Michael Stiffler

2731139

3/23/2022

CIS 492 Lab\_3

**Table of Contents:**

**1: Explanation of twitter tweet grabbing code**

**2: Explanation of twitter tweet text natural language processing**

**3: Explanation of part 4 MongoDB aggregation queries**

**4: Pictures**

For the collection of twitter data (get\_data.py), I used the **tweepy** library. I used the **dotenv** library to load my environment variables and json library to help with parsing some of the json that was return from the twitter api. I also used **pymongo** to send and receive the tweet data from my NoSQL database I set up.

import tweepy

from tweepy import Stream

from dotenv import load\_dotenv

from pymongo import MongoClient

import pymongo

import json

For the parsing of twitter data (parse\_data.py), I used **pymongo** and **json** as described above along with **spacy** to handle the natural language processing part of the lab

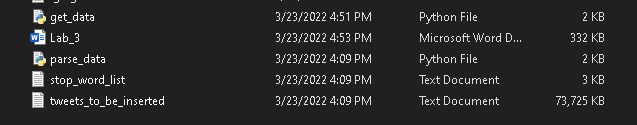
from pymongo import MongoClient

import pymongo

import json

import spacy

All of these libraries are needed to run my lab.



Here is the file structure for my lab. I have a file get\_data.py that handles grabbing the information from twitter and storing it in my MongoDB collection. Another file, parse\_data.py gathers the tweet text from my collection for each tweet and does natural language processing on them to find the top 10 most frequent words. I have a text file stop\_words.txt which contains stop words that I can use (this was from the lecture notes). Another text file tweets\_to\_be\_inserted.txt which is the list of ~10000 tweets I grabbed in json form, but stored in a text file.

**1. Explanation of twitter tweet grabbing code (get\_data.py):**

if \_\_name\_\_ == "\_\_main\_\_":

    load\_dotenv()

    access\_token = os.getenv('ACCESS\_TOKEN')

    access\_token\_secret = os.getenv('ACCESS\_TOKEN\_SECRET')

    consumer\_token = os.getenv('API\_KEY')

    consumer\_token\_secret = os.getenv('API\_KEY\_SECRET')

    connection = MongoClient('localhost', 27017)

    db = connection.LAB3

    collection = db.tweets

    db.tweets.create\_index([("id", pymongo.ASCENDING)], *unique*=True)

    stream = writeTweepyDataToFile(*consumer\_key*=consumer\_token, *consumer\_secret*=consumer\_token\_secret,

*access\_token*=access\_token, *access\_token\_secret*=access\_token\_secret)

    stream.filter(*track*=["Ukraine Russian war",

                         "Russian Invasion"], *languages*=['en'])

The main method of get\_data.py is self-explanatory. I load my environment variables with load\_dotenv() and fill each api token from my .env file using the values described in the strings. Once I have my api tokens, I load my MongoDB collection using localhost and the port 27017 and connect to my database “LAB3”. Once I am connected to the database, I move to my collection called “tweets”. I also made sure that each document that is entered is ordered ascending by the “id” and must be unique.

After this I start my tweepy initialization. Twitter API v1.1 required that a tweepy Stream class must be used, with the api tokens/keys passed into it, so that is what the stream variable does. Once I have my stream, I filter the tweets that I get by two topics, “Ukraine Russian war” and “Russian Invasion” and make sure the responses I get are in English.

*class* writeTweepyDataToFile(*tweepy*.*Stream*):

*def* \_\_init\_\_(*self*, *consumer\_key*, *consumer\_secret*, *access\_token*, *access\_token\_secret*):

        self.counter = 0

        self.limit = 10000

        return *super*().\_\_init\_\_(consumer\_key, consumer\_secret, access\_token, access\_token\_secret)

*def* on\_data(*self*, *data*):

        try:

            tweet\_data = json.loads(data)

            with open("tweets\_to\_be\_inserted.txt", "a", *encoding*='utf-8') as f:

                f.write(*str*(tweet\_data) + "\n")

                f.close()

            self.counter += 1

            if self.counter < self.limit:

                collection.insert\_one(tweet\_data)

                return True

            else:

                stream.disconnect()

        except *BaseException* as e:

            print(e)

*def* on\_status(*self*, *status*):

        print(status)

Here is the writeTweepyDataToFile(tweepy.Stream) class that I was required to make. The name is arbitrary, but when you pass tweepy.Stream, it knows this is the class that you’re going to be using to using to grab tweets. Each class requires an on\_data(self, data) method along with an on\_status(self, status) method to be define. On\_data() is used when the API returns with tweet data and on\_Status() is used when something occurs, such as an error, and the program has to handle it.

In my init() method, I declare a counter and limit variable which will help me grab tweets to a certain limit, and call super() with the api tokens/key I passed into the class earlier in main().

In my on\_data() function, I first call json.loads() on the data that was passed into the method, that being the twitter tweet that just got passed in. Once I load the data I open my tweets\_to\_be\_inserted.txt file and append the data as a string with a newline. I increase counter by 1 and checked if the counter is less than the limit (10000) and if it is, then the twitter data stream stops, otherwise, I append the tweet\_data to my tweets collection in MongoDB and return True so that the program knows that is exited successfully.

On\_status() will handle any exceptions the twitter api throws, so we just print the status if one was passed in.

**2. Explanation of twitter tweet text natural processing (parse\_data.py):**

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated