "index".

3. Construct the inverted index using the SPIMI object defined above. For each list of tokens in the list of lists of tokens returned by the Reuters object, generate a block file, storing terms and their corresponding postings in it. After going through all the lists, merge these blocks into a single file, containing all terms and all of their corresponding postings. This merged index is used to make AND and OR queries. I worked a lot with the set type to get the query results, so as to not have duplicate postings returned to the user.

4. A table showcasing statistics is generated, using this merged index as the baseline, unfiltered index. From there, we filter it using various methods, such as further eliminating numbers, stopwords, case folding, and stemming. We look at the differences in amount of terms in each set.

5. Before the program is terminated, we run some queries against the constructed index. For each term in the query, we look at its postings list in the index, and we do the intersection

or the union of the postings lists, depending on if it's an AND query, or an OR query, respectively.

Make sure to use Python 3 and install the required packages listed in the *README.md* file of the submission.

To run the program, type in the command line: **python3 main.py [arguments]**. The arguments are the following:

1. -docs or --docs-per-block: number of documents per block. Default is 500.
2. -r or --reuters: number of Reuters files to parse, choice from 1 to 22. Default is 22.
3. -rs or --remove-stopwords: stopwords will be removed from index. Default is false.
4. -s or --stem: terms in index will be stemmed. Default is false.
5. -c or --case-folding: tems in index will be converted to lowercase. Default is false.
6. -rn or --remove-numbers: terms that are just numbers will be removed from index. Default is false.

While completing this project, I learned that efficiency and optimization are key in data science. The Reuters corpus seemed huge while I was working with it, but it pales in comparison to what major search engines, such as Google, Bing, and DuckDuckGo, deal with. 90 seconds to run the whole Reuters corpus doesn't seem long, but once that's stretched out to a much bigger data set, the time to complete can be daunting.

The assignment was enjoyable however, and it motivated me to come up with some other, albeit much simpler, scripts used to scrape data from various websites to track prices of items I'm interested in purchasing.