TWITTER SENTIMENT ANALYSIS

The goal of this project was to extract N data, with a specific keyword and time period, from Twitter by calling Twitter API (using **Tweepy**) and sentiment of the data would then be analyzed. A tweet may have a positive, neutral or negative sentiment. Data visualization (using **Plotly**) would be shown in a simple dashboard (using **Dash**) in the web server. Recently, I had this interest in data analysis and I had this idea to extract social media data to analyze it further. It turned out that Twitter was one of the most accessible social media to collect real time and latest data. Through sentiment analysis, we could derive the opinion or attitude of the people who posted the tweet. My first idea was to do a real time analysis in which the data could be updated every minute. But I would need a database that would store every data. This would be more costly and complicated. So, I decided to store the data I got in a csv file instead and I would analyze the data from the file. Then, I also developed my idea further to create a simple database that will show the results of the analysis because it would look more presentable and attractive than just running it in Python console.

Installation of some libraries need to be done by User before starting. This could be done downloading requirements.txt file provided, setting the current working directory where you downloaded the file in the command prompt, and then typing "**pip install -r requirements.txt**" in the command prompt.

First, I got my own authentication credentials by creating an app and requesting access through Twitter Developer website. I would get API Key, API Secret Key, Consumer Key and Consumer Secret Key. The program authenticated the keys using OAuthHandler class from Tweepy library to set the credentials used in every API call. Note that **Tweepy**[1] is an open source Python package to access Twitter API with Python. Then an API object was created by running tweepy.API to call the API. Next, the program would create a data frame that has TweetDate, UserLocation, Tweets, Username, FavoriteCount and RTCount as its columns.

The program would start when **MainProgram()** function was called. Program would ask User if he would like to extract the data by calling Twitter API or by uploading a file. I gave two options as it might need a very long time to extract data through Twitter API. This was due to limitation of 15 requests per rate limit window, then it would allow User to make 15 request per window per access token in every 15 minutes that Twitter set.[2] Therefore, I provided a csv data, which I had extracted through Twitter API before, for the second option. If User chose to extract data through Twitter API, the program would ask User's preference by calling **UserPreference()** function. The function would ask User to input one specific keyword he wanted to search, a file name, year, month and date of the starting and ending date of data. The input would then be stored in a list as it would be easier to retrieve them back through its index. I also checked if the dates that User entered exist and if start date is earlier than the end date. If these conditions were not met, the program would continue to ask User to input the correct dates. Also, the program would continue to ask User if User input an empty string either for keyword or file name. Program then asked User to enter number (positive) of data to be extracted. Program would continue to ask User if he input a wrong number.

Next the program would call **StreamTwitter()** function with a keyword, starting date and ending date that User input before. The program would get each data using tweepy.Curssor.item() by passing API.search and the attributes that User preferred such as the keyword and language set as English. The function would keep on iterating and then filter the extracted data to be in a period that User wanted. The program would extract data about the tweet creation date, location, tweets, username, number of favorites and retweet that each tweet got, and the data would be inserted at each specific column. The program then iterate for N times in which N is the number of data that User wanted to extract. At first, I got a problem when calling this function as it did not show any data for such a long time and suddenly it gave an error, and at the same time the program did not stop. It gave an error as the program has reached its API calls limit. I spent quite a long time figuring it out and finally decided to use a try and except syntax[3] to catch the error and to keep track of the number of data it had extracted using the variable Count. Although Twitter limited API calls per 15 minutes, the program would still perform API calls every 5 seconds (by sleeping the program) in order to maintain the TCP connection. When the 15-minute limit period ended, the function would continue to return the data. After the twitter data was extracted, it would be saved in a csv file which would make it more effective and efficient for analyzing the data. The program would not need to call the API again (which would take a longer time). If User chose to upload a file, then the program would ask the file name. The program would then read the file and save it as a

[1] https://realpython.com/twitter-bot-python-tweepy/#what-is-tweepy

[2] https://developer.twitter.com/en/docs/basics/rate-limiting

[3] https://stackoverflow.com/questions/21308762/avoid-twitter-api-limitation-with-tweepy

pandas data frame. I chose pandas library as it has rich functionality for data analysis and storing it as a data frame made it more accessible.

Then I would create a new column of CleanedTweet by applying **CleaningTweets()** function to Tweets column. **CleaningTweets()** function would remove URLS, RTs, usernames, hashtags, emojis [4] and repeated words punctuations and stopwords (commonly used words like 'the', 'an', 'a', etc.) as they would be unnecessary for the analysis and may be misleading. The words were split by tokenizing the sentence then joined again at the end. I used **Natural Language Toolkit (NLTK)**[5] library that made it easier to do statistical natural language processing (NLP). Specifically, I used it to tokenize words and to identify stopwords.

The program would then analyze each tweet sentiment using **TextBlob**[6] library, which is used for processing textual data, specifically using its sentiment.polarity method to get it's polarity which lies in the range of -1 to 1. Through its polarity, we could know if the tweet has a positive (polarity above 0), neutral (polarity 0) and negative (polarity below 0) sentiment. The function **AnalysingSentiment()** was applied to the cleaned tweets and 2 new columns of TweetPolarity and TweetSentiment were made.

After the data frame was completed, we call the **DashboardApp()** function with the data frame as its input. It would create a dashboard app that showed the visualization of the analyzed data. The visualizations were made using **Plotly**[7] which is an interactive, open-source plotting library which would be linked to **Dash**[8], which is an open source Python library for creating reactive web-based applications. I chose using Plotly instead of Matplotlib because embedding Matplotlib graphs to other applications like Dash would be more tedious and Plotly visualizations looked more appealing and interactive than Matplotlib's.

I am particularly interested to know the sentiment distribution of all tweets (in percentage), the distribution of the tweet's sentiment in each day for the period User stated, and the 20 most frequent words with its frequency. Before creating the charts, I collected some data specifically to answer each of the questions. **OverallSentiment(),** with data frame as its input, prepared data about number of total tweets in each sentiment group (positive, neutral and negative). **LineChart()**, with data frame as its input, prepared data about number of total tweets for each sentiment group on each date (grouped by the dates). **FrequencyDist()**, with CleanedTweet column as its input, prepared a list of the 20 most common words and each of its frequency.

DashboardApp() would initialize Dash by calling the Dash class of dash. Then I created a simple layout for the application. I used dash_html_components to generate HTML components such as H1, H2 etc. Then using Graph class from dash_core_components, I created graphs (pie, line and bar chart)[9] using data from the functions prepared before. The charts layout could be modified further by creating titles, changing its component colors, etc. Last, the program would run the web server to view the visualizations. Note that when the function was called, it would generate a link: http://127.0.0.1:4050/. If the web does not pop up, User could just enter the link shown. Whenever, User run **MainProgram()** function, he would need to refresh the web link mentioned.

In my opinion, this project can be improved further if real time data, which can be collected in a database, are used, so that the dashboard can be updated every minute. More problems can also be answered by analyzing real time data, for example how the sentiment analysis differs based on location, how the sentiment changes every minute, how every Users are connected, etc. I tried to make an analysis of location, but data about location was insufficient as not every User tags his location in his tweets and it was quite complicated to extract a specific city or country from the data available. The dashboard can also be improved by making it more interactive for Users, for example by asking User to specify his preference of keywords, dates and number of data in the dashboard instead.

---

[4] https://stackoverflow.com/questions/33404752/removing-emojis-from-a-string-in-python

[5] https://www.nltk.org/

[6] https://textblob.readthedocs.io/en/dev/advanced_usage.html#sentiment-analyzers

[7] https://plot.ly/python/getting-started/

[8] https://pythonprogramming.net/data-visualization-application-dash-python-tutorial-introduction/

[9] https://www.youtube.com/watch?v=QJPN2J_KGXI&t=354s