Vartan Benohanian
ID – 27492049
October 22, 2018

COMP 479 – Project 1

SPIMI Indexer

Description

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 is 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 60 seconds without any compression techniques, including index construction). 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. 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. 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. Default value is false, meaning we do 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. 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. 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 one parameter:

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

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. The block files and the merged index are stored in an output directory, which is defined in the SPIMI object's *__init__* method.

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. If the index hasn't been stemmed, we stem it, and use it for the queries. The user is asked if they'd like to conduct some queries. Before each query, they must specify if it will be an AND or an OR query. Each term in the query is stemmed, in order to obtain more consistent results – e.g. if we don't stem both the index and the query and we search for "environmentalist", we wouldn't get the documents that contain "environmentalists". For each term in the query, we look at its postings list in the index, and we do the intersection of the sets if it's an AND query. If it's an OR query, we conduct some operations to place the

postings with the most keywords at the beginning of the list, and those with the least at the end. We return the postings list to the user. I worked a lot with the set data type to get the query results, so as to not have duplicate postings returned to the user.

Running

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 in index are removed. Default is false.
4. -s or --stem: terms in index are stemmed. Default is false.
5. -c or --case-folding: tems in index are converted to lowercase. Default is false.
6. -rn or --remove-numbers: remove terms in index that are numbers. Default is false.
7. -a or --all: makes arguments 3 to 6 true. Default is false.

Challenge Queries

3 AND Queries

- **Jimmy Carter**, stemmed to **jimmi carter**
  - 6 results found: 12136, 13540, 17023, 18005, 19432, 20614
- **Green Party**, stemmed to **green parti**
  - 5 results found: 3885, 5774, 10230, 13257, 21577
- **Innovations in telecommunication**, stemmed to **innov in telecommun**
  - 1 result found: 5891

1 OR Query

- **environmentalist ecologist**, not affected by stemming
  - 5 results found: 5774, 7532, 7566, 19776, 21577

Conclusion

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 on an extremely regular basis. 60 seconds to run through the whole Reuters corpus (with no compression) and generate an index 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 proved to be a great introduction to information retrieval techniques. It also motivated me to come up with some other, albeit much simpler, Python scripts. I use them for purposes such as scraping data from various websites to track prices of items I'm interested in purchasing.